Sammendrag

I denne hovedrapporten vil leseren få innsikt i hvordan vi i samarbeid med Atea IRT har utviklet et produkt som overfører mange av de manuelle prosessene som teamet utfører i dag, til et system som på en effektiv og pålitelig måte håndterer både automatisk innhenting av sikkerhetsfeil og et brukervennlig grensesnitt for å behandle informasjonen som kommer inn.

Som gruppe har vi tidligere hatt erfaring med å utvikle full-stack applikasjoner med mye funksjonalitet, men ikke på langt nær så omfattende og innholdsrik som det AIRS 1.0 kom til å bli. Gjennom prosessen har vi lært å bli mer fleksible rundt hvordan vi velger å løse enkelte oppgaver, og har måttet omstille både tankegangen og fremgangsmåten å løse ulike oppgaver på.

Ved å følge en løs scrum-metodikk har vi i tillegg fått erfare hvor mye god kommunikasjon kan påvirke samarbeidet mellom de forskjellige partene i prosjektet, og hvordan det kan bidra til at alle får sagt sitt under utviklingsprosessen.

For å underbygge valgene vi har tatt og funksjonaliteten vi har implementert har vi brukt ulike kvalitative forskningsmetoder for å forske på både industristandarder og mindre kjente metodikker. På denne måten har vi fått innsikt i både standardiserte måter å gjøre ting på, samtidig som vi har sett på mer alternative løsninger som også har fungert godt for våre formål.

Du kommer til å lære om hvordan man gunstig kan lagre brukerdata gjennom hashing og kryptering, sikker autentisering av brukere ved hjelp av en rekke autentiseringsmetoder, samt hvordan man kan sørge for effektiv og pålitelig drift av systemet ved bruk av microservices, containerisering, tråd-optimalisering og andre metoder som bidrar til et mer robust system.

Den teoretiske delen av hovedrapporten tar opp viktige begreper som blir referert til eller brukt gjennom resten av rapporten, og hjelper leseren med å få en mer helhetlig forståelse for ulike temaer som blir gjennomgått etter hvert som vi utforsker hvert forskningsspørsmål.

Leseren blir også presentert med resultatene vi har kommet frem til, og en grundig diskusjon på hvorfor vi valgte å gjøre ting som vi gjorde, og hvordan ting kunne blitt gjort annerledes.

Mot slutten av hovedrapporten konkluderer vi med å skissere fremtidige implementasjoner av systemet, da oppdragsgiver har uttrykt både interesse for og ønske om å videreutvikle produktet etter endt bacheloroppgave.

Abstract

In this main-report the reader will get a good look into how we, in cooperation with Atea IRT, managed to develop a product that transforms many of the manual tasks they're performing today - into a system that effectively and reliably handles both automatic retrieval of security-issues and a user-friendly interface for interacting with the information on the platform.

As a group we have had experience with developing full-stack applications with a lot of functionality, but nothing comes close to the rich amount of content that would be applied to AIRS 1.0. Throughout the process we have learned to become more flexible around how we choose to solve different tasks, while having to completely rewire both our thought-process and approach to solving these tasks in a satisfying manner.

By following a loosely-based scrum-methodology we have also had the chance to experience exactly how much communication can affect the cooperation between the different stakeholders in the project, and how it can contribute to everyone having a say in what happens during the development of the product.

To substantiate the choices we have made and the functionality that we have implemented, we have used different kinds of qualitative research processes to learn more about both industry-standards and lesser known methodologies. In this way, we have had a chance to get a good look into both standardized ways of doing things, and also learning more alternative methods for solving these same tasks, which turned out to work great for our purposes.

You will learn how to conveniently store user-data through hashing and encryption, secure authentication of users by utilizing different kinds of authentication-methods, and also how to ensure an effective and reliable system through the use of microservices, containerizing, thread-optimisation and other methods which contributes to a more robust system.

The theoretical part of the main-report brings up important concepts that are referred to and used throughout the rest of the report, and helps the reader with obtaining a more broad understanding of the different themes that are reviewed during our run-through of the different research-questions.

The reader will be presented with the results that we have produced, and a thorough discussion on why we chose to solve things the way that we did, and what could have been done differently.

Towards the end of the main-report we conclude our work by presenting future implementations of our system, based on both the interest and wishes from our task-provider for the product after the end of our bachelor thesis.

Forord

Valg av oppgaven

Helt i begynnelsen av prosjektet, før vi i prosjektgruppa i det hele tatt hadde valgt oppgave, ønsket vi å se om vi klarte å skaffe en interessant ekstern oppgave selv. Dette ble delvis gjort fordi vi ønsket oss en systemutviklings-oppgave, og ingen av oppgavene skolen stilte med var noe vi anså som interessant. Dermed så formulerte vi en e-post som vi sendte ut til en del store IT-firma i Trondheim og omegn. En av respondentene til denne e-posten var Vegard Kjerstad, som virket engasjert og kom med en oppgave vi i prosjektgruppa så på som spesielt interessant da det gikk ut på å utvikle et system som kunne komme til nytte for deres arbeid i hverdagen.

Prosess

Prosessen som ledet fram til resultatene og konklusjonene fulgte ingenørfaglige prosesser som vi lærte underveis i studiet *Bachelor i ingeniørfag, Data* ved NTNU. Disse prosessene innebærer blant annet Smidig utviklingsmetodikk, ingeniørfaglig systemtenkning og generell kunnskap om programmering lært underveis i studiet. Flyten av de ulike prosessene kan man se på i *vedlegg E Gantt-Diagram-127.*

Takk til:

Olav Skundberg fra NTNU:

- Veileder under hele prosjektet
- Bidro med god veiledning angående alle prosessene vi gikk igjennom i løpet av hele prosjektet basert på tidligere erfaringer og fornuft. Dette gjelder alt fra helt i starten av forprosjektsplanen til helt på slutten av hovedrapporten.

Vegard Kjerstad fra Atea:

- Oppgavestiller
- Ga oss en interessant bacheloroppgave som vi kunne bryne oss på, og ellers god kunnskap rundt hvordan systemet bør implementeres for å være anvendbart.

Atea Incident Response Team:

- Spesielt: *Geir Olav Skei.*
- Brukergruppen til produktet.
- Bidro med kjempegod kunnskap rundt hva som bør være med i systemet og hvordan det bør implementeres for at de skal få nytte av det.

Morten Schjetne fra Atea:

- Ga oss tilgang til de flotte kontorlokalene til Atea på Ranheimsvegen 9 slik at vi kunne samarbeide fysisk, og fulgte delvis med på fremgangen underveis i prosjektet.

Deltakere i prosjektgruppa med underskrifter

		
Scott Rydberg Sonen	**Endré Hadzalic**	**Odin Kvarving**

Oppgavetekst

Opprinnelig oppgavetekst

Hensikt

Lage et system som kan varsle registrerte kunder om eventuelle sikkerhetsrisikoer som gjelder dem

Hensikten med oppgaven er å lage et system som gjør det enklere for Atea sitt Incident Response Team å få en oversikt over nye sikkerhetsfeil som angår kundene deres. Systemet skal også kunne brukes til å sende ut varslinger om alvorlige sikkerhetsfeil.

Kort beskrivelse

Skal omfatte en system som har oversikt over ulike teknologier som kunder er registrert med, og ha kontinuerlig oversikt over sikkerhetsrisikoer ved de aktuelle systemene. Varsling og informasjon skal sendes ut fortløpende, og alt av informasjon skal krypteres og sikres tilstrekkelig.

Atea Incident Response System (AIRS) er et sikkerhetssystem laget for Atea sitt Incident Response Team, som automatisk henter inn nye sikkerhetsfeil fra pålitelige nettsteder, RSS-feeds og Atom-feeds. Inne på systemet kan brukerne filtrere, sortere og redigere informasjon, samt sende ut varsling om alvorlige sikkerhetsfeil som blir funnet av systemet.

Detaljert beskrivelse

En portal eller applikasjon som vi har oversikt over kunder i, de kundene har igjen en rekke teknologier / produkter. Så overvåkes/innhentes informasjon fra åpne kilder om sårbarheter tilknyttet teknologi / produkter. En kunde kan ha mange teknologier /produkter. og en teknologi/produkt kan finnes hos mange kunder. Basert på dette kan vi da ta ut oversikt over kunder som berøres av en bestemt sårbarhet. og ideelt sett sende et varsel til kundelisten om at de bør ta grep. Det hadde også vært fint å sende ut custom alarmer/info til samme kundelistene. Oversikt over kunder må sikres og spesielt hvilken teknologi de ulike har bør ikke komme ut. Tilgangsstyring slik at kun autorisert personell kan få tilgang må være implementert.

Tolket oppgavetekst

En liten oppsummering av hvordan gruppa tolket oppgaveteksten som ble beskrevet over står beskrevet under, mens en mer detaljert gjennomgang av dette er beskrevet i vedlegg F (visjonsdokument) (Hadzalic et al., 2022).

Oppgaven består i å lage en plattform som i hovedsak tar for seg innhenting av nye sikkerhetsfeil, svakheter og utnyttelser for teknologier i en fastsatt kundebase. Hvis det blir registrert en ny sikkerhetsfeil som kan påvirke en kunde, skal man enkelt kunne kartlegge hva feilen består i og hvilke teknologier den påvirker.

Innhenting av denne informasjonen skal foregå ved å overvåke en rekke nettsteder fra både utviklere av de gitte teknologiene og forumer/tredjepartsprogrammer som gir ut slik informasjon.

Sikker behandling av kundedata skal stå sentralt, da den inneholder informasjon om alle kundene til Atea, og hvilke teknologier som blir brukt. Usikker håndtering av slik data kan føre til at en angriper får tilgang til svært sensitiv informasjon, så håndtering av denne dataen skal utføres nøye.

Plattformen skal først og fremst være et hjelpemiddel for Atea for å gjøre det enklere å innhente nye sikkerhetsfeil og knytte dem opp mot spesifikke teknologier, noe som tidligere har vært en veldig manuell prosess. Hvis det blir tid, og plattformen egner seg til det - kan den også utvides til et platform hvor kundene til Atea kan logge seg inn, få oversikt over nye sikkerhetsfeil og se nye varslinger på egenhånd.

Innholdsfortegnelse og ev. figur- og tabelliste

Kapittel 1: Introduksjon og relevans

I dette kapitlet skal vi først ta for oss hovedrapportens struktur og alle grunnsteiner i hovedrapporten vil bli introdusert. Deretter blir akronymer og forkortelser presentert. Videre beskriver vi oppdragsgiver og deres behov. Problemstillingen som er knyttet til prosjektet vil også bli presentert. Her vil vi basere vår problemstilling på oppdragsgivers behov samtidig som vi bestemmer hvilke prioriteringer vi har. Prosjektets forskningsspørsmål blir også beskrevet.

1.1 Hovedrapportens struktur

Hovedrapporten består av relevante deler som primært er knyttet opp mot produkt, problemstilling og forskningsspørsmål.

Relevant teori for problemstillingen som brukes underveis i prosessen og i denne hovedrapporten skal redegjøres for. Dette omhandler hvilke teknologier som er brukt i tilknytning til koding, litteratur knyttet til besvarelse av forskningsspørsmål, og annen relevant teori.

Metode blir beskrevet i kapittel 3. Her presenterer vi både forskningsmetoden vi har brukt, valg av teknologier og verktøy, samt hvilken utviklingsmetode vi har brukt. Forskningsmetoden baserer seg på forskningsspørsmålene våre og hvordan vi går frem for å løse dem. Utviklingsmetode og valg av teknologier gir en oversikt over den overordnede fremgangsmåten vi har brukt, og hvilke teknologier som har blitt brukt for å produsere produktet i oppgaven.

Resultater-kapittelet tar for seg vitenskapelige, ingeniørfaglige og administrative resultater. Disse resultatene baserer seg på svar vi har funnet knyttet til problemstilling, forskningsspørsmål, prosess og system.

Diskusjon kommer etter resultatene, og er hvor det reflekteres og drøftes rundt de. Denne delen skal handle om årsaker til at resultatene ble som de ble.

Konklusjon og videre arbeid handler om konklusjoner tatt rundt forskningsspørsmålene, produktet og videre arbeid. Konklusjonene er gjort i utgangspunktet av resultater knyttet til hvert forskningsspørsmål og problemstillingen, samt produktet i seg selv. Når det gjelder videre arbeid er dette gjerne oppgaver og punkter som ikke ble prioritert eller som ikke ble gjennomført på grunn av tidspress.

Samfunnspåvirkning kommer til slutt. Dette kapittelet knytter oppgaven opp mot samfunnspåvirkning, og prøver å påpeke aspekter ved oppgaven som kan være med på å påvirke valg og løsninger som blir brukt av andre IT-selskaper når de utvikler et nytt system.

Ved å dekke opp under alle disse punktene inneholder hovedrapporten alle nødvendige og relevante deler av prosjektet og viser prosessen vi har gjennomgått for å komme til resultatene og konklusjonene vi har fått.

1.2 Akronymer og forkortelser

Forkortelse	Betydning	Kort forklaring
AIRS	Atea Incident Response System	Det automatiserte systemet vi har laget. Vi har brukt navnet til Atea Incident Response Team som en del av navnet på produktet.
IRT	Incident Response Team	En gruppe i Atea som holder orden på kunders sikkerhet. Teamet har blant annet som oppgave å varsle kunder ved sikkerhetsfeil som relateres til deres teknologier. Er også brukergruppen til AIRS.
RDBMS	Relational Database Management System	Vedlikeholder relasjonsdatabaser og bruker tabeller til å håndtere og lagre data (Brush, n.d.).
API	Application Programming Interface	En kode som brukes for å utveksle data mellom to forskjellige systemer eller apper. Hjelper systemene å automatisk kommunisere med hverandre – uten at mennesker er involvert.
IDE	Integrated Development Environment	På norsk betyr dette et integrert utviklingsmiljø. Dette er en type programvare eller plattform som kan brukes til å utvikle annen programvare. I en IDE kan man skrive og teste kode (Wikipedia, n.d.).
JVM	Java Virtual Machine	JVM er en virtuell (abstrakt) maskin som tilbyr et kjøremiljø hvor java-kode kan kjøres (Javatpoint, n.d.).
JSON	JavaScript Object Notation	JSON er en forkortelse for JavaScript Object Notation og er et tekstbasert format for overføring av data over nettet. JSON-objekter kan inneholde mye informasjon og data, men er likevel enkle å håndtere, og enkle å lese og skrive. JSON er også en av de mest foretrukne syntaksene for APIer, da alle programmeringsspråk kan ta i bruk dette formatet (JSON, n.d.).

CVE	Common Vulnerabilities and Exposures	Klassifiseringer av sikkerhetsfeil.
ORM	Object Relational Mapping	ORM er en teknikk som gjør at man enkelt kan utføre spørringer og manipulere data mot en database ved hjelp av objekt-orientert programmering
JPA	Java Persistence API	JPA er Java sin ORM-standard for å lagre, få tilgang til og håndtere Java-objekter i en relasjonsdatabase

Tabell 1.2.1: Tabellen inneholder akronymer og forkortelser, betydningen deres, og en kort forklaring.

1.3 Oppdragsgiver

Vår oppdragsgiver er Vegard Kjerstad, leder av Atea Incident Response Team (IRT). Et av våre ønsker for bacheloroppgaven var å samarbeide med en ekstern bedrift. Vi hadde allerede før valg av oppgave bestemt oss for at fullstack-applikasjonsutvikling og sikkerhet var to temaer vi gjerne ville jobbe med. Etter en søknadsprosess, kom vi i kontakt med Atea og deres IRT ledet av Kjerstad.

Atea-konsernet er et IT-selskap som tilbyr tjenester og løsninger til sine kunder. Bedriften opererer i hele Norden og Baltikum (Atea, n.d.). Atea IRT er Ateas sikkerhetsteam. En av deres oppgaver er å varsle sine kunder om relevante sikkerhetsfeil som blir oppdaget. Deres nåværende løsning baserer seg på manuell innhenting og varsling av slike sikkerhetsfeil.

1.4 Behov

Som leder av Atea Incident Response Team (IRT), hadde Kjerstad og teamet hans ønske om å automatisere systemet deres for innhenting og varsling av nye sikkerhetsfeil. Behovet for et automatisk system var til stede, da Atea IRT praktiserte manuell innhenting og varsling til sine kunder, dersom nye sikkerhetsfeil skulle oppstå. Dette er både tidkrevende og kan føre til lengre responstid for Ateas kunder. Systemet må dermed kunne hente inn sikkerhetsfeil fra alle eksisterende kilder som Atea IRT bruker i dag, og presentere dem på en slik måte at det blir oversiktlig og enkelt å behandle. Det skal også være mulighet for å sende ut en varsling til berørte kunder direkte inne på systemet, hvis tid og ressurser tillater det.
Det må være mulighet for å knytte Atea IRTs kunder og teknologier opp mot de forskjellige sikkerhetsfeilene, slik at man enkelt kan finne ut hvilke kunder som blir berørt av sikkerhetsfeilen. Filtrering, sortering og redigering av kunder, teknologier og sikkerhetsfeil må være mulig, siden man enkelt skal kunne håndtere data som er inne på systemet. Brukergrensesnittet må være enkelt å bruke, og det skal også være mulighet for å sende ut en varsling til berørte kunder direkte inne på systemet dersom annen funksjonalitet er på plass.

For en mer detaljert beskrivelse av krav og mål for oppgaven, se vedlegg F - Visjonsdokument.

1.5 Problemstilling

Problemstillingen vår går ut på å lage et pålitelig system som gjør det enklere for Atea IRT å få en oversikt over nye sikkerhetsfeil som angår kundene deres. Denne delen av systemet skal være automatisert, og skal brukes for å unngå unødvendig ressursbruk på manuell innhenting. Det skal også være mulig å sende ut varslinger til kunder som blir berørt av innhentede sikkerhetsfeil, dersom annen viktig funksjonalitet fungerer som den skal. Denne problemstillingen relaterer til effektmål nummer 1 i Vedlegg D - Forprosjektplan.

I et slikt system er det implisitt at det skal være så sikkert og pålitelig som mulig. Vi har derfor lagt stor vekt på sikker behandling av kundedata og bruk av systemet i sin helhet. For å fastslå hvordan man skal oppnå et slikt system har vi derfor undersøkt hvordan det er gunstig å lagre brukerdata og hvilke metoder- og/eller systemer som er best egnet til sikker autentisering av brukere på forskjellige deler av systemet. For at systemet skal være så robust som mulig, ville vi også undersøke ulike løsninger for å gjøre applikasjonen så effektiv og pålitelig som mulig. Til slutt undersøkte hvilke effekter bruken av dette systemet fikk sammenlignet med det tidligere manuelle systemet. Dette samsvarer med effektmål nummer 2 i Vedlegg D (Kvarving et al., 2022).

Basert på dette ender vi opp med følgende problemstilling:

Hvordan sikre pålitelig, oppdatert og automatisk innhenting av sikkerhetsdata, samt sørge for sikker lagring av kundedata og autentisering av systemets brukere.

1.5.1 Forskningsspørsmål

	Forskningsspørsmål	Utdypende beskrivelse
1	Hvordan er det gunstig å lagre brukerdata?	Her ønsker vi å finne svar på hva som er tilstrekkelig sikker lagring av passord eller annen sensitiv informasjon i systemet. Dette inkluderer blant annet å definere hva som er sensitiv informasjon, og hvordan det bør lagres for at det skal kunne ansees som tilstrekkelig sikkert i forhold til moderne standarder for lagring av data.
2	Hvilke metoder er best egnet for sikker autentisering av brukere på forskjellige deler av systemet?	Det vi ønsker å svare på er hvilken måte kan vi kan identifisere brukere av systemet på en sikker måte, og kun gi identifiserte brukere tilgang til bruk av systemet. Vi ønsker også å se på hvordan dette kan gjøres på en så brukervennlig måte som mulig, samtidig som det skal kunne anses som tilstrekkelig sikkert i forhold til moderne standarder for brukerautentisering.
3	Hvordan sikre effektiv og pålitelig	Her er hensikten å finne ut av hvilke

	drift av systemet?	teknikker eller metoder som kan benyttes i produktet for å oppnå best mulig ytelse og oppetid under drift av systemet. Dette kan være teknikker som innebærer bedre ressursutnyttelse, isolering av ulike funksjoner og prosesser, og andre måter å gjøre systemet mer robust på.
4	Hvilke effekter gir bruken av det nye systemet sammenlignet med tidligere manuelt system?	Dette forskningsspørsmålet omhandler en sammenligning mellom det gamle systemet til Atea IRT og AIRS. Hvor sikre er metodene som blir brukt av Atea i dag, og hva kreves for å tilstrekkelig sikre denne dataen? Er det nye systemet mer oversiktlig? Blir innhenting av sikkerhetsfeil en mindre tidkrevende prosess? Slike sammenligninger vil hjelpe oss med å besvare dette spørsmålet.

Tabell 1.5.1.1: Tabellen inneholder våre forskningsspørsmål, med selve spørsmålet og en beskrivelse av det.

Kapittel 2: Relevant teori for problemstilling

Dette kapitlet beskriver den teoretiske bakgrunnen til de forskjellige teknologiene eller metodene som ble benyttet under prosjektet. Kapittelet er delt inn i flere underkapitler, hvor hvert punkt dekker relevant teori for ett forskningsspørsmål. Det siste underkapitlet dekker systemteori, som er knyttet til andre deler av systemet vi har produsert. Dette er teori som er nødvendig for at systemet skal fungere på slik måte at det løser problemstillingen vår, men ikke nødvendigvis kan knyttes direkte opp mot et forskningsspørsmål. Forskningsspørsmål fire blir ikke dekket i dette kapitlet, siden det ikke finnes relevant teori til dette spørsmålet.

2.1 Sikker lagring av kundedata

2.1.1 Relasjonsdatabaser

RDBMS (Relational Database Management System) er en type database som lagrer- og tilbyr adgang til datapunkter som er relatert til hverandre. Relasjonsdatabaser er basert på en relasjonell modell, hvor data er representert i tabeller. Hver rad i tabellen representerer et datapunkt med en unik ID, kalt en primærnøkkel. Kolonnene i tabellen inneholder attributter for datapunktene, og er som oftest representert av en eller annen verdi. For å skape en relasjon mellom de ulike databaseobjektene bruker man noe som kalles en fremmednøkkel. Dette er en unik identifikator (ofte en primærnøkkel) for et annet databaseobjekt som settes inn i tabellen man ønsker en kobling i. (Oracle, n.d.) Her kan man endre strukturen på relasjonen mellom tabellene ved å enten definere en OneToOne, OneToMany, ManyToOne eller ManyToMany-kobling. Det eneste som endres ved de forskjellige relasjonene er hvor fremmednøkkelen(e) blir plassert; i den ene tabellen, i begge tabellene eller i en egen relasjonstabell. Denne strukturen gjør det enkelt å etablere forhold mellom ulike datapunkter, og er foretrukket blant mange som bruker databaser under utvikling.

Den relasjonelle modellen tilbyr altså en standardisert måte å representere- og utføre spørringer på data, som enkelt kan bli implementert av enhver applikasjon. Relasjonsdatabaser er også en av de raskeste og mest konsistente databasetypene på tvers av instanser eller applikasjoner som bruker databasen, noe som tilbyr nesten umiddelbar respons på endringer i databasen (Oracle, n.d.).

2.1.2 Hashing og Salting

Hashing kan forklares som en teknikk eller prosess hvor hensikten er å generere en ny verdi for en input, ved hjelp av en matematisk algoritme (Educative, n.d.). En god og pålitelig hashing-funksjon vil gjøre det umulig å gå tilbake til den gamle, ikke-hashede verdien. De beste funksjonene er derfor enveis-funksjoner. I følge et blogginnlegg skrevet av eksperter fra UiO angående kryptering av konfidensiell data er det tre punkter som er viktig når man skal velge krypteringsmetode; den må være gjennomanalysert og sjekket for svakheter, være beregnings-tung for CPU/GPUer og må benytte saltede avtrykk. (Titan UiO, 2016)

Resultatet av en hashing-funksjon kalles ofte for en *hash-verdi* eller *hash*. Det skal ikke være mulig å kunne finne ut hvor en hash stammer fra, i form av hvilke data og nøkler som har vært brukt. Skal man sammenligne to verdier er det derfor hashene som må

sammenlignes. Et eksempel er når en bruker skal logge seg inn i et system. Dersom brukeren allerede er lagret i en database, vil man ha behov for å sammenligne passordet som ligger inne i databasen med passordet som blir skrevet inn av brukeren. Uten hashing er dette en enkel jobb, da man sammenligner verdiene direkte uten noe mellomsteg. For å skape et sikkert og pålitelig system må man hashe brukerpassordet. Sammenligningen av verdiene vil være en suksess dersom for eksempel hashen til det registrerte passordet er lik hashen til innloggingspassordet. På denne måten kan hashing brukes til autentisering og validering av data (Constantin, 2021).

Et salt er en unik og tilfeldig generert tegnstreng som blir lagt til på hvert passord som en del av hashing-prosessen. Siden saltet er unikt for hvert passord, er en potensiell angriper nødt til å knekke hashene én av gangen ved bruk av dets respektive salt istedenfor å kalkulere hashen én gang og sammenligne den med hver eneste lagrede hash. Dette gjør at prosessen ved å knekke et større antall hasher blir særdeles mye vanskeligere, da tiden det tar å knekke dem øker direkte proporsjonalt med antallet hasher som skal knekkes.

Salting beskytter også mot at en angriper kan forhåndsberegne hasher ved hjelp av "rainbow-tables" eller database-baserte oppslag. Rainbow-tables er en database som inneholder tabeller av alle mulige kombinasjoner av enveis hasher på klartekst-passord fra lengde 1 til 7 eller 8. Hvis en angriper da får tak i en cache med enkle passord-hasher, kan de bruke oppslag på hash-verdien i et rainbow-table for å finne klartekst-passordet. database-baserte oppslag fungerer ved å internt få tilgang til spesifikke deler av databasen, og utnytte informasjonen man finner for å knekke hasher. Begge disse metodene er nytteløse om man utnytter salting. Moderne hashing-algoritmer som Argon2id, bcrypt og PBKDF2 salter automatisk passordene, så her trenger man ikke å tenke på saltingen i det hele tatt (OWASP, n.d.).

2.1.3 Kryptering

Kryptering er en matematisk metode som sørger for konfidensialitet ved at informasjonen ikke kan leses av uvedkommende. Informasjonen "låses" med en nøkkel og kan ikke leses før man har låst den opp igjen med riktig nøkkel. (Datatilsynet, n.d.)

Det finnes to former av kryptering; symmetrisk og asymmetrisk kryptering. Symmetrisk kryptering innebærer at man bruker samme nøkkel for å både kryptere og dekryptere data, mens man i asymmetrisk kryptering bruker både en privat og en offentlig nøkkel til å henholdsvis kryptere og dekryptere.

Disse to formene for kryptering har ulike funksjoner og bruksområder som gjør at de blir brukt til å sikre forskjellige typer data. Asymmetrisk kryptering er for eksempel mye tregere enn symmetrisk kryptering, og brukes derfor ofte til å kryptere bare små mengder med data. Det er også vanlig å bruke asymmetrisk kryptering under overføring av data, mens symmetrisk kryptering brukes på statisk data. Dette er på grunn av egenskapen ved at man har både en privat og en offentlig nøkkel, som gjør den ideell for å dele informasjon over lengre distanser uten at noen kan avlytte eller stjele data underveis. Symmetrisk kryptering egner seg bedre hvor man vet at dataen blir liggende, som i en database eller på en kryptert fil. (ssl2buy, n.d.)

2.2 Sikker autentisering av brukere

2.2.1 Elektronisk autentisering

En digital signatur i kryptografi er brukt for å verifisere autentisiteten av en melding eller data (Google, n.d.). JSON Web Token, også kjent som JWT, er et eksempel på en slik form for digital signering. En JWT-token er en kjent internett-standard brukt for sikker deling av data mellom to parter, gjerne en klient og en server. Informasjonen er pålitelig og kan verifiseres. En slik token vil signeres ved bruk av en kryptografisk algoritme (JWT, n.d.).

Hver token inneholder et JSON-objekt og et sett med påstander. JWT-tokenet blir generert ved hjelp av en kryptografisk algoritme, som sørger for at disse påstandene ikke kan endres etter utstedelse av tokenet (JWT, n.d.).

JWT kan brukes i sammenheng med autorisering av en bruker. Når en bruker logger inn i en applikasjon eller et nettsted, vil denne brukeren kunne gjenbruke JWT-tokenet i en fastsatt tidsperiode, slik at brukeren har tilgang til forskjellige endepunkter og ruter innad i applikasjonen eller nettstedet (JWT, n.d.).

Måten en JWT-token blir tildelt på, er ved at en bruker logger seg inn med riktig påloggingsinformasjon, som blir verifisert ved å sammenligne verdiene til en allerede registrert bruker. Hvis autentiseringen går gjennom, tildeles man en unik JWT-token. Grunnen til at denne må være unik er for at ingen brukere skal ha samme JWT-token samtidig, da det er denne som bestemmer hvilke deler av nettsiden man har tilgang til. Får man for eksempel tildelt samme JWT-token som en administrator som bruker systemet, har man plutselig tilgang til alle deler av systemet som en administrator har.

Gjennom sikker oppretting av en JWT-token vil man unngå slike problemer, og sørger for at brukerens JWT-token sjekkes opp mot registrerte privilegier hver gang den utfører et kall til serveren. Dersom en angriper prøver å navigere til en side i applikasjonen gjennom et kjent endepunkt, vil ikke angriperen bli sendt til denne siden før vedkommende har autentisert seg med en JWT-token, og sjekket at brukeren har privilegier til å gå til denne siden. JWT-token er derfor viktig for å skape sikker brukerautentisering, både for å verifisere identiteten til brukeren og skjerme nettstedet fra potensielle angripere.

2.3 Pålitelig drift av systemet

2.3.1 Applikasjons-containerisering

Applikasjons-containerisering omhandler det å "isolere" en applikasjon fra resten av prosessene som kjører på et operativsystem gjennom programvare. Man kan altså si at applikasjonen lever i sin egen "container" innad i verts-operativsystemet. Dette innebærer at applikasjonen har sin egen kjøretidsmotor som typisk er en forenklet versjon av et et vilkårlig operativsystem. Videre har også applikasjonen med seg alle avhengigheter til annen programvare som den benytter inn i containeren. Dette kan for eksempel være programvarebiblioteker, rammeverk eller drivere (IBM, 2021).

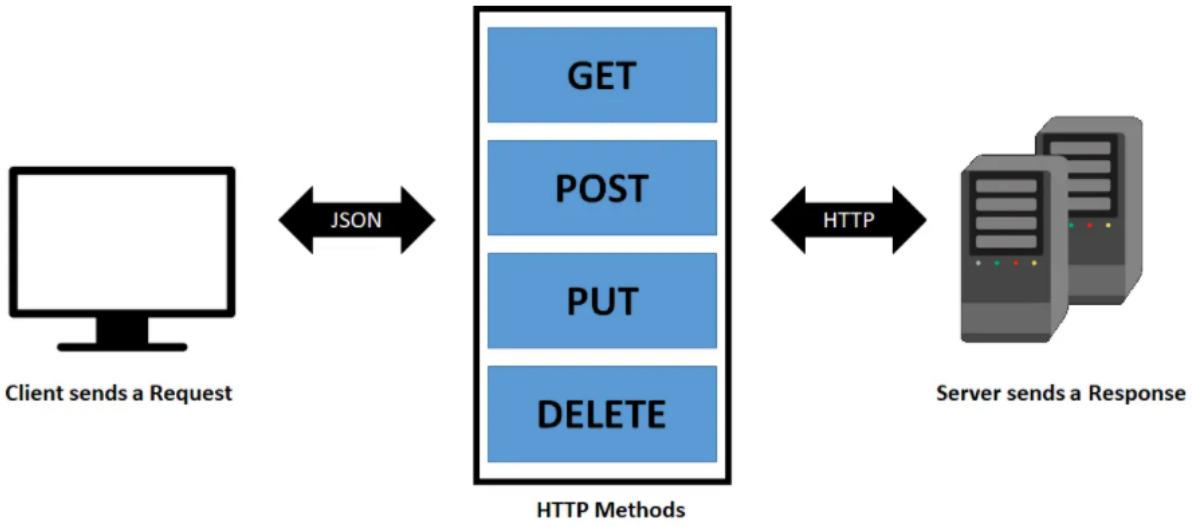
Bruk av applikasjons-containerisering gir fordeler som vises både under utvikling av applikasjonen og etter distribuering. Under utvikling av en applikasjon så er det fordelaktig å benytte containere da dette sikrer at applikasjonen sitt kjøretidsmiljø under

utvikling er det samme som etter distribuering, siden applikasjonen kjører i samme container i begge scenarioene. Dette gjør at utviklere som programmerer applikasjonen kan benytte de operativsystemene, verktøyene eller prosessene de er mest komfortable med uten at det påvirker kjøretiden til applikasjonen etter distribuering. Andre fordeler som i hovedsak gjelder *etter* distribuering av en applikasjon innad i en container er at eventuelle virus eller annen ondsinnet kode som kjører innad i containeren aldri vil kunne påvirke verts-systemet som containeren kjører på, da tilgangen til innholdet i containeren er begrenset til nettopp containeren.

2.4 Systemutviklings-teori

2.4.1 RESTful API

REST står for Representational State Transfer, og er en enkel måte å sende og motta data mellom klient og tjener. Denne typen kommunikasjon krever ingen ekstra programvare eller standarder for å overføre data, og bruker en forhåndsdefinert struktur for å utføre API-kallet. Et API-kall er altså en spørring til en tjener om at et API skal gi informasjon eller en tjeneste tilbake til klienten som spør. (Gocoding, 2019) I tilfellet ved RESTful er dette som oftest snakk om Http-spørringer (Get, Post, Put, Delete) hvor datautvekslingen skjer ved hjelp av JSON-objekter. Her bruker klienten et spesifikt endepunkt (URL), velger hvilken type spørring som skal sendes til tjeneren, og legger ved et JSON-objekt som passer inn i et format som serveren forstår. Hvis klienten sender en gyldig spørring, vil tjeneren svare med en respons som passer til spørringen.



Figur 2.1.1.1: RESTful-prosessen. (Hentet fra Go Coding (Go Coding, 2019))

Man kan også legge ved headere på spørringen for å gi ekstra detaljer om kommunikasjonen mellom klient og tjener. Dette er fordi REST er tilstandsløst, og lagrer ikke noe informasjon om de ulike spørringene. Svaret som tjeneren sender tilbake til klienten inneholder en status, content-type og set-cookie. Disse sier noe om spørringen gikk gjennom og fikk en gyldig eller ugyldig respons, hvilken type data som blir sendt tilbake og eventuelle cookies for kommunikasjonen. Cookies er en type data som sendes ved i en respons fra tjener, og kan brukes til å spesifisere ekstra parametere for kommunikasjonen - litt slik som headere. (Red Hat, 2020)

For at noe skal kunne klassifiseres som en RESTful webtjeneste, er den nødt til å inneholde seks begrensninger/attributter:

1. Følger en klient/tjener-arkitektur
2. Er tilstandsløs
3. APIet oppmuntrer nettleseren og serverens caching-prosess til å effektivisere kommunikasjonen mellom klient og tjener.
4. Grensesnittet mellom klienten og tjeneren forblir uniformt, altså påvirker ikke endringer på klient -eller tjenersiden selve API-funksjonaliteten
5. Bruker et lagdelt system, hvor man for eksempel kan ha data på én server, autentisering på en annen server og selve APIet på en tredje server. Klienten vet ikke hvilken data den får fra hvilken server.
6. Skal ha mulighet til å sende kjørbar-kode fra tjener til klient når det blir forespurt, som bidrar til ekstra funksjonalitet til klienten.

2.4.2 Web-scraping

Generelt sett så er web-scraping et begrep som angår det å gå inn på internett og hente ut informasjon fra nettsteder. Dette kan gjøres ved å direkte benytte *HTTP-protokollen* eller igjennom nettlesere (Wikipedia, 2022). Måten det gjøres på kan enten være ved at et menneske gjør det manuelt, eller at det gjøres automatisk ved hjelp av en bot eller en *webcrawler* (en bot som navigerer fra side til side på internett ved hjelp av linker funnet på sidene) (Wikipedia, 2022). Begrepet benyttes i all hovedsak nå i moderne tid for automatiserte prosesser som kopierer data fra nettsteder og lagrer dataene inn i sentraliserte databaser eller regneark, noe som gjøres for å senere kunne hente ut eller analysere dataen.

2.4.3 RSS og Atom-feeder

Really Simple Syndication eller RSS er et simpelt og standardisert format som hjelper nettsteder som endrer innhold jevnlig med å enkelt publisere det nye innholdet. De kan også brukes som et verktøy for lesere å enkelt for oversikt over nye endringer. Atom-feeder er en modernisering av RSS, til tross for at de teknisk sett er konkurrenter.

Figur 2.4.3.1: Eksempel på ikonet som representerer RSS eller Atom feeder (Hentet fra (Brett, n.d.))

Nettsteder for nyheter eller blogger er eksempler på nettsteder hvor RSS eller Atom-feeder typisk anvendes. Grunnen til at man vil ha et slikt informasjonsmedium som publiserer ny informasjon i et standardisert format kan være at man kanskje ønsker å holde seg oppdatert på flere forskjellige nettsider eller blogger samtidig. På denne måten kan man enkelt abonnere til de forskjellige kildenes RSS eller atom-feeder, og få oppdatert informasjon på et fast format. Siden innholdet fra de forskjellige feedene er standardisert vil dette si at oversikten over informasjon ser lik ut uavhengig av kilde, og gjør det enkelt og brukervennlig å navigere mellom de ulike informasjonskildene.

2.4.4 JavaScript-rammeverk

JavaScript-rammeverk er en viktig del av moderne web-applikasjonsutvikling. Implementasjon av slike rammeverk gir tilgang til pålitelige verktøy for utvikling og design av systemet, som bidrar til en enklere måte å bygge store prosjekter. Et rammeverk som fungerer over en hel applikasjon vil gjøre applikasjonen forutsigbar, som betyr at applikasjonen er enkel å vedlikeholde selv når den øker i kompleksitet og størrelse (MDN Web Docs, 2022). JavaScript-rammeverk er mye brukt og er et populært hjelpemiddel i web-applikasjonsutvikling.

Kapittel 3: Metode

3.1 Hva er en forskningsmetode?

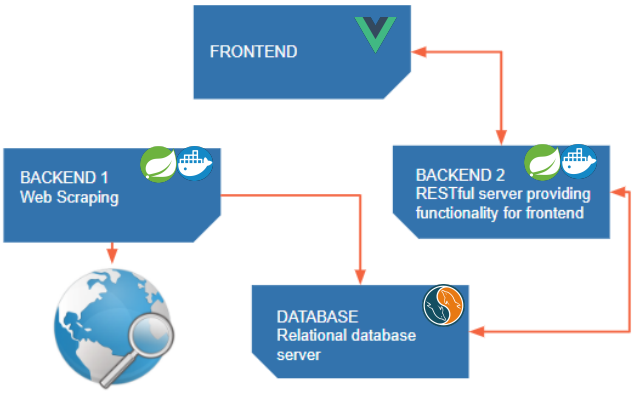
En forskningsmetode er en systematisk og planmessig fremgangsmåte som benyttes innenfor vitenskapelig forskning for å etablere pålitelig kunnskap og holdbare teorier om et forskningstema (Grønmo, 2021). For å besvare problemstillingen og forskningsspørsmålene på best mulig måte, bør man også velge en god forskningsmetode. Tydelige forskningsmetoder er viktig for å anskaffe faglige forankrede og robuste forskningsresultater. Forskningsmetoder kan i stor grad påvirke hvor kunnskapsbaserte vurderinger og valg er, samtidig som det får hovedrapporten til å fremstå systematisk, logisk, vitenskapelig og faglig troverdig (Hentet den 05.04.2022 fra bachelor-seminaret på Blackboard "Å tenke vitenskapelig" 12. jan 2022).

En forskningsmetode skiller seg fra en utviklingsmetode, da en utviklingsmetode er en prosess eller serie med prosesser brukt i blant annet systemutviklingsprosjekt (Alliance Software, n.d.). Det finnes forskjellige metoder, ut fra hvilken utviklingsprosess man ønsker. Utviklingsmetoden omhandler utviklingen av produktet og systemet vårt.

3.2 System

3.2.1 Systemets arkitektur og oppbygning

AIRS er et system som skal bestå av flere komponenter; to backends, én frontend og én database. Begge backendene kjøres i Docker og er isolert fra vertsoperativssystemet. Den ene backenden skal ta seg av funksjonaliteten rundt innhenting av sikkerhetsfeil, og fungerer ved å bruke *web-scraping* (se punkt 2.1.7 Web-scraping).



Figur 3.2.1: Oversikt over komponentene i systemet. Figuren har vi tegnet selv.

Denne delen av systemet skal kommunisere med en database som inneholder informasjon om kundene til Atea, og brukes til å lagre innhentede sikkerhetsfeil og annen informasjon som er relevant for plattformen. Den andre backenden skal dermed bidra med funksjonalitet til frontend og ta seg av kommunikasjonen mellom databasen og frontend. Slik vil alle delene av systemet ha en måte å samspille på samtidig som de er separert fra hverandre av sikkerhetsmessige og praktiske grunner. Definisjonen på en slik arkitektur er en *microservice- arkitektur*. Se vedlegg H (Sonen et al., 2022).

Frontend, altså brukergrensesnittet til systemet skal deretter tilby en måte for brukerne å få en oversikt over nye sikkerhetsfeil, produsenter, kunder, teknologier og andre viktige identifikatorer og tilleggsinformasjon. Her skal man også kunne redigere eksisterende informasjon, samt sortere og filtrere på ulike kriterier.

3.2.2 Valg av teknologier og verktøy

Når vi skulle velge teknologier og verktøy var det viktig for oss å betrakte hva produktet skulle bestå av, hvilke funksjonaliteter det skulle innehave og at det stod i tråd med det som er beskrevet i Vedlegg F (Hadzalic et al., 2022). Som pekepinne på hva produktet skulle inneholde av funksjonaliteter eller begrensninger tar vi utgangspunkt i delene av Vedlegg F som beskriver brukernes behov (punkt 3.4) og ikke-funksjonelle egenskaper og andre krav (punkt 6) (Hadzalic et al., 2022).

Valg av teknologier

Under utviklingen av AIRS-prosjektet har vi valgt å bruke en rekke teknologier for å utvikle et produkt som er både sikkert og pålitelig, samtidig som det dekker behovene som oppdragsgiver har definert.

Applikasjons-containerisering - Docker
En teknologi som er brukt i prosjektet og produktet er Docker. En av hensiktene til at vi valgte å implementere Docker, er at systemet skal kunne distribueres sikkert og systemuavhengig. Docker gjør det slik at man ikke er avhengig av et spesielt operativsystem for å kjøre programmene. På denne måten kan begge backendene være isolert fra resten av operativsystemet under kjøring. Hvis en av tjenerne slutter å fungere vil ikke dette påvirke resten av datamaskinen, kun applikasjonen. Docker underbygger vårt ønske om en pålitelig applikasjon.

Relasjonsdatabase - Microsoft Azure MySQL
En MySQL-database var en av teknologiene som ble forhåndsplukket av oss i bachelorgruppen, siden vi har hatt god erfaring med denne typen database gjennom studieløpet. Den kvalifiserer seg godt mot å dekke behovene for sikker lagring og behandling av data, som beskrevet som et krav i punkt 3.4.2 Datahåndtering i Vedlegg F (Hadzalic et al., 2022). I tillegg fant vi ut at vi ville ha databasen i Microsoft Azure, da alle i gruppen kunne ha enkel tilgang til den. På denne måten kunne den også implementeres på en enkel måte i flere deler av systemet. Relasjonsdatabasen vi valgte har i tillegg et brukervennlig grensesnitt, er godt dokumentert og har godt integrerte sikkerhetsrutiner.

REST API - Spring Boot Web
En annen teknologi som vi også valgte selv å implementere var Spring Boot, som er et open-source rammeverk for utvikling av java-applikasjoner. Dette rammeverket har blitt mye brukt under utvikling av ulike full-stack applikasjoner gjennom studieløpet, og er det mest populære Java-web rammeverket på markedet i dag. (Fol, 2021)
Her har vi gode ressurser fra både tidligere undervisning, Spring Boot sin egen dokumentasjon og online-forumer, som gjør at dette valget av utviklingsverktøy skilte seg ut blant andre alternativer. Spring Boot har også integrert et rammeverk som kalles Spring Security, som gjør det enkelt å implementere sikkerhetsrutiner for webapplikasjonen. Dette rammeverket dekker også opp under kravet i punkt 3.4.2 Datahåndtering i Vedlegg F - sikker lagring og behandling av data, da vi kan bruke dette for å implementere autentisering og kryptering i webapplikasjonen (Hadzalic et al., 2022).

Database-kommunikasjon - Hibernate og Spring Data JPA
Hibernate er et rammeverk som gjør det enkelt å knytte Java-klasser opp mot databasetabeller i en relasjonsdatabase ved hjelp av Java Persistence API. Denne koblingen kan enten opprettes ved å konfigurere en XML-fil eller ved å bruke Java-annotasjoner på de ulike klassene som skal knyttes opp mot databasen. Her kan man også definere ulike forhold mellom databasetabellene ved å annotere dem med one-to-many eller many-to-many annotasjonen. Alt av relasjoner mellom tabeller i databasen blir altså definert i Java-programmet, og man slipper dermed å skrive egne database-queries for å lage innholdet i den.

Spring Data JPA er en add-on til Java Persistence API, som brukes til å kommunisere med en database ved hjelp av Jpa Repositories. Her bruker man en samling av klasser og metoder for å kommunisere med databasen, slik at man ikke trenger å manuelt skrive database-queries for å legge til eller hente ut informasjon.

JavaScript-rammeverk - Vue 3
Vue.js er et JavaScript-rammeverk produsert for utvikling av frontend-delen av systemer. Her var det flere gode alternativer til rammeverk, blant annet React som også var godt kunne blitt brukt. Grunnen til at vi istedenfor valgte å bruke Vue 3 er fordi React krever mer erfaring for å beskytte mot web-sårbarheter. Det er derimot langt mer populært å bruke React, og det er både bedre dokumentert og enklere å skalere opp. Til tross for dette er det enklere å bruke Vue for å komme raskt i gang med utviklingen av webapplikasjonen, vi har brukt det mer under studieløpet og det passer utmerket godt til å utvikle sikre applikasjoner. Det beskytter blant annet automatisk mot script-injections - som er en sårbarhet hvor man kan injisere ondsinnet kode i brukergrensesnittet, og atributt-binding hvor man kobler ondsinnet kode opp mot en funksjon. Det er også godt dokumentert hvilke grep man burde ta hvis man skal utvikle en sikker applikasjon med Vue 3. (Vue.js, 2022). Bruk av dette rammeverket oppfyller kravet i Vedlegg F under punkt 3.4.1 Innhenting - portal med oversikt over kunder og teknologier (Hadzalic et al., 2022).

Valg av verktøy

I kombinasjon med de ulike teknologiene vi har brukt gjennom utviklingen av prosjektet har vi også valgt å bruke en del verktøy for å gjøre utviklingsprosessen enklere. Disse valgene er basert på tidligere erfaringer eller valgt ut i fra industristandarder eller anbefalinger fra eksterne aktører.

Discord
Discord er et program designet for kommunikasjon over nettet. Dette gjelder kommunikasjon som vanlig tekst og videosamtaler. Programmet gjør det mulig å lage flere kanaler for enkelte grupper hvor man kan snakke sammen. I vårt prosjekt har vi brukt Discord som vårt primære møtested. Vi foretrekker Discord over for eksempel Microsoft Teams, da Discord er et enklere verktøy å bruke. Discord tilbyr også bedre oversikt over ressurser, kanaler og dialoger.

Figma
Figma er et skybasert designprogram med mye funksjonalitet. Programmet kjører i nettleseren og er enkelt og effektivt å bruke. Man kan samarbeide med flere og se

endringer kontinuerlig (Kopf, n.d.). En grunn til at vi valgte å ta i bruk dette verktøyet er at vi har brukt det tidligere og er kjent med hvor enkelt det er å bruke. Figma er et program vi brukte for å lage Wireframe for brukergrensesnittet. Verktøyet tillot oss å jobbe sammen om denne Wireframen.

Git
Git er et kontrollsystem som brukes for å spore filendringer. Systemet er også brukt for å holde orden på endringer i prosjekter, noe som gjør Git til et bra verktøy i gruppeprosjekter. I slike gruppeprosjekter har hvert medlem mulighet til å gjøre endringer på alle filer. Arbeid utført av hvert medlem vil ikke påvirke arbeidet til andre medlemmer. Git passer på at alt arbeid blir sporet og håndtert (Kelley, 2022). Git har i stor grad hjulpet oss med av utviklingen av systemet vårt. Ved hjelp av Git kunne vi lage egne grener ut fra en hovedgren. Hvert medlem av gruppen jobbet på sin egen gren av prosjektet, slik at man ikke påvirket arbeid og progresjon for de andre medlemmene. Underveis brukte vi en utviklingsgren, hvor vi fikk sammenflettet arbeid. Til slutt har vi hovedgrenen hvor det ferdige produktet skal være.

GitKraken
GitKraken er en GUI-applikasjon for Git som tilbyr de fleste kommandolinje-operasjoner og er effektiv og pålitelig å bruke. Den gjør Git mer visuell, da man ikke har et like stort behov for å bruke Git-terminalen. I GitKraken har man blant annet oversikt over *commit*-historie, *merge*-konflikter og *pull/push*-forespørsel (GitKraken, n.d.). Vi liker å bruke dette verktøyet, da det er enkelt og sikkert å bruke. Vi kan enkelt slå sammen Git-brancher ved å få en visuell oversikt over hvor konflikter oppstår. GitKraken er et verktøy vi har brukt tidligere. Derfor var det enkelt for oss å implementere det i bachelorprosjektet.

Google Drive
Google Drive er en skybasert fillagrings- og synkroniseringstjeneste som gjør det mulig å lagre forskjellige dokumenter på nett og i skyen (Muchmore & Duffy, 2022). Gjennom prosjektet har vi produsert mange dokumenter. Vi har valgt å bruke Google Drive-tjenesten for å lagre disse filene. En grunn til dette er at tjenesten tilbyr lagring av filer på nett. Vi som gruppe baserer vår dokumentering på nett. Google Drive er derfor en brukbar plattform for oss, hvor vi kan lagre all dokumentasjon vi produserer underveis. Filene blir i tillegg synkronisert og oppdatert radig på alle våre maskiner. Dette gjør det enklere for oss, da vi hele tiden har oversikt over hvor i dokumentet alle er. Slik skriver vi ikke unødvendig tekst, da vi kan se live progresjon.

IntelliJ IDEA
IntelliJ er et integrert utviklingsmiljø (IDE) skrevet i Java for utvikling av programvare, og er den plattformen vi har brukt for å utvikle backend-delen av produktet. Det har et enkelt grensesnitt, god støtte for implementering av eksterne biblioteker, verktøy for feilsøking og kode-assistanse. Her har man alt man trenger for å utvikle backend-delen av en webapplikasjon.

Maven
Maven er et styrings- og tolkningsverktøy for Java-baserte-prosjekter. Verktøyet baserer seg på POM (Project Object Model) som gjør at Maven kan styre byggeprosess, dokumentering og rapportering i et prosjekt. Hensikter med Maven er å skape en effektiv og enkel byggeprosess, samt bidra med til bedre kvalitet rundt prosjektinformasjon og

utvikling (Apache Maven, n.d.). I vårt prosjekt har vi tatt i bruk Maven for å få en effektiv byggeprosess da applikasjonen blir relativt stor. Da POM den fundamentale byggesteinen for Maven, har vi tatt i bruk pom.xml-filen. Denne filen inneholder informasjon angående prosjektet, ulike konfigurasjoner og avhengigheter Maven bruker for å bygge prosjektet. Uten Maven antar vi at prosjektbyggingen ville vært mer tidkrevende og komplisert.

Microsoft Azure
Microsoft Azure er en skyplattform som består av flere produkter og skytjenester. Gjennom Azure kan man bygge, kjøre og styre prosjekter i skyen. Man har mulighet til å jobbe mellom flere skyer samtidig, og knytte opp prosjekter lokalt (Microsoft, n.d.). Plattformen kan brukes for å erstatte servere og databaser, som egner seg bedre i skyen. Azure er tilgjengelig for nesten alle operativsystem og er pålitelig å bruke (McCoy, n.d.). Vi valgte å ta i bruk Azure da vi så et behov for å plassere vår relasjonsdatabase i skyplattformen. Dette bidro til å gi oss enklere tilgang til databasen i ulike deler av prosjektet, slik at ikke bare et enkelt medlem hadde tilgang til den. Vi har også gjort det slik at begge backendene kan kjøre i Azure, slik at kjøringen deres ikke påvirker resten av datamaskinen applikasjonen skal kjøre på. På denne måten kan også flere klienter bruke samme tjener.

Microsoft Teams
Microsoft Teams er en plattform designet for kommunikasjon og samarbeid. Plattformen tilbyr funksjonalitet som vanlig meldingutveksling, ringing og videosamtaler. I tillegg til kommunikasjonsfunksjonalitet har Teams mer å by på. Man kan også dele dokumenter og filer. Dette gjør Teams til en plattform hvor man enkelt kan jobbe med gruppeprosjekter (Microsoft, n.d.). Gjennom prosjektet har vi brukt Teams til digitale møter med veileder og oppdragsgiver. Vi har også tatt i bruk issue-boardet som er tilgjengelig. Dette har hjulpet oss med å få oversikt over mangler og progresjon gjennom prosjektet.

MySQL Workbench
MySQL Workbench er et verktøy som gir en visuell oversikt over databaser. Verktøyet er enkelt å bruke, og gjør det mulig å modellere data og konstruere, utføre og optimalisere SQL-spørringer (MySQL, n.d.). En grunn til at vi bruker MySQL Workbench er fordi det nettopp er enkelt å bruke. Vi kan lett legge til nye entiteter i ulike tabeller. En annen grunn er at vi er vant til å bruke verktøyet. Vi har tidligere hatt emner som *IDATT2103 Databaser* og *IDATT2106 Systemutvikling 2 med smidig prosjek*t, hvor vi fikk bruk for programmet. I prosjektet er det ikke utført mange dedikerte SQL-spørringer, da vi har brukt andre metoder for datahåndtering i forhold til databasen. MySQL Workbench har likevel vært et brukbart verktøy for vår del, da det ga oss en oversikt over våre tabeller og entiteter.

Node Package Manager
Node Package Manager (NPM) er et kommandolinjeverktøy som blant annet brukes for å installere, avinstallere og oppdatere tilgjengelig pakker gjennom Node.js. NPM er også et åpent bibliotek med Node.js-pakker og moduler som alle utviklere kan bruke. NPM er inkludert dersom man laster ned Node.js. I prosjektet har vi brukt NPM til å installere nødvendige moduler i klient-delen av systemet. NPM er et fint verktøy for enkel installering av brukbare pakker. Hvis man skriver "npm install" etter oppretting av prosjektet vil man laste ned alle nødvendige moduler, basert på prosjektets informasjon.

Deretter kan man skrive "npm run serve" for å åpne web-applikasjonen. NPM er derfor et viktig verktøy i dette prosjektet.

NTNUs GitLab
GitLab er et nettbasert Git-repository som tilbyr funksjonalitet og håndtering av private og åpne prosjekter og filstrukturer. Man har også tilgang til et issue-board og flere wikier. GitLab bidrar til enkelt samarbeid innad i grupper og oversikt over progresjon og prosess (Kelley, 2022). NTNUs GitLab gjør det mulig å lage private prosjekt som er synlige for forelesere og andre studenter. Vårt prosjekt er først og fremst kun synlig for oss og veilederen vår. GitLab hjelper oss med å få en god oversikt over alle mapper, filer, grener og endringer. På denne måten kan vi kontinuerlig se hvilke endringer som er gjort hvor, av hvem og i hvilken gren.

Visual Paradigm Online
Visual Paradigm Online (VP Online) er et gratis verktøy for design og skissering av programvare og system. På den nettbaserte plattformen kan man lage blant annet klassediagram, domenemodell og sekvensdiagram (Visual Paradigm, n.d.). I prosjektet har vi brukt dette verktøyet for å skissere domenemodellen vår. Vi var kjent med Visual Paradigm fra før av og visste at plattformen var bra for tegning av systemer.

Visual Studio Code
Visual Studio Code er et integrert utviklingsmiljø (IDE) hvor man selv kan velge blant en mengde programmeringsspråk for å utvikle programvare. Denne er mer fleksibel enn IntelliJ, og brukes i vårt produkt for å utvikle frontend-delen av produktet. Denne plattformen er svært godt dokumentert og gjør det enkelt å jobbe med rammeverk som Vue 3, som vi har brukt for å utvikle frontend.

3.2.3 Utviklingsmetode

Gjennom utviklingsprosessen har vi tatt i bruk en form for smidig utviklingsmetodikk. I tidligere prosjekter på studiet har vi blitt kjent med konseptet scrum og hvilke effekter det kan ha på gruppesamarbeid. Derfor fant vi tidlig ut at vi ville ta i bruk en tilpasset form for scrum, hvor vi ikke følger scrum-modellen til punkt og prikke, men utformer vår egen løsning for scrum. Hovedgrunnene til at vi valgte å ta utgangspunkt i scrum er fordi vi var vant med denne typen utviklingsmetodikk og følte at scrum kunne bidra positivt i dette prosjektet. Vår tilpasning baserte seg på at vi hadde et annet emne som gikk parallelt med dette prosjektet, fra januar til mars. På grunn av dette måtte vi disponere tiden på en smart måte og fant ut at en ren scrum-metodikk ikke passet, da vi ikke hadde tilstrekkelig med arbeidstimer til å følge den strenge utviklingsmetodikken i scrum.

Tilpasningen vår tar utgangspunkt i noen av scrum-metodikkens hovedattributter. I tilpasningen har vi sprinter på lik linje med vanlig scrum, hvor hver sprint varte gjennomsnittlig i tre uker. Vi startet likevel ikke med sprinter før 7. februar, da vi begynte med utviklingen av produktet. Se Gantt-diagrammet i vedlegg E for en mer detaljert oversikt (Kvarving et al., n.d.). Vi har ikke hatt en scrum-master under prosjektet, men har hatt gode dialoger med veileder og oppdragsgiver for å vise hva vi har gjort og planer for videre arbeid. På denne måten følger vi en smidig utviklingsmetodikk, samtidig som vi har tilpasset til vår egen prosess og arbeidstid.

Møteplassen for gruppen i starten av utviklingsprosessen har stort sett vært på nett over Discord, siden samfunnet fortsatt var påvirket av Covid-19. På grunn av dette hadde vi heller ikke et fysisk møterom, verken på skolen eller hos Atea. Vi forhørte oss om kontor eller møterom hos Atea, men omtrent alle av Ateas ansatte var sendt hjem på hjemmekontor, noe gjorde det vanskelig å ta imot studenter.

Som studenter under Covid-19 hadde vi allerede tilpasset oss en skoledag på nett, og det var derfor ikke et stort problem å samarbeide på denne måten. Som nevnt brukte vi Discord som hovedplattform, men tok i bruk Microsoft Teams for møter med veileder og oppgavestiller. Etter hvert som Covid-restriksjonene ble fjernet fra samfunnet, åpnet det seg en mulighet for å få tilgang til Ateas lokaler. Den 21. april fikk vi adgangskort, som gjorde at vi kunne komme og gå når vi ville. Jobbing over nett gikk smertefritt, men vi benyttet oss av kontorplassen de dagene vi hadde mulighet etter at vi fikk adgangskort.

Gjennom prosjektet har vi hatt som mål å opprettholde god kommunikasjon med både veileder og oppdragsgiver, og hadde veiledningsmøter annenhver til hver tredje uke for å få en rask gjennomgang av utført arbeid. Vi ville også holde en god dialog med oppdragsgiver og Atea IRT angående produkt og prosess, slik at vi holdt oss oppdatert på oppdragsgivers krav og behov underveis i utviklingen.

3.2.4 Arbeids- og rollefordeling

Som vist i figur 3.2.1, er systemet hovedsakelig delt opp i fire deler. I starten av prosjektet hadde vi en idé om hvordan vi skulle fordele utviklingen av produktet mellom oss, fordi vi ville forsikre oss om at arbeidsmengden var lik for hvert gruppemedlem. Arbeidsfordelingen i starten av utviklingen var derfor slik:

- Backend for innhenting av sikkerhetsfeil skal settes opp av Endré.
- Backend som er knyttet til frontend skal utvikles av Odin.
- Databasen skal utvikles av Endré og Odin, da begge må samarbeide om oppsett for å sørge for at database-dataen kan skrives og leses.
- Frontend-delen av applikasjonen skal utvikles av Scott.

Underveis i prosessen har fordelingen vært noenlunde slik som beskrevet over, ved at hvert gruppemedlem har hatt én hovedoppgave hver. Etter at de viktigste delene av systemet var satt opp, bidro alle litt over alt eller der hvor det fantes mangler og behov for hjelp. I backenden for kommunikasjon med frontend har Endré vært med å bidratt og Scott har hjulpet til. Endré og Odin har også bidratt i frontend-delen. På denne måten har vi sørget for at alle har vært med å arbeidet på alle delene av applikasjonen, samtidig som vi har tilspisset hvem som jobber med hvilken del, slik at det er enklere å fokusere på én hovedoppgave for hver person. Det vi også har tenkt på er at oppgavene skal være så uavhengige som mulig. Dette betyr for eksempel at Scott har brukt testobjekter i frontend for å få en visuell anelse av hvordan ting vil se ut når hele systemet knyttes sammen, og enkelt få en . Slik har arbeidet vært effektivt samtidig som vi har hatt omtrent like store arbeidsmengder.

Når det gjelder fordeling av dokumentering har vi gjort det på en litt annerledes måte. I motsetning til produktutviklingen, hvor vi stort sett har jobbet selvstendig på totalt uavhengige oppgaver, har vi sørget for å samarbeide om alle dokumenter. Slik kan vi forsikre oss om at alle dokumenter blir skrevet på en faglig og bra måte, samtidig som alle får lest gjennom dokumentene og får ryddet opp hvor det finnes feil eller mangler. Innad i hvert dokument har hvert gruppemedlem skrevet egne avsnitt og deler, men vi

har i fellesskap gått gjennom alle punkt. Slik oppnår vi en effektiv arbeidsprosess hvor alle bidrar.

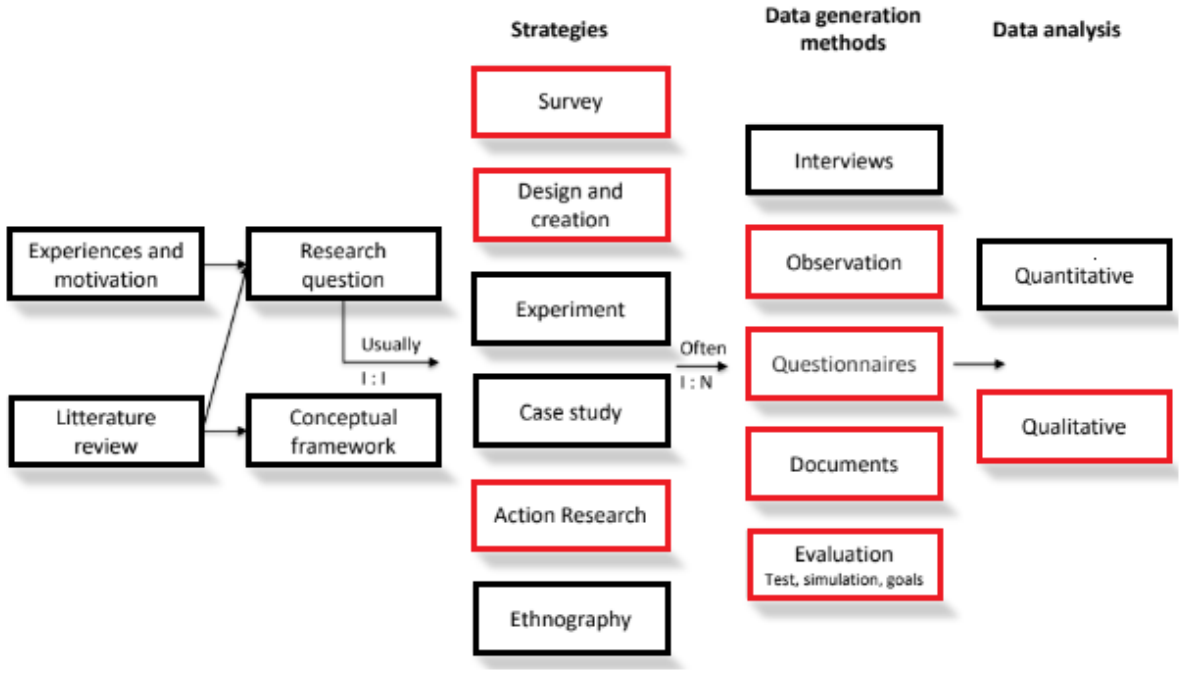
3.3 Forskningsmetode

I dette delkapittelet ser vi på de ulike forskningsmetodene som ble brukt underveis i prosjektet for å kunne svare på problemstillingen som står beskrevet i oppgaveteksten. Måten vi gikk fram for å løse den generelle problemstillingen var først og fremst å dele den opp i mindre spørsmål. Disse spørsmålene er mer konkrete og dekker en liten del av den komplette problemstillingen som vi ønsker å svare på. Ved å svare på forskningsspørsmålene så vil vi, basert på de funnene vi fikk der, kunne ta en vurdering på problemstillingen. Dette forenkler også forskningsdelen forbundet med AIRS-prosjektet, da vi kan spesifisere flere enklere fremgangsmåter for å finne svar på de ulike spørsmålene - da de er uavhengige og krever ulike fremgangsmåter som både kan være kvantitative og kvalitative. Styrker ved denne splittelsen av hovedproblemstillingen er at det blant annet bidrar til at vi unngår usystematisk forskning, feiltolkninger og misforståelser, samt å gå glipp av noe eller svare på noe som er irrelevant til hovedproblemstillingen.

Vi valgte å dele problemstillingen inn i fire forskningsspørsmål, og bestemte oss for en kvalitativ fremgangsmåte for å svare svare på spørsmålene. De tre første forskningsspørsmålene er mer teoretisk i sin natur, mens det siste forskningsspørsmålet er mer empirisk. Fordelingen på spørsmålene som blir tatt opp i delkapittel 1.5.1:

1. *Hvordan er det gunstig å lagre brukerdata?*
2. *Hvilke metoder er best egnet for sikker autentisering av brukere på forskjellige deler av systemet?*
3. *Hvordan sikre pålitelig drift av systemet?*
4. *Hvilke effekter gir bruken av det nye systemet sammenlignet med tidligere manuelt system?*

Forskningsmetoden som ble fastsatt er basert på seminaret i IDATT2900 Bacheloroppgave som ble gjennomført onsdag 12. jan 2022. Under seminaret ble det presentert en modell som ble beskrevet som et hjelpemiddel for å lage spesifiserte forskningsmetoder. Under vises en figur hvor forskningsmetodene for dette prosjektet visualiseres.



Figur 3.3.1: Kvalitativ forskningsmetode for prosjektet, der de røde boksene representerer metode for forskningsspørsmål 1, 2, 3 og 4. Figuren baserer seg på en hjelpemodell for valg av forskningsprosess. (Hentet fra foiler til Seminar VT1 "å tenke vitenskapelig" på Blackboard)

Kvalitativ metode: Spørsmål 1, 2, 3 & 4	
Strategi	Aksjonsforskning, design og utforming, undersøkelse
Datagenerering	Evaluering, dokumenter, observasjon, spørreundersøkelse
Dataanalyse	Kvalitativ

Tabell 3.3.2: Tabell som viser fremgangsmåten for å generere gode kvalitative data på spørsmål 1, 2, 3 og 4.

3.3.1 Justeringer på forskningsspørsmål underveis i prosjektet

Underveis i prosjektet så ble det gjort justeringer på både problemstilling og sammenhengende spørsmål etter tilbakemeldinger fra prosjektveileder samt diskusjoner som ble tatt i plenum i gruppa. Justeringene ble i hovedsak gjort enten for å konkretisere selve spørsmålet/problemstillingen eller for å få spørsmålet til å bedre tilhørigheten sin til problemstillingen. Vi hadde i utgangspunktet to forskjellige forskningsmetoder, der den ene var kvalitativ og den andre var kvantitativ. Forskningsspørsmål 4 skulle i utgangspunktet basere seg på en kvantitativ forskningsmetode. Vi var nødt til å gjøre om metoden til en kvalitativ metode, da vi ikke fikk tilstrekkelig med svar på spørreundersøkelsen vår.

Tabellen under beskriver alle endringer som ble gjort, sammen med dato og begrunnelse til endringen:

Dato	Endring	Beskrivelse/Grunn
02.Feb 2022	Gikk fra kun en hovedproblemstilling til problemstilling med tilhørende konkrete spørsmål. Fikk også koblet problemstillingen bedre opp mot oppgaven.	Veileder Olav kom med gode grunner på hvordan det blir vanskelig å konkret svare på det generelle spørsmålet vi hadde satt. Fant ei bedre problemstilling samt abstraherte den ned i tilhørende konkrete spørsmål. Spørsmål som veileder poengterte at var viktig svarer ordentlig på problemstillingen.
16.Feb 2022	Gjorde justeringer som konkretiserer spørsmålene som hører til problemstillingen. Gikk altså bort ifra begrep som "best egnet" og over til mer konkrete spørsmål som krever klare svar.	Ble gjennomført etter møte med veileder Olav. Under møtet så kommenterte han at han syns spørsmålene var litt for utydelige på hva de spør etter.
23.Mar 2022	Gjorde justeringer på spørsmål fire.	Ble gjort grunnet kommentar fra veileder under møte mellom han og gruppen. Spørsmål fire som omhandler sammenlikning mellom det gamle og det nye systemet var fortsatt for generelt.
07.Apr 2022	Redefinerte forskningsspørsmål tre for å få det til å svare bedre til problemstillingen.	Redefinerte det tredje forskningsspørsmålet da det forrige spørsmålet ikke ga noe svar gruppa mente var relevant til å løse problemstillingen.
04.Mai 2022	Gjorde om forskningsmetode på spørsmål 1 og 2. Plasserte forskningsspørsmål 4 i kvantitativ kategori.	Etter diskusjon fant vi ut at vi burde endre på noen av forskningsmetodene, da de ikke samsvarte med vår faktiske fremgangsmåte.
07.Mai 2022	La til ytelse som en del av forskningsspørsmål 3.	Dette ble gjort for å konkretisere spørsmålet mer da vi merket under research på spørsmålet at pålitelighet og ytelse går hånd i hånd når man snakker om programvare.
12.Mai 2022	Ga forskningsspørsmål 4 en kvalitativ metode, og fjernet den kvantitative forskningsmetoden helt.	Vi fikk måtte konkludere med at spørsmål 4 måtte svares på ved en kvalitativ metode siden vi bare fikk en tilbakemelding på spørreundersøkelsen. Dermed var det ingen spørsmål som benyttet seg av den tidligere kvantitative metoden, som førte til at vi foreldet den og dermed fjernet den fra rapporten.

Tabell 3.3.2: Tabell som viser endringer på forskningsspørsmålene underveis i prosjektet.

Kapittel 4: Resultater

I dette kapittelet skal vi ta for oss ulike resultater knyttet til prosjektet. Dette omhandler vitenskapelige, ingeniørfaglige og administrative resultater rundt system og forskningsspørsmål.

4.1 Vitenskapelige resultater

4.1.1 Hvordan er det gunstig å lagre brukerdata?

Sensitiv data regnes i sammenheng med denne oppgaven som data som avslører informasjon om en kunde - eller passord for filer, brukere og databaser.
Når man skal lagre sensitiv data inne på et hvilket som helst system, er det viktig å lagre denne dataen på en slik måte at potensielle angripere ikke kan få tak i informasjonen direkte. Hvis man har mulighet for å lese ut sensitiv informasjon direkte, kalles det gjerne at det er lagret i klartekst. Dette kan for eksempel være passord til en bruker som blir lagret direkte i en database, uten noen form for kryptering. Denne typen lagring er den minst sikre metoden man kan bruke for å oppbevare sensitiv informasjon, og er lett leselig for både mennesker og maskiner. Ifølge en studie utført av eSecurity Planet hvor de undersøkte over 750 IT-bedrifter rundt omkring i verden, fant de ut at over 40% lagret priviligerte og administrative passord i klartekst på et Microsoft Word-dokument eller et spreadsheet, mens ytterligere 28% lagret dataen i klartekst på en delt server eller USB-stikke (Goldman, 2016). Det kan være mange grunner til at man ikke lagrer sensitiv informasjon på en sikker måte, men den største grunnen er sannsynligvis at det er mye enklere og mer praktisk å ha det lagret slik. Vil man derimot være beskyttet mot potensielle angripere, kan man ta i bruk mange forskjellige måter å gjøre dette på. For å finne svar på dette spørsmålet bruker vi den kvalitative forskningsmetoden, som definert i kapittel 3.3.

En av de enkleste og mest brukte metodene for sikker lagring av data er ved hjelp av kryptering, og går ut på å bruke en nøkkel for å kryptere og dekryptere informasjon. Dette er en toveis metode hvor data sendes inn som klartekst og kommer ut som en chiffertekst, og kan reverseres fra chiffertekst tilbake til klartekst. På grunn av denne toveis-egenskapen egner den seg ikke til å lagre ekstra sensitiv informasjon som passord, da disse burde oppbevares enda sikrere (Mullvad VPN, n.d.).

For lagring av passord bruker man som oftest hashing, salting og peppering (se punkt 2.1.2). Her finnes det mange forskjellige hashing-funksjoner å velge imellom, men noen av dem er sikrere enn andre. Det er også vanlig at disse algoritmene implementerer en kombinasjon av både hashing, salting og peppering som en del av prosessen. Forskjellen mellom hashing og kryptering er at hashing er en enveis-funksjon, som vil si at man ikke kan hente ut det opprinnelige klartekst-passordet fra hashen.

Blant de beste hashing-algoritmene finner vi Bcrypt og Argon 2, som på hver sin måte bidrar til ulike typer sikkerhet når det kommer til lagring av sensitiv data.

Hashing-algoritmer for lagring av passord

Bcrypt

Et brute-force-angrep er et av de vanligste angrepene på for å finne passord. Målet med et brute-force-angrep er å gjette innloggingsinformasjonen til en bruker (Kaspersky, n.d.). La oss si at en angriper vet mailen eller brukernavnet til en bruker. Gjennom brute-force kan denne angriperen gjette seg frem til hvilket passord den samme brukeren har, ved å prøve nye og nye passord etter hvert som man feiler (Kaspersky, n.d.). Denne prosessen blir ofte automatisert, og utføres gjerne ved hjelp av ekstremt kraftige datamaskiner.

Bcrypt er en hashing-algoritme som gjør det ekstremt kostbart å brute-force passord, og har i praksis gjort det nesten umulig. Denne algoritmen kalles ofte for en CPU-tung algoritme, fordi den krever flere CPU-sykluser for å kalkulere én enkelt hash. Grunnen til dette er at man definerer en arbeidsfaktor, som avgjør hvor mange iterasjoner av algoritmen passordet skal hashes med. Her må man finne balansen mellom sikkerhet og ytelse, siden en for høy work factor gjør at verifisering av passord tar for lang tid, mens en for lav work factor reduserer sikkerheten.

Når man hasher et passord med bcrypt sender man inn et passord som er maksimalt 72 bytes langt, en numerisk kostnad og en 16-bytes (128-bit) salt-verdi. Saltet er typisk tilfeldig generert. Bcrypt-funksjonen bruker disse parameterne til å beregne en 24-byte (192-bit) hash. For å autentisere et passord som er hashet med bcrypt henter man ut saltet til det registrerte passordet. Hvert hashet passord registrert i databasen lagres med saltet som en del av tekststrengen. Dette saltet blir deretter brukt under hashing av passordet som skal autentiseres, og sammenlignes deretter med hashen til det registrerte passordet (*Bcrypt*, 2022).

Bcrypt gir blant annet god beskyttelse mot rainbow-tables-angrep, som utnytter en stor database av allerede utregnede hasher for å finne et passord. Den skalerer også meget godt med økende beregningskraft siden man kan benytte seg av flere hashing-iterasjoner, noe som gjør utregningen for en eventuell angriper eksponentielt vanskeligere. Bcrypt har vært brukt og kjent lenge, og har som følger kjente metodikker for å angripe systemer som bruker algoritmen. Til tross for dette er det en av de sikreste hashing-algoritmene vi vet om, og foretrekkes av mange til sikring av passord. (Preziuso, n.d.) (OWASP, n.d.)

Argon 2

Argon 2 er en hashing-algoritme (avtrykksalgoritme) som er blitt anerkjent for å være ekstremt sikker for både lagring av sensitiv data og passord, og ble kåret til beste hashing-algoritme i Password Hashing Competition i 2015. (Wikipedia, n.d.) Denne algoritmen har tre utgaver, hvor hver av dem gir beskyttelse mot ulike typer angrep; Argon2i, Argon2d og Argon2id.

Argon2i bruker noe som kalles data-uavhengig minnetilgang, som er en funksjon for beregning av hasher som krever mye minne. Denne utgaven egner seg best til hashing av passord, og gir blant annet god beskyttelse mot “side-channel attacks”, som utnytter

informasjon om bruken av selve datamaskinen slik som strømbruk og tidsbruk på forskjellige utregninger for å få innsikt i hemmeligheter om systemet.

Argon2d er raskere og bruker noe som kalles data-avhengig minnetilgang, som er en funksjon for beregning av hasher som er tung for GPUen. Denne utgaven gir god beskyttelse mot GPU og ASIC-angrep. GPU-angrep utføres ved å utnytte de mange tusen kjernene til moderne GPUer for å brute force passord, mens ASIC bruker spesielt tilpassede kretser for å gjøre det samme.

Argon2id er en kombinasjon av disse to utgavene, som bruker en hybrid-versjon av data-avhengig og data-uavhengig minnetilgang for å gi god beskyttelse mot både side-channel attacks, GPU/ASIC-angrep og andre typer angrep som ofte blir brukt i dagens angrep. (Biryukov et al., 2016)

Argon2id er altså den hashing-algoritmen som gir best beskyttelse mot flest typer angrep, og foretrekkes av mange industri-profesjonelle overfor andre hashing-algoritmer som prøver å oppnå det samme som Argon 2.

I et blogginnlegg skrevet av eksperter fra UiO som kritiserer Datatilsynet blir Argon 2 også nevnt som en hashing-algoritme med stort potensiale, men det blir notert at algoritmen er for ny til at den kan anbefales som ny standard. Innlegget ble skrevet i 2016, og algoritmen har gjennomgått stor utvikling siden den tid. Den er både sikrere og mer fleksibel enn tidligere anbefalte algoritmer som SHA-2, og kan derfor vurderes fremfor disse om man har ressurser til å implementere dem. Det samme gjelder for Bcrypt - noe som også blir notert i blogginnlegget fra UiO. (Titan UiO, 2016). Begge disse algoritmene støtter også opp under kravene fra Datatilsynet som ble nevnt i punkt 2.1.2.

Krypteringsalgoritmer for behandling av annen sensitiv data

For sikker behandling av sensitiv data er det mest vanlig å bruke krypteringsalgoritmer, enten for å skjule data mens den beveger seg mellom to parter eller for at den skal være skjult mens den ligger klar for uthenting.

Ved å kryptere data før den havner inne på en database blir den beskyttet både på vei inn i databasen og i ettertid hvis en angriper skulle få tilgang til databasen. Her bruker man ofte en spesifikk krypteringsalgoritme for å kryptere og dekryptere data, mens data i bevegelse mellom en tjener og en klient som oftest blir håndtert av protokoller som HTTPS.

For at data skal kunne lagres så sikkert som mulig er det viktig at man bruker en tilstrekkelig sikker krypteringsalgoritme, og to av de sikreste og mest brukte algoritmene er AES og RSA. Dette er de to krypteringsalgoritmene som blir anbefalt av Datatilsynet, i henhold til krav fra Nasjonal Sikkerhetsmyndighet. (Datatilsynet, n.d.)

AES - lagring av sensitiv sensitiv data

AES er en symmetrisk krypteringsalgoritme som er godt kjent for å kunne motstå alle slags typer angrep, og er en av de sikreste krypteringsalgoritmene på markedet i sin 256-bits form. Den ble satt som standard av National Institute of Standards (NIST) i

2001, og har siden blitt implementert i både kryptering av topphemmelig informasjon i det amerikanske militæret og bredt implementert av både statlige og offentlige aktører. (NordPass, 2020). Denne krypteringsalgoritmen passer best til kryptering av data i et isolert system, som en database eller harddisk hvor den kommer til å bli liggende statisk. (Townsend, 2019)

Datatilsynet anbefaler AES med 128 eller 156-bits nøkkel, basert på hvor sikkert det er nødvendig å lagre dataen. (Datatilsynet, n.d.). Denne algoritmen er også svært rask og relativt enkel å implementere, noe som gjør at den utmerker seg til kryptering der hastighet og pålitelighet er viktig.

RSA - kommunikasjon med sensitiv data

RSA er en asymmetrisk krypteringsalgoritme. Denne er også veldig sikker, og har litt andre bruksområder enn AES siden man her har både en privat og en offentlig nøkkel for kryptering og dekryptering. På grunn av dette er den mer vanlig å bruke i systemer hvor man har to forskjellige endepunkter for dataoverføringen. I Datatilsynets anbefalinger for bruk av RSA-algoritmen anbefaler de å bruke en 3072-bits nøkkel, noe som gjør krypteringen så å si uknekkelig. (Datatilsynet, n.d.). Denne algoritmen er mye tregere enn AES, og krever langt flere ressurser for å operere. Den blir derfor ofte brukt kun til å kryptere mindre mengder med data.

På grunn av de sterke sikkerhets-egenskapene til både RSA og AES er det ikke uvanlig å kombinere de to krypteringsalgoritmene for dobbel sikkerhet - hvor man først krypterer mesteparten av dataen ved hjelp av AES for deretter å kryptere nøkkelen ved hjelp av RSA-algoritmen. (Townsend, 2019)

Sikker oppbevaring av krypteringsnøkler

Selv om kryptering av sensitiv data kan gjøres ekstremt sikkert med moderne krypteringsalgoritmer, har det lite å si hvis man ikke lagrer krypteringsnøklene på et sikkert sted. Hvis en angriper får tilgang til disse, vil man enkelt kunne dekryptere alt av sensitiv informasjon uten å utføre noen form for angrep, og det er derfor særdeles viktig å ikke la dem ligge lett tilgjengelig. Det finnes heldigvis mange metoder for å lagre disse sikkert, selv om det ikke alltid blir gjort. En av de sikreste alternativene er å lagre krypteringsnøkler i en ekstern maskinvare som er laget for å håndtere slik lagring, eller ved å kryptere nøklene ved hjelp av et administratorpassord lokalt, på en separat server eller på en sikker database. Å bruke programvare for å sikre krypteringsnøkler er også ekstremt sikkert med moderne programvare, og enklere å implementere enn hardware-lagring. (Panda, 2021). Datatilsynet anbefaler å bruke programvare for å generere eller lagre krypteringsnøkler for datakommunikasjon, og hardware for å lagre langtidsnøkler brukt i statiske systemer. (Datatilsynet, n.d.)

4.1.2 Hvilke metoder er best egnet for sikker autentisering av brukere på forskjellige deler av systemet?

Det finnes flere gode metoder som sørger for sikker autentisering av brukere. Autentisering av brukere er viktig for å sikre begrenset tilgang til sensitiv data, slik at en bruker ikke kan få tilgang til informasjon om en annen bruker. Forskjellige systemer bruker forskjellige autentiseringsmetoder ut ifra hvor sikker autentiseringen må være.

Likevel finnes det noen metoder som er sikrere enn andre. Disse skal vi ta for oss i dette delkapitlet. For å svare på dette forskningsspørsmålet har vi tatt i bruk en kvalitativ forskningsmetode. Svarene vi kommer frem til baserer seg på dokumentanalyse, forskning, og en form for evaluering.

Autentisering ved bruk av passord

Den mest brukte formen for autentisering er passord, som kan bygges opp på forskjellige måter. Selv om man har sterke passord, vil man alltid være sårbar for ulike typer angrep som for eksempel brute-force angrep.

Et annet eksempel er phishing-angrep. Et phishing-angrep er et angrep som baserer seg på å stjele eller få tilgang til brukerdata, blant annet innloggingsinformasjon (Imperva, n.d.). Slike angrep kan forekomme gjennom ondsinnede lenker man kan få tilsendt via SMS eller på mail, og kan ofte være helt identiske til mailer eller SMSer som blir sendt ut av kjente aktører til vanlig. Angriperne som har konstruert mailen har ofte lagt stor innsats i å speile den legitime nettsiden som phishing-angrepet skal etterligne, og brukes til å få intetanende mottakere til å fylle inn brukernavnet og passordet som er registrert på den legitime nettsiden. De som har laget lenken vil dermed kunne lese innloggingsinformasjon og forsøke å bruke denne informasjonen både på den legitime nettsiden og flere steder hvor brukernavnet og passordet er registrert. Ifølge Ranjeet Vidwans fra ClearedIn, har antall phishing angrep økt med 76% fra 2017 (Vidwans, n.d.). En undersøkelse gjort i 2012 av CSID og Experian Partner Solutions viser at 61% av amerikanske kunder bruker samme passord på flere nettsider, og 54% har fem forskjellige passord eller mindre. 21% av kundene har også fått en av deres nettkontoer kompromittert (CSID, n.d.). I dagens samfunn kan man ha ganske mange brukere på nett. En analyse av Dashlane viser at en gjennomsnittlig bruker har rundt 90 nettkontoer i løpet av livet sitt (Lord, 2020). Ved å bruke en sikker form for autentisering er man bedre beskyttet mot eksterne angrep, men det er vanskelig å forhindre menneskelige feil som det å trykke seg inn på en forfalsket lenke.

Autentisering ved bruk av passord er mye brukt, men er også utsatt for flere typer angrep. Passord-autentisering har flere svakheter og er ikke ideelt når det kommer til lagring av sensitiv data (Swoop, 2020). Heldigvis finnes det flere metoder som er sikrere enn passord-autentisering:

Multifaktorautentisering

Multifaktorautentisering (MFA) baserer seg på verifisering gjennom bruk av flere forskjellige autentiseringsmetoder. Disse metodene er ofte uavhengige og er ikke knyttet til innloggingsinformasjonen brukeren skriver inn (Swoop, 2020). Et eksempel på en slik innloggingsprosess kan være en vanlig passordautentisering, kombinert med at bruker får en sikkerhetskode tilsendt via SMS som brukeren må skrive inn i nettleseren.

Dersom en angriper har tilgang til en brukers innloggingsinformasjon, må vedkommende også ha tilgang til flere autentiseringsmetoder (Swoop, 2020). Microsoft Authenticator, Microsofts løsning for sikker multifaktorautentisering, tar i bruk prinsippet om multifaktorautentisering. Innlogging ved bruk av denne applikasjonen krever først innloggingsinformasjon som fylles inn i et nettsted, hvor brukeren deretter får mulighet til å autentisere seg via Authenticator-applikasjonen. Et eksempel på et nettsted som

bruker denne typen autentisering er Blackboard, som er e-læringssystemet til NTNU. Ifølge Microsoft er denne typen autentisering sikrere enn vanlig passord-autentisering (Microsoft, n.d.). Multifaktorautentisering har fortsatt noen mangler i form av sikkerhet rundt tilgjengelighet av identifikatorer.

Biometrisk autentisering

En form for autentisering som finnes i flere applikasjoner og plattformer er biometrisk autentisering. En slik type autentisering involverer en fysisk og biologisk egenskap for autentisering. Dette innebærer fingeravtrykk, iris-skanning og ansikts-identifikasjon (ScienceDirect, n.d.). Disse egenskapene knyttes til en enkelt person, ofte gjennom brukernavn.

Måten en slik autentisering fungerer på er at man sammenligner en type egenskap med en lagret egenskap i applikasjonen eller systemet. To eksempler på innlogging ved bruk av biometrisk autentisering er Face ID for iPhone og fingeravtrykk for Android. Nøkkelen her er at biometri er unik for hver bruker. Et sikkerhetsproblem med biometrisk autentisering er sensitivitetsnivå, altså hvor sensitiv en sammenligningstest ved innlogging skal være. Er den for lav, kan resultatet av skanningen kollidere med andre prøver som er lagret, og er den for høy kan man risikere å hindre tilgang til en tidligere autorisert bruker. I tillegg kan en slik type autentisering være komplisert å implementere (ScienceDirect, n.d.). Et annet eksempel er at en angriper kan manipulere biometri eller at angriper har fått tilgang til en brukers biometri. Dette kan være bilder av ansikt som brukes for ansiktsidentifikasjon (OneSpan, n.d.). Likevel finnes det noen løsninger for disse ulempene, som hindrer slike angrep. Biometrisk autentisering er under stadig utvikling og sies å være en bedre og mer sikker metode for autentisering enn passord-autentisering (OneSpan, n.d.).

Sertifikatbasert autentisering

Sertifikatbasert autentisering er en annen metode for autentisering. Autentiseringen er en kryptografisk teknikk som tillater en maskin til å identifisere og autentisere seg på internett. Sertifikatet brukes for å verifisere offentlige nøkler hos en maskin. Slike sertifikater brukes blant annet i TLS/SSL protokollen (Woland, n.d.).

En fordel med sertifikatbasert autentisering er at den kan verifisere både brukere, maskiner og enheter (Yubico, n.d.). Autentiseringer fungerer godt i samarbeid med vanlig innlogging ved passord, da tofaktorautentiseringen er sikkerhetsmessig bedre (Yubico, n.d.). Alle inkluderte parter i en slik forbindelse må identifisere seg, uavhengig av hvilken type forbindelse det er. Dette kan være forbindelser mellom klienter, tjenere eller en klient og en tjener. Før en forbindelse opprettes, må begge parter verifiseres og identifiseres, slik at ingen uønskede eller ondsinnede forbindelser skal oppstå (Olenski, 2016). Sertifikatbasert autentisering er derfor en sikker metode som kan implementeres og integreres i en multifaktorautentisering.

Autentisering ved bruk av token

Token-basert autentisering er også en metode for autentisering av brukere i et system. En token opprettes etter en klient verifiserer sin identitet, ofte gjennom en elektronisk

autentisering (se punkt 2.2.1 om elektronisk autentisering). I retur vil klienten motta en unik token. I og med at en token ikke blir utlevert før etter innlogging, bør systemet i tillegg ha en sikker autentisering av innloggingsinformasjon. Et eksempel på en slik token er JWT, som er presentert i punkt 2.2.1 om elektronisk autentisering.

Token-basert autentisering er en sikker metode for autentisering, da den gir tilgang til bestemte deler av systemet (Okta, n.d.). Innad i systemet kan man begrense tilgang til forskjellige endepunkter, i form av brukerroller og tillatelser (Auth0, n.d.). Tokens kan opprettes omtrent hvor som helst og er liten i størrelse. Det er derfor enkelt å sende slike tokens mellom parter i en forbindelse (Okta, n.d.). Et problem med tokens er at sikkerheten ligger i én enkelt nøkkel. Dersom denne nøkkelen kommer på avveie, vil en angriper ha tilgang til systemet og kan oppgi tokenet ved tjenerkall for å hente ut informasjon (Okta, n.d.). I tillegg kan noen implementasjoner være såpass dårlige, at ondsinnede brukere kan unngå begrenset tilgang til endepunkt, samt hente ut sensitiv eller konfidensiell informasjon (The Hacker Recipes, 2021). Likevel er er tokens sett på som en sikker metode for autentisering, dersom man tar godt vare på nøkkelen deres.

4.1.3 Hvordan sikre effektiv og pålitelig drift av systemet?

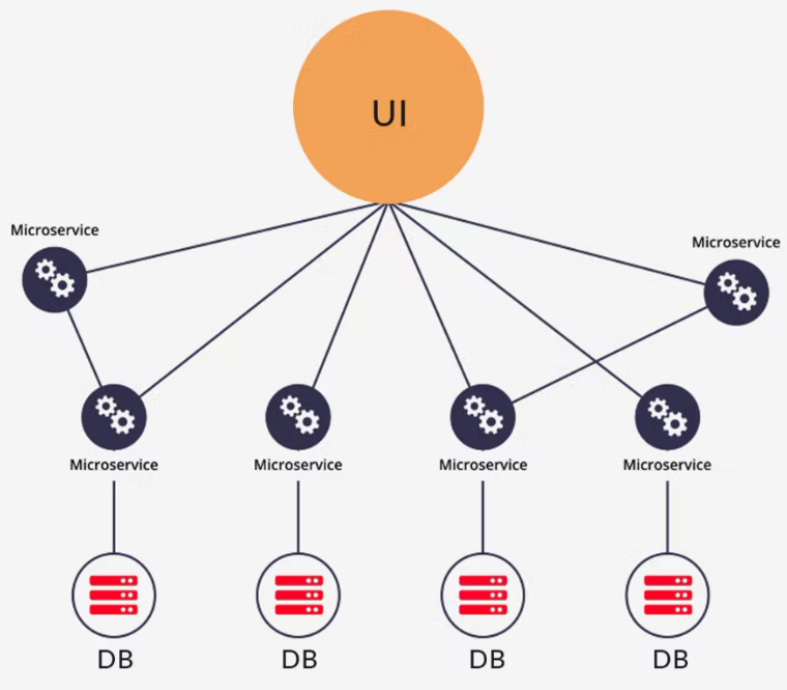
En moderne web-applikasjon som benyttes i dagens samfunn har to generelle egenskaper ved seg som man kan se på for å definere om den bra eller ikke. Den ene egenskapen er hvor pålitelig systemet er. Dette går på at systemet skal ha maksimal oppetid, og minimal nedetid. Med andre ord vil man ikke ha en applikasjon som krever masse vedlikehold eller som krasjer på grunn av feil eller overbelastning i systemet. Den andre egenskapen går på effektiviteten, som kan anses som ytelsen til systemet. Her ser man på hvor responsivt og raskt systemet er. Et tregt system hvor man bruker masse tid på å se på at siden bufrer eller laster inn data er system som kan oppfattes som frustrerende for brukerne å benytte seg av.

Resultatene som er presentert i dette delkapitlet kom vi fram til ved hjelp av å følge den kvalitative forskningsmetoden som presentert i kapittel 3.3. Måten vi kom frem til resultatene var først ved å gjøre dokument-research på internett for å se på hvilke relevante metoder man kan benytte seg av for å sikre effektiv og pålitelig drift av moderne web-applikasjoner. En viktig kilde for informasjon for å besvare dette spørsmålet er ISO 5055 - Software Quality Standards (CISQ, n.d.). Etter å ha funnet en del relevante teorier knyttet opp mot dette gjorde vi et dypdykk inn i det som virket mest relevant. Dette, kombinert med kunnskapen vi satt igjen med etter tre år som dataingeniørstudenter er det vi anser som resultatene, og er dermed også det som er presentert under.

Microservices

Microservices angår arkitekturen til en applikasjon, og påvirker i all hovedsak påliteligheten til applikasjonen. Microservices-arkitekturen går ut på å abstrahere funksjonene til en applikasjon ned i flere mindre funksjoner. I stedet for å konstruere brukergrensesnitt og funksjoner inn i en stor sammenhengende applikasjon og kodebase (monolittisk arkitektur), så abstraherer man altså ut brukergrensesnittet og alle de forskjellige funksjonene ned til små, selvstendige tjeneste-applikasjoner som visualisert i figur 4.1.3.1.

Konkrete eksempler på at å benytte microservices-arkitekturen som øker påliteligheten til en web-applikasjon, er for eksempel at man kan individuelt oppdatere mindre deler av systemet uten at det blir noe nedetid på resten av systemet under oppdateringen. En annen fordel er at ved en eventuell krasj i en av microservicene vil den delen av systemet som benytter microservicen bli utilgjengelig, mens resten av systemet fremdeles vil fungere.



Figur 4.1.3.1: visualisering av microservice arkitekturen (Hentet fra (Perry, 2022)).

Containerisering i docker

Som forklart i kap 3.2.2 så kan man "pakke" en applikasjon inn i en Docker container for å gjøre plattformen uavhengig, samtidig som den blir mer stabil av å kjøre i et kjøretidsmiljø som er isolert fra resten av applikasjonene på verts-operativsystemet. Applikasjonen kan altså ikke "forstyrres" av andre applikasjoner som kjører på samme maskinvare, og driver/programvare-endringer på verts-operativsystemet har ingenting å si siden alt applikasjonen trenger blir pakket inn med den i containeren.

Tråd-optimalisering

Tråd-optimalisering i en web-applikasjon vil si å splitte store komplekse oppgaver ned i mindre oppgaver som kan gjøres parallelt, og kan dermed påvirke effektiviteten drastisk. Dette kan for eksempel benyttes for sortering av ei stor liste eller for å hente inn data fra mange forskjellige nettsteder på internett parallelt. Selv om multithreading er raskere i mange scenarioer så er det ikke alltid man kan implementere det. En grunn kan for eksempel være at algoritmen man bruker ikke kan parallelliseres. En annen grunn kan være at kostnaden til programmet ved at systemet oppretter trådene for så å kjøre de (med andre ord; overheaden), er større enn å bare kjøre algoritmen sekvensielt. Dette er noe som kan beregnes ved bruk av Amdahl's lov (Klein, 2022).

Applikasjons-testing

Under utviklingen av en web-applikasjon er det normalt sett mennesker som gjør alt fra design til programmering. Man kan dermed si at det er uunngåelig at det ikke oppstår defekter eller feil underveis i utviklingsprosessen. Det ligger rett og slett i menneskets natur (Sharma, 2022).

Tester implementert i et system hjelper med å:

- Identifisere defekter i systemet
- Redusere generelle feil i komponenter av systemet.
- Øke den generelle påliteligheten og kvaliteten til systemet.

Stacken til en typisk web-applikasjon består av en frontend, minst en backend og minst en database. Siden de ulike delene av stacken har ulik funksjonalitet og kanskje ulike kodespråk, må de også ha ulike metoder å testes med. Det eneste som er til felles for alle metodene er at de har positive og negative testscenarioer. Et positivt testscenario vil si at testen er oppbygd på en måte som tester funksjonene på måten de er ment å brukes. Negativ testing er det motsatte, hvor man tester funksjonene på en måte som de ikke er ment å brukes på.

Måter å teste forskjellige deler av stacken på:

Frontend (*Web Application Testing - Techniques*, n.d.)

- Brukertesting
- Funksjonalitet-testing
- Brukervennlighet-testing
- Grensesnitt-testing
- Kompatibilitetstesting
- Ytelsestesting
- Sikkerhetstesting

Backend/Database

- Unit-testing
- Integrasjonstesting
- Mocking
- Database-testing

Kodekvalitet

Kodekvalitet er et subjektivt begrep innen programmering som blir påvirket av mange faktorer. Kodekvalitet varierer fra person til person samt innad i ulike programmeringsspråk. Når det angår å se på kodekvaliteten til et system finnes det dermed også forskjellige typer standarder som gjelder i ulike typer programmeringsspråk eller innad i spesifikke bedrifter. Det er uansett noen målbare nøkkelpunkter innen kodekvalitet som generelt gjelder for all kode og sier ganske mye om kvaliteten på den gitte koden (Bellairs, 2019):

- *Pålitelighet*
  Kodekvalitet er med på å påvirke påliteligheten til et system. Ved å se direkte på påliteligheten til et system vil man også delvis kunne bedømme kodekvaliteten.

Påliteligheten kan måles ved å se på sannsynligheten for at et system vil kjøre uten feil over en bestemt driftsperiode.

- *Vedlikeholdbarhet*
  Ved å opprettholde god kodekvalitet så vil vedlikehold av kode bli mye enklere. Et par eksempler på dette er godt dokumentert kode og ingen duplisering av kode. Ved å unngå duplisert kode slipper man å oppdatere et segment med kode flere plasser i koden, men i stedet kanskje bare en plass.

- *Testbarhet*
  Som nevnt i delen om "Applikasjons-testing" så er testing med på å gjøre en web-applikasjon mer pålitelig og effektiv. Men forutsetningen for å kunne implementere tester er at man har produsert testbar kode. Testbarhet blir målt på hvor mange tester som må til for å finne potensielle problemer i systemet.

- *Portabilitet*
  Portabel kode er kode som relativt enkelt kan flyttes over på en annen plattform, uavhengig av maskinvare eller operativsystem. Portabilitet i koden gjør systemet mer pålitelig da man unngår uforutsette problemer med applikasjoner grunnet kompatibilitetsproblemer i maskinvare eller programvare. Portabilitet er vanskelig å måle, men har blitt enklere å forbedre i moderne programvare da man kan benytte verktøy som Docker.

- *Gjenbrukbarhet*
  Kode som er gjenbrukbar går litt igjen i det som ble nevnt på vedlikeholdbarhet, men har i tillegg fokus på at kode skal være så modulær og uavhengig som mulig. Dette gjør at man kan implementere én "beste" implementasjon som man kan gjenbruke flere ganger om nødvendig. De engelske (og mest brukte) begrepene som beskriver et system med god gjenbrukbarhet er: "low coupling and high cohesion". Det finnes automatiske systemer for å sørge for at dette blir passet på under produksjon av kode (typisk integrert i IDEer).

4.1.4 Hvilke effekter gir bruken av det nye systemet sammenlignet med tidligere manuelt system?

Svarene presentert i dette delkapitlet har som mål å svare på forskningsspørsmål nr 4. slik som det står definert i kapittel 1.5.1. Dette spørsmålet har ingen direkte tilknytning til hovedproblemstillingen, men søker heller etter å måle hvor mye av oppdragsgivers og Atea IRTs behov systemet som vi produserte klarte å oppfylle. Metodikken vi skulle følge for å finne svar på dette var opprinnelig tenkt til å være en kvantitativ forskningsmetodikk. Senere i prosjektet endte vi opp med å gå bort i fra dette og over på den kvalitative metodikken definert i kapittel 3.3, som beskriver en spørreundersøkelse. Siden antallet svar på spørreundersøkelsen endte opp med å ikke bli nok til å kunne forsvare et kvantitativt svar, ble det heller omgjort til kvalitativt.

Måten vi gikk fram for å finne svar på effektene AIRS hadde i forhold til behovet oppdragsgiver presenterte til oss i oppgaveteksten, var først å definere selve spørsmålet vi ønsket å svare på. Deretter, for å ha noe å sammenlikne med gikk vi over til systemutviklings-fasen av prosjektet, hvor selve systemet ble produsert. Etter tolv uker med systemutvikling så landet vi til slutt på versjon 1.0 som ble

sammenligningsgrunnlaget til resultatene presentert i dette delkapitlet. For å generere resultatene som svarer på forskningsspørsmålet inviterte vi til et møte med Atea IRT, hvor vi hadde en presentasjon og en demonstrasjon av systemet. Presentasjonens hensikt var å gi en innføring i hvordan systemet fungerer, for så å gi deltakerne mulighet til å teste systemet så mye som de ønsket. Måten de fikk testet det på var ved at vi distribuerte versjon 1.0 av systemet på internett slik at de hadde tilgang til å teste den fra hvor som helst ved hjelp av flere demo-brukere. Vi informerte også om at vi ønsket å få tilbakemelding på hvordan bruken av versjon 1.0 kan sammenlignes med dagens manuelle system. Måten tilbakemeldingene ble samlet inn på var ved hjelp av et anonymt online spørreskjema som vi lagde i Google Forms, og som er vedlagt i *vedlegg J - Spørreundersøkelse*. Svarene vi fikk ligger vedlagt i vedlegg *K - Svar spørreundersøkelse*, og er også presentert i tabellene under (Hadzalic et al., 2022):

Vi fikk totalt 2 tilbakemeldinger på spørreskjemaet. Tall-resultatene er presentert i form av gjennomsnitt, og flervalg er presentert ved hjelp av prosent-svar per alternativ.

Del 1: Bruk av systemet

Spørsmål:	**Svar:**		**Forklaring på svar:**	**Antall svar:**
1. Hvor nyttig tror du AIRS v1 som presentert og testet i demoen, kan være som et verktøy i arbeidsdagen din?	4		Heltall-skala fra 1 til 5 hvor 5 er best.	2
2. Hvor nyttig tror du en fremtidig versjon av AIRS (med bla.a. funksjonalitet for varsling av kunder innad i systemet, en komplett liste med kilder den henter data fra, mulighet for uthenting av statistikk, egen feed som kunder har tilgang til samt andre optimaliseringer) kan være som et verktøy i arbeidsdagen din?	4		Heltall-skala fra 1 til 5 hvor 5 er best.	2
3. Hvordan syns du oppsettet til AIRS v1 med menyer og funksjoner(søk, kommentarfelt, filtrering og redigering av elementer) er?	Menyene og funksjonene var enkle å forstå og bruke.	100%	Alternativer med et alternativ for å svare med egendefinert tekst (utelatt da ingen gjorde det).	2
	Menyene var enkle å forstå og bruke,men funksjonene var ikke det.	0%		
	Funksjonene var enkle å forstå og	0%		

	bruke, men menyene var ikke det			
	Verken menyene eller funksjonene var enkle å forstå eller bruke.	0%		
4. *Frivillig* Vil du tilføye noe ekstra som angår oppsettet av AIRS?	"Brukervennlig grensesnitt, praktisk bruk av funksjonene vil uansett være avgjørende da det er en kompetent brukergruppe."		Frivillig kortsvars-oppgave	1

Tabell 4.1.4.1: Resultater fra del 1 av spørreundersøkelsen i *vedlegg K - Svar spørreundersøkelse* (Hadzalic et al., 2022).

Del 2: Dagens prosesser VS AIRS v1

I denne delen henger spørsmålene sammen to og to. Først ble det spurt etter hvordan dagens prosesser er uten AIRS, og deretter med AIRS. Spørsmålene som henger sammen er representert ved fargen til radene. Den kumulative påvirkningen AIRS har på den spesifikke prosessen er representert i kolonne 3 - "systemets påvirkning".

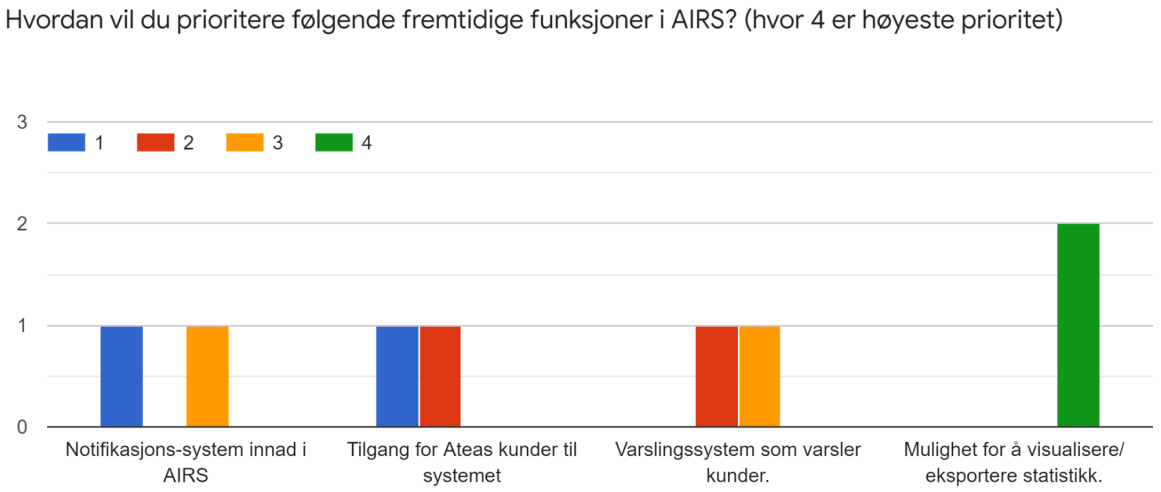
Spørsmål:	Svar:	Systemets påvirkning:	Forklaring på svar:	Antall svar:
5. Hvor manuell syns du dagens prosesser som angår innhenting/oppdaging av sikkerhetsfeil er?	1.5	+2	Heltall-skala fra 1 til 5, hvor 1 er helt manuell og 5 er helt automatisert.	2
6. Hvor manuell syns du AIRS v1 gjør prosessene som angår innhenting/oppdaging av sikkerhetsfeil blir?	3.5			
7. Hvor manuell syns du dagens prosesser vedrørende det å holde seg oppdatert på endringer i sikkerhetsfeil er?	1.5	+2	Heltall-skala fra 1 til 5, hvor 1 er helt manuell og 5 er helt automatisert.	2
8. Hvor manuell syns du AIRS v1 gjør prosessene vedrørende det å holde seg oppdatert på endringer i sikkerhetsfeil blir?	3.5			
9. Hvor effektivt syns du dagens system for å se hvilke kunder som benytter hvilke teknologier er?	4		Heltall-skala fra 1 til 5, hvor 1 er usystematisk/ treigt og 5 er	

10. Hvor effektiv syns du AIRS v1 gjør systemet for å se hvilke kunder som benytter hvilke teknologier blir?	4.5	+0.5	systematisk/raskt.	2
11. Hvor effektivt syns du dagens system for å se hvilke kunder som blir påvirket av en nyoppdaget sikkerhetsfeil eller CVE er?	4.5	+0	Heltall-skala fra 1 til 5, hvor 1 er usystematisk/treigt og 5 er systematisk/raskt.	2
12. Hvor effektivt syns du AIRS v1 gjør systemet for å se hvilke kunder som blir påvirket av en nyoppdaget sikkerhetsfeil/CVE blir?	4.5			
13. Hvor effektivt syns du dagens system for å varsle kunder som blir påvirket av en sikkerhetsfeil er?	1	+2.5	Heltall-skala fra 1 til 5, hvor 1 er usystematisk/treigt og 5 er systematisk/raskt.	2
14. Hvor effektivt syns du AIRS v1 gjør systemet for å varsle kunder som blir påvirket av en sikkerhetsfeil blir?	3.5			

Tabell 4.1.4.2: Resultater fra del 2 av spørreundersøkelsen i *vedlegg K - Svar spørreundersøkelse.*

Del 3: Framtiden til AIRS

Spørsmål 15 :
Diagrammet under spør etter en prioritetsrekkefølge på fremtidige funksjoner i AIRS hvor man måtte gi hver enkelt funksjon en prioritet fra 1-4. Funksjonene kunne ikke få samme prioritet.



Figur 4.1.4.1: Direkte resultater fra del 3 av spørreundersøkelsen i *vedlegg K - Svar spørreundersøkelse.*

Gjennomsnitt av diagrammet over, i synkende prioritet: (4 høyeste prioritet, 1 laveste)

Funksjon:	**Gjennomsnittlig prioritet:**
Mulighet for å visualisere/eksportere statistikk.	4
Varslingssystem som varsler kunder.	2.5
Notifikasjons-system innad i AIRS.	2
Tilgang for Ateas kunder til systemet.	1.5

Tabell 4.1.4.3: Gjennomsnittlig prioritet i synkende rekkefølge fra del 3 av spørreundersøkelsen i *vedlegg K* - Svar spørreundersøkelse.

Følgende svar var frivillige, og det ble ikke gitt full respons på alle spørsmålene. Kolonnen "antall svar" beskriver mengden svar på hvert enkelt spørsmål.

Spørsmål:	**Svar:**		**Forklaring på svar:**	**Antall svar:**
16. *Frivillig* Er det noen fremtidige funksjoner som ikke er nevnt over som du syns bør være med i en fremtidig versjon av AIRS? Gjerne gi den en prioritet.	" "Enkel" metode for adde kilder."		Frivillig kortsvars-oppgave	1
17. *Frivillig* Er det noen funksjoner som allerede ER IMPLEMENTERT i AIRS v1 som du mener ikke trenger å være der?	Nei		Frivillig kortsvars-oppgave	1
18. *Frivillig* Syns du AIRS v1 bør videreutvikles?	Ja	100%	Frivillig oppgave med alternativ for "ja" eller "nei"	2
	Nei	0%		

4.2 Ingeniørfaglige resultater

4.2.1 Systemets egenskaper

Dette punktet baserer seg på målene vi satte for produktets funksjonelle og ikke-funksjonelle egenskaper, som man finner i punkt 5 og 6 i vedlegg F - Visjonsdokument. Produktet inneholder også annen funksjonalitet som oppsto underveis i utviklingsprosessen, men som ikke er presentert i visjonsdokumentet. Disse beskrives også under punktet om *Egenskaper som ikke er inkludert i visjonsdokumentet.*

Produktets funksjonelle egenskaper

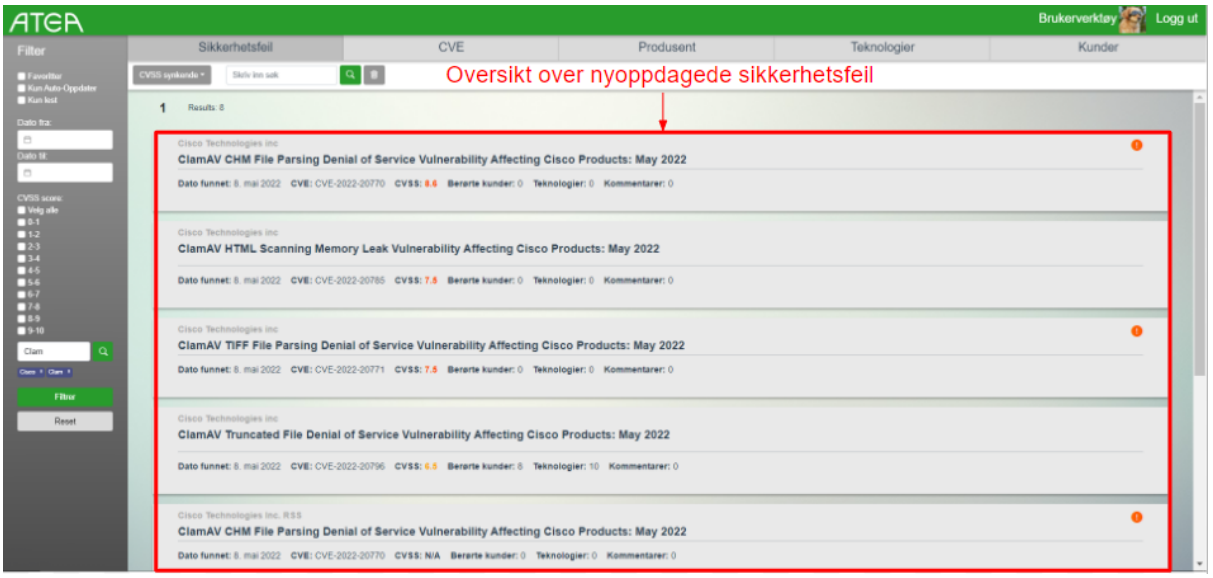
Overvåkingsprogram for innhenting av nye sikkerhetsfeil

I løpet av prosjektet har vi utviklet en webapplikasjon som ligner mye på løsningen som blir beskrevet under punktet om funksjonelle egenskaper i vedlegg F - Visjonsdokument (Hadzalic et al., 2022). Vi henter inn sikkerhetsfeil ved hjelp av web-scraping, som beskrevet i punkt 3.2.1. Innhentingen er tråd-optimalisert, og utnytter 75% av kjernekapasiteten til tjeneren. Vi har også produsert et enkelt rammeverk for å implementere innhenting fra flere forskjellige kilder.

Automatisk kobling mellom sikkerhetsfeil og teknologi er ikke implementert, men man har derimot mulighet til å lage relasjoner mellom sikkerhetsfeil og teknologier manuelt. Dersom man kobler en teknologi sammen med en sikkerhetsfeil, vil man også kunne se hvilke kunder som påvirkes av sikkerhetsfeilen. Dette gjøres gjennom relasjonen som oppstår automatisk ved koblingen mellom teknologier og kunder.

Oversikt over nyoppdagede sikkerhetsfeil

Sikkerhetsfeilene som innhentes blir vist frem i en feed i applikasjonen, hvor det listes opp 25 elementer for hver side. En ny side blir lastet inn etter hvert som man blar mellom sidene. Øverst i feeden vil man ha mulighet til å se den total mengde med sikkerhetsfeil som ble funnet ved enten søk eller filtrering. Dersom filteret er tomt, vil man se det totale antallet sikkerhetsfeil som er registrert i databasen. Man har også mulighet til å hoppe mellom forskjellige sider ved å bruke en nedtrekksliste som viser alle sidetall, som gjør det enkelt å manøvrere mellom ulike sider. Dette er implementert for alle feeder. For hver sikkerhetsfeil kan man se produsent, hvilken type sikkerhetsfeil det er, datoen den ble oppdaget, CVE-nummer og CVSS-score, antall berørte kunder, teknologier og kommentarer, samt om den er markert som favoritt eller auto-oppdater. CVE-en brukes for å kategorisere sikkerhetsfeilen mens CVSS-scoren brukes for å klassifisere hvor kritisk sikkerhetsfeilen er. Vi gir en grundigere beskrivelse av dette senere i kapittelet.



Figur 4.2.1.1: Sikkerhetsfeil-feeden (markert i rødt) bidrar til oversikt over sikkerhetsfeil som er registrert i databasen. Man kan bla mellom sider for å se gjennom alle sikkerhetsfeil. Uleste sikkerhetsfeilene er markert med et rødt merke til høyre, slik at brukere har oversikt over hvilke sikkerhetsfeil som er lest og ikke.

Dersom man trykker på et av elementene i feeden, vil man åpne en popup som viser en mer detaljert oversikt for hver sikkerhetsfeil. Informasjonen i popupen er mer utfyllende, da den inneholder beskrivelse, kommentarer og en liste med alle teknologier og kunder som er knyttet sammen med sikkerhetsfeilen. I popupen har man også mulighet til å endre på sikkerhetsfeilen, blant annet på beskrivelsen, hvilke kunder den gjelder og hvilke teknologier som blir påvirket. I tillegg kan man markere sikkerhetsfeilen som favoritt- og auto-oppdater. Vi kommer tilbake til disse to merkene senere.

Oversikt over Ateas kunder og deres teknologier samt eventuelle sikkerhetsfeil i gitte teknologier

I tillegg til en feed for sikkerhetsfeil har vi også implementert feeder for teknologier og kunder, som også viser 25 elementer per side. I teknologi-feeden har man oversikt over teknologier og antall kunder og sikkerhetsfeil knyttet til teknologien. Kunde-feeden bidrar med oversikt over kunder og antall teknologier relatert til hver kunde, slik at man ser hvilke teknologier og kunder som påvirkes av en sikkerhetsfeil. I likhet med feeden for sikkerhetsfeil har man også mulighet til å åpne popuper med mer detaljert informasjon. I kunde- og teknologi-popupen har man også mulighet til å endre og slette kunder og teknologier. Den nåværende versjonen av applikasjonen viser ikke hvilke sikkerhetsfeil en kunde har inne på kunde-popupen. Vi har gjort det slik at man kan videresendes til relaterte teknologier, slik at man får en oversikt over sikkerhetsfeil inne i teknologi-popupen.

CVSS - Metode for å klassifisere hvor kritisk en sikkerhetsfeil er

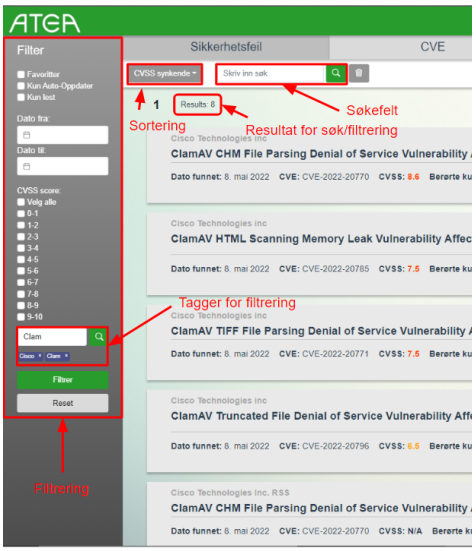
Klassifisering av sikkerhetsfeil gjøres ved å bruk av en CVSS-score. Denne scoren er et desimaltall mellom 0-10, der 0 er minst kritisk og 10 er mest kritisk. Vi endte opp med å integrere denne klassifiseringen i web-scraperen, slik at hver sikkerhetsfeil automatisk blir tildelt en CVSS-score hvis den er tilgjengelig på nettsiden. For enkelte feeder blir det ikke angitt noe score, slik som RSS og Atom-feeds, siden disse inneholder minimalt med informasjon. Her satte vi klassifiseringen blankt, og gjorde det mulig for brukerne av systemet å manuelt sette eller endre på en score for hver enkelt sikkerhetsfeil.

CVE - Kategorier for sikkerhetsfeil

Hver sikkerhetsfeil som blir hentet ut av de forskjellige feedene blir inndelt i noe som heter en CVE, som er et slags oppslagsverk for offentlige cybersikkerhets-feil. "CVE-2022-20770" er et eksempel på hvordan en CVE kan se ut, der de fire sifrene i midten representerer årstallet og sifrene på slutten representerer sekvensnummeret (MITRE, n.d.).

Sortering, filtrering og søk

For hver feed er det lagt inn funksjonalitet for sortering og søk, og hver feed har minst fire forskjellige sorteringsmetoder. Helt i slutten av produktutviklingen, etter distribuering i sky, fant vi ut at sortering kun fungerte i sikkerhetsfeil-feeden. Dette var problemer vi ikke hadde tid til å fikse, da vi prioriterte å få skrevet en god hovedrapport mot



Figur 4.2.1.2: Et bilde av hvordan filtrering, sortering og søk fungerer i sikkerhetsfeil-feeden.

slutten av prosjektet. Søkefunksjonen fungerer som den skal, slik at man kan søke på attributter som er spesifikke for hvilken feed man befinner seg i.

I sikkerhetsfeil-feeden har man også mulighet til å filtrere på ulike attributter som dato, CVSS-score og tagger. Filtrerer man på tagger vil man kunne finne spesifikke sikkerhetsfeil basert på taggene. Disse taggene filtreres slik at man sjekker om de for eksempel finnes i typenavnet til sikkerhetsfeilen eller navnet til en kunde eller teknologi. Man kan også filtrere på favoritter, auto-oppdaterte og leste sikkerhetsfeil.

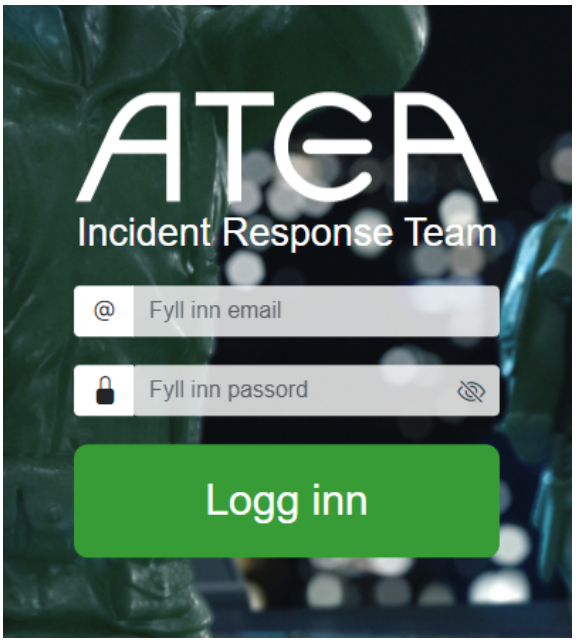
Brukergrensesnitt for å legge til nye kunder med programvarer/teknologier

Inne på oversiktssiden for både sikkerhetsfeil, teknologier og kunder har vi gjort det enkelt å koble disse sammen ved hjelp av oversiktslister. Her kan man søke etter teknologier å legge til for sikkerhetsfeilen, kunder å legge til for teknologien, eller simpelthen bla seg gjennom hver av de registrerte objektene til man finner den man har lyst til å legge til eller fjerne. Når forbindelsen er opprettet blir de automatisk oppdatert i databasen, slik at man enkelt kan manøvrere mellom dem for å se hvilke teknologier som er koblet opp mot hvilke sikkerhetsfeil, eller hvilke teknologier som tilhører hvilke kunder.

I tillegg har man mulighet til å legge til nye kunder og teknologier, da det finnes egne endepunkter for dette. Hvis man velger å manuelt opprette slike objekter vil man også ha mulighet til å relatere det nye objektet til enten kunder eller teknologier, avhengig av hva man oppretter.

Side for innlogging

Web-applikasjonen har en innloggingsside hvor bruker kan skrive inn en e-postadresse og passord. Til høyre kan man se et beskåret bilde av innloggingssiden. Man kan også veksle mellom å se passordet i klartekst eller ikke. Dersom man trykker på "Logg inn"-knappen kan man få forskjellige responser fra nettsiden, basert på om autentiseringen gikk gjennom eller ikke. Hvis e-postadressen ikke har rett format vil man få feilmelding om dette. Dersom mailen har rett format, vil innloggingsinformasjonen sendes sikkert til backend over HTTPS-protokollen. E-postadressen blir sammenlignet med brukere som er registrert i databasen, og dersom e-postadressen tilsvarer e-postadressen til en registrert bruker, vil man prøve å sammenligne hashen til det registrerte passordet med hashen til det innskrevne passordet. Se punktet om bcrypt under delkapittelet 4.1.1.



Figur 4.2.1.3: Et utklipp av innloggingssiden, med felt for mail og passord.

Hvis hashen ikke samsvarer med hashen i databasen, vil man få en feilmelding om at e-postadresse eller passord er feil. Dersom hashene er identiske, vil backend returnere

en JWT-token. JWT-tokenet har e-postadressen integrert i tekststrengen, og blir deretter lagret i nettleserens sessionStorage. Kort fortalt er sessionStorage et verktøy man kan bruke for å eksempelvis lagre en JWT-token, og blir brukt som nettleserens eget minne. Når økten er ferdig, vil sessionStorage automatisk tømmes (MDN Web Docs, 2022). JWT-en kan gjenbrukes i en viss tidsperiode for å få tilgang til endepunkt basert på brukerrolle. Siden for innlogging er responsiv og oversiktlig.

Brukerroller

Vi har lagt inn tre ulike brukerroller på systemet som kan tildeles til en bruker; administrator, autorisert bruker eller uautorisert bruker.

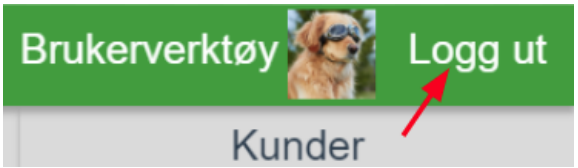
```
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    private static final String[] AUTHORIZED_ADMIN = {
            "/user",
    };
    private static final String[] AUTHORIZED_USER = {
            "/issue/**",
            "/technology/**",
            "/provider/**",
            "/comments/**",
            "/user/**",
            "/customer/**",
            "/cve/**",
            "/alert/**"
    };
    private static final String[] NO_AUTHORIZATION = {
            "/images/**",
            "/user/authenticate",
    };
```

Figur 4.2.1.4: Et utklipp av brukerroller og relaterte endepunkter en rolle har tilgang til.

I systemet vårt har alle brukere en attributt som forteller systemet om brukeren er administrator eller ikke. En administrator har tilgang til alle endepunkt en vanlig bruker har tilgang til, pluss de som er merket med kun administrator privilegier. En autorisert bruker er begrenset til endepunktene som er definert i bruker-kategorien, og ikke de som er for administratorer. I den nåværende versjonen av systemet omhandler administrator-endepunktene kun oppretting og endring av andre brukere. For Atea IRT kan dette bety at en overordnet leder har administrator-privilegier, mens andre medlemmer i Atea IRT har autorisert bruker-privilegier. Uautoriserte brukere kan kun sende forespørsel om autentisering og se bilder på nettsiden.

Utloggingsknapp

Web-applikasjonen har en egen knapp for utlogging i navigasjonsbaren. På bildet til høyre ser man en rød pil som peker på "Logg ut"-knappen. Når denne knappen trykkes vil man logge ut bruker og fjerne variabler fra sessionStorage, samt tømme nettsidens status. På denne måten vil brukeren være nødt til å logge inn på nytt for å få tilgang til nettsiden, da JWT-token er slettet.



Figur 4.2.1.5: Et utklipp av navigasjonsbaren hvor man kan trykke på "Logg ut" for å logge ut.

Mulighet for en innlogget administrator fra Atea til å godkjenne/forkaste/endre nyoppdagede sikkerhetsfeil fra systemet før det settes inn i systemet.

Alle sikkerhetsfeil blir automatisk lagt til i databasen, uavhengig av CVSS-score. Det er lagt til støtte for administrator-privilegier (se punktet om brukerroller), men alle vanlige brukere av systemet har også mulighet for å gjøre endringer på sikkerhetsfeil. Det er ikke lagt inn funksjonalitet for at en bruker skal kunne gjøre endringer på en sikkerhetsfeil *før* den blir lagt inn i databasen.

Ikke-funksjonelle egenskaper

Universelt design iht. WCAG 2.1 standarden

Vi har lagt stor vekt på at brukergrensesnittet skal være enkelt å forstå og manøvrere, og at det er responsivt og forklarende. Knapper og søkefelt har beskrivende tekst, og alle oversiktssider har en tittel som beskriver hva de inneholder. Mye av innholdet på nettsiden er også responsivt ved at det utvides når pekeren holdes over det, eller at pekeren selv endrer utseende basert på hva den peker på. Vi har brukt klare farger på områder som er viktig å legge merke til, slik som regnbuefarget CVSS-score basert på trusselgrad og tydelig markering av sikkerhetsfeil som ikke er blitt sjekket. Det er et gjennomgående tema på nettsiden hvor vi bruker en grå-grønn fargekombinasjon for å utheve funksjonelle knapper og interaktive felt. På store lister er hvert element inndelt i egne bokser slik at man enkelt kan skille mellom hvert listeelement, slik som elementene i hver av feedene eller i listen over teknologier for sikkerhetsfeil og listen over kunder for teknologier. Webapplikasjonen dekker altså mange av retningslinjene for WCAG 2.1, men kan utvides ytterligere for å bli enda mer brukervennlig for andre brukere av systemet.

Behandling av personvernopplysninger

Vi har kryptert all informasjon om kundene i databasen, slik at man ikke direkte kan lese ut informasjon om kundene hvis man skulle få tilgang til databasen. For å kryptere og dekryptere kundeinformasjonen valgte vi å bruke JPA AttributeConverter, som konverterer en primitiv datatype i Java til en spesifisert kolonnetype i databasen og tilbake igjen til en valgt primitiv datatype. Som krypteringsalgoritme brukte vi AES, som er en av de sikreste og mest brukte krypteringsalgoritmene for slik type data (se punkt 2.1.3). I tråd med Datatilsynets anbefalinger bruker vi en 128-bit nøkkel for å kryptere dataen. Denne nøkkelen blir hentet fra en kryptert fil, hvor vi har spesifisert at den skal bruke kun de 128 første bitsene fra filen som nøkkel. Den krypterte filen som nøkkelen hentes fra ligger lagret separat fra applikasjonen. Etter kryptering blir både kundens navn og profilbilde-lenke kryptert, slik at man ikke kan hente ut informasjonen til en kunde i klartekst. I relasjons-tabellen mellom en kunde og en teknologi i databasen er de kun knyttet sammen av en unik ID, og man kan dermed ikke knytte en kunde opp mot en teknologi siden resten av attributtene for kunden er kryptert. Vi har også brukt samme metode for å kryptere brukere av systemets e-postadresser, slik at man i praksis ikke har tilgang til innloggingsinformasjonen til en bruker uten å dekryptere innholdet i databasen. For passord har vi brukt hashing istedenfor kryptering, og valgte å bruke bcrypt som hashing-algoritme (se punktet om bcrypt i delkapittel 4.1.1).

Generell "web-service security" i systemet er minimum iht. OWASP sine retningslinjer for "Web Service Security"

Av OWASP sine tips og råd for sikring av webapplikasjoner har vi implementert støtte for TLS, autentisering og autorisering av brukere samt beskyttelse mot script-injections og attribute-binding.

Egenskaper til systemet som ikke er inkludert i visjonsdokumentet

Web-scraper

Web-scraperen er en av de viktigste delene applikasjonen, og har som hovedoppgave å innhente sikkerhetsfeil fra ulike kilder. Etter innhenting plasseres de i databasen og kobles opp mot en CVE. Måten web-scraping fungerer på er beskrevet i delkapittel 2.4.2.

Web-scraperen er i tillegg tråd-optimalisert, slik at den utnytter tilgjengelige ressurser effektivt. Vi har også laget et rammeverk for implementering av nye innhentingskilder, som for eksempel kan være nettsider, RSS eller Atom-feeds, da web-scraperen har støtte for alle slike typer kilder. I web-scraperen er det i tillegg lagt inn funksjonalitet for automatisk oppdatering av sikkerhetsfeil, slik at man daglig kan hente inn samme sikkerhetsfeil fra en kilde, og se om det er gjort noen endringer på den.

Tråd-optimaliserte backends

Begge backendene i AIRS benytter seg av tråd-optimaliserte funksjoner. For eksempel så benytter web-scraperen seg av multithreading for å lese inn data fra de ulike feedene på internett, mens system-backenden bruker tråd-optimaliserte funksjoner for å effektivisere filtrering og sortering.

CVE og produsent

Etter at funksjonaliteten for innhenting av sikkerhetsfeil var blitt laget, lagde vi en ekstra funksjonalitet for å hente ut CVE-ene til hver sikkerhetsfeil, og lagde nye elementer i databasen for hver CVE. Vi fikk dermed knyttet opp hver enkelt sikkerhetsfeil mot et eget CVE-objekt, som kan brukes for å katalogisere sikkerhetsfeilene. En CVE kan ha flere sikkerhetsfeil, og en sikkerhetsfeil kan ha flere CVE-er. Vi løste dette ved å lage en ny relasjonstabell for mange-til-mange-forholdet mellom sikkerhetsfeil og CVE. Etter at dette forholdet var blitt etablert, fikk vi laget en feed av CVE-er på lik linje som sikkerhetsfeil-feeden. I feeden blir en CVE vist sammen med antall sikkerhetsfeil relatert til CVE-en. Denne feeden er også søkbar og har funksjonalitet for sortering.

Produsent representerer produsenten som innhentingskilden hører til. Eksempler på produsenter som er tatt i bruk under testing av produktet er Cisco Systems og Palo Alto Networks. Dersom man skal legge til en ny kilde er man nødt til å registrere produsenten i databasen før man kjører web-scraperen. Etter at dette er gjort og web-scraperen kjøres, vil alle sikkerhetsfeil tilegnes en attributt som representerer den relaterte produsenten. I web-applikasjonen er det en egen produsent-feed som viser oversikt over produsenter og antall sikkerhetsfeil som er funnet for denne produsenten. I likhet med de andre feedene finnes det også funksjonalitet for sortering og søk.

Dockerisering

Begge backendene til AIRS kjører i hver sin Docker-container som opprettes ved hjelp av egne Dockerfile. Dette ble implementert som følge av ønsket om å enkelt kunne hoste applikasjonen i cloud-tjenere, og for å gjøre web-scraperen tryggere for vertsoperativsystemet siden den jobber opp mot internett.

Database

For å opprette databasen har vi brukt Hibernate-rammeverket, og kommuniserer med databasen ved hjelp av Spring Data JPA (se punkt om database-kommunikasjon i delkapittel 3.2.2). Hibernate bruker Java Persistence API for å opprette databaseobjekter, og kommuniserer med disse ved å bruke JPA Repositories. Databasen som opprettes av JPA er en relasjonsdatabase, og er fleksibel i forhold til hvilken plattform man ønsker å bruke.

Vuex store og lagring av status i hver feed

Et annet verktøy som har vært nyttig i produktet er Vuex store, som kan brukes til å lagre status i hver feed. Dette verktøyet får man bruk for når man blar mellom flere ulike feeder og endepunkt, slik at nettsiden ikke glemmer informasjonen fra tidligere besøkte feeds. (Vuex, 2022). I web-applikasjon lagres elementene som er synlige i feedene, filtre som er brukt, sidetall, valgt element (dersom man har trykket på et element i en feed), og totalt antall elementer i databasen for hver feed. Store lagrer også automatisk rotadressen til backend-system i containeren.

SessionStorage og lagring av brukerattributter

SessionStorage brukes for å lagre JWT-tokenet til en innlogget bruker. I tillegg lagres brukers e-postadresse og brukerrolle, slik at disse variablene kan brukes i hele applikasjonen for autentisering og identifisering.

Oppretting og endring av bruker

En annen funksjonalitet som ikke var med i visjonsdokumentet er oppretting og endring av brukere. Måten vi implementerte dette på var å lage egne endepunkter for oppretting og endring. Oppretting av en bruker gjør man ved å trykke på *brukerverktøy* i navigasjonsbaren. Endepunktet for oppretting er begrenset til brukere med administratorrolle, og kan ikke utføres av andre brukere som ikke har administrator-privilegier. Dersom man oppretter en ny bruker, kan ikke e-postadressen allerede være registrert i databasen.

Endring av informasjon om en bruker kan gjennomføres av en bruker som ikke har administrator-privilegier, men er fortsatt nødt til å være en autorisert bruker. Endringen er naturligvis begrenset til modifisering av ens egen brukerkonto, slik at man ikke kan endre på informasjonen til en annen bruker som er registrert i systemet. Slike endringer kan gjennomføres ved å trykke på profilbildet ved siden av *Brukerverktøy-knappen*. Man blir deretter omdirigert til en ny side hvor man har mulighet til å bytte passord eller profilbilde. E-postadressen som er registrert på brukeren kan ikke endres.

Profilbilder

I web-applikasjonen har vi implementert profilbilder for brukere, produsenter, teknologier og kunder. Profilbildet til en innlogget bruker vil vises i navigasjonsbaren øverst til høyre ved *Logg ut-knappen*. Ellers brukes profilbildene til de andre objektene i deres respektive feeder, og kan sees i både feed og popup-vinduene.

Favoritter, auto-oppdatering og leste sikkerhetsfeil

Hver sikkerhetsfeil har attributter som sier om de er markert som favoritt og lest, og en attributt som forteller om feilen skal oppdateres automatisk hvert døgn. Hvis man befinner seg i en sikkerhetsfeil-popup har man mulighet til å trykke på to avmerkingsbokser som er knyttet til favoritt- og auto-oppdateringsattributten. Når man trykker på disse boksene vil attributtene automatisk endres til motsatt verdi. For eksempel; en sikkerhetsfeil som ikke er markert som favoritt vil dermed markeres som favoritt, og motsatt. Attributter som viser om sikkerhetsfeilen er lest eller ikke vil automatisk settes til lest dersom man åpner en sikkerhetsfeil-popup. I sikkerhetsfeil-feeden vil man ha mulighet til å se hvilke sikkerhetsfeil som er lest og ulest. I figur 4.2.1.1 kan man se at uleste sikkerhetsfeil er markert med et rødt merke.

Disse merkene vil fjernes dersom sikkerhetsfeilen åpnes. I filter-menyen har man mulighet til å filtrere sikkerhetsfeilene ved hjelp av disse attributtene (se punktet om Sortering, filtrering og søk i delkapittel 4.2.1).

Testing av systemet

Tidligere i prosjektet hadde vi tester for den viktigste funksjonaliteten av programmet, blant annet for metodene i service-klassene til sikkerhetsfeil og kunder. Etter hvert som vi utvidet programmet og gjorde endringer på databasekoblingene ble det vanskelig å justere på testene, og det endte opp med at vi heller testet programmet interaktivt ved hjelp av Postman eller testing i frontend.

Egenskaper spesifisert i visjonsdokumentet som ikke er implementert

- Varsling av kunder ved oppdaging av sikkerhetsfeil
- Filter funksjonalitet for andre feeder
- Statistikk
- Mulighet for valg av varslingsmetode
- Grundig logging av viktige deler av plattformen
- Sikker kommunikasjon over internett skjer ved hjelp av HSTS-protokollen

4.2.2 Status på systemet ved innlevering

Vårt sluttprodukt for bachelorprosjektet er en solid og brukbar web-applikasjon. Den har noen mangler, men er fullt funksjonell. Under utviklingsprosessen har vi prioritert å gjøre den implementerte funksjonaliteten så effektiv som mulig, slik som pålitelig innhenting av sikkerhetsfeil og effektiv kommunikasjon med databasen. Enkelte deler av applikasjonen burde utvides, for eksempel antallet innhentingskilder for sikkerhetsfeil eller automatisk varsling av kunder og brukere. Det er lagt inn støtte for å både utvide antallet innhentingskilder og å opprette et varslingssystem, men tiden det krever å implementere disse gjør at disse funksjonalitetene ikke er implementert. Til tross for dette har vi likevel endt opp med et produkt som inneholder den viktigste funksjonaliteten som er beskrevet under delkapittel 3.4 *Brukernes behov* i vedlegg F - visjonsdokument, og sluttproduktet kalles derfor AIRS versjon 1.0.

4.3 Administrative resultater

4.3.1 Prosjektmøter, kommunikasjon og samarbeid med veileder og oppdragsgiver

Gjennom utviklingsprosessen har vi hatt god kommunikasjon med veileder og oppdragsgiver. Vi har hatt veiledningsmøter med veilederen vår, Olav Skundberg, annenhver uke til hver tredje uke. Vi har opprettholdt en god dialog med veileder og har fått gode tilbakemeldinger og konstruktiv kritikk på ulike deler av prosjektet.

Møter og kommunikasjon med oppdragsgiver og Atea IRT har vært solid og bra under hele prosessen. Møtene har vært givende, da vi har lært mye om deres behov og krav for produktet. Vi har også fått god informasjon rundt deres nåværende system og om hvordan vi best kunne utvikle et automatisk system som ville automatisere de manuelle prosessene de gjør i dag. Underveis i prosessen har vi også blitt kjent med en del av Atea IRTs medlemmer, og fått innsikt i hvordan deres arbeidshverdag fungerer. Vi hadde blant annet et møte med oppdragsgiver og seks andre medlemmer fra Atea IRT, hvor vi

presenterte en enkel versjon av produktet, og fikk tilbakemeldinger basert på implementerte funksjonaliteter. Vi har opprettholdt et godt samarbeid med alle parter under hele utviklingsprosessen.

4.3.2 Bruk av utviklingsmetode

Vår utviklingsmetode har vært en tilpasning av scrum-metodikken (se punkt 3.2.3 for en mer detaljert beskrivelse av utviklingsmetoden). Vi gjorde ingen endringer på utviklingsmetoden underveis. Som beskrevet i punkt 3.2.3 varte hver sprint i gjennomsnitt tre uker. Dette førte til at vi fikk nok tid til å både få tilbakemeldinger på utført arbeid og gjort endringer på både web-applikasjonen og dokumentasjon basert på disse. Vi kunne også vise frem tydelig progresjon for veileder, oppdragsgiver og Atea IRT ved at sprintene hadde faste intervaller mellom seg. I punktet om prosjektmøter i delkapittelet 4.3.1 nevner vi at møtene våre ble holdt med to til tre ukers mellomrom. Derfor kan man si at omtrent hvert møte har vært et slags statusmøte for hver sprint.

4.3.3 Milepæler

Alle milepæler for prosjektet ble satt i forprosjektplanen, som blir beskrevet i vedlegg D - Forprosjektplan (Kvarving et al., 2022). Alle milepæler frem til 20. mai er blitt nådd i løpet av utviklingsprosessen, og stort sett alle datoer er fulgt. De eneste unntakene er datoene for godkjenning av visjonsdokument, wireframe-presentasjon og MVP-presentasjon. De oppdaterte datoene er henholdsvis 7. februar, 16. februar og 8. mars.

4.3.4 Tidsplan og tidsbruk

Ved starten av bachelorprosjektet var det satt et mål i arbeidskontrakten om å bruke minst 500 timer per gruppemedlem (Sonen et al., 2022). Til slutt endte vi opp på mindre enn 500 timer. Vår totale tidsplan og tidsbruk kan ses i timelisten vår i vedlegg C - timeliste. Tidsplanen ble satt opp i form av et Gantt-diagram, som vi har fulgt nøye og oppdatert underveis i prosjektet (se vedlegg E - Gantt-diagram).

Kapittel 5: Diskusjon

5.1 Diskusjon rundt vitenskapelige resultater

5.1.1 Hvordan er det gunstig å lagre brukerdata?

Når vi skulle velge hvordan vi skulle sikre brukerdata på systemet var det mange metoder å velge mellom, men vi tok utgangspunkt i Datatilsynets anbefalinger for kryptering av data og behandling av passord.

For kryptering av konfidensiell data anbefalte Datatilsynet å bruke enten AES med 128 eller 156-bits nøkkel, eller RSA med 3072-bits nøkkel. (Datatilsynet, n.d.) Vi endte opp med å bruke AES for å kryptere data i databasen, blant annet fordi denne algoritmen er mye raskere enn RSA og fordi dataen skal lagres statisk i en database (se punkt om sikker lagring av sensitiv data i delkapittel 4.1.1). Vi sjekket også på internett om det var andre krypteringsalgoritmer som egnet seg bedre til kryptering av brukerdata, men fant ut at det var bred enighet om at AES var den desidert beste algoritmen å bruke i slike scenarioer.

Hvilke data som skulle krypteres ble diskutert flere ganger gjennom prosjektet, og vi tenkte i utgangspunktet å kun kryptere koblingen mellom en teknologi og en kunde ved å kryptere primærnøkkelen deres i relasjons-tabellen i databasen. Dette viste seg å være vanskelig å implementere, så vi bestemte oss for å heller kryptere all informasjon om en kunde. På denne måten vil en angriper uansett ikke kunne vite hvilken kunde det gjelder, selv om koblingen mellom teknologien og kunden er kjent.

Datatilsynet anbefalte også å bruke SHA-2 som hashing-algoritme for å hashe passord, men her var det mer uenighet om hvilke algoritmer som egnet seg best. Et blogginnlegg skrevet av Titan fra UiO kritiserte blant annet Datatilsynet for å komme med utdatert informasjon om passordbehandling, og anbefalte istedenfor å bruke enten bcrypt eller scrypt som hashing-algoritme. (Titan UiO, 2016). De listet også opp Argon2 som en potensiell fremtidig hashing-algoritme, men konkluderte med at den var for ny til å implementeres som standard. På internett blant eksperter på området virket det derimot som om denne var en klar vinner for beste hashing-algoritme, så vi vurderte også å bruke denne. Det endte derimot opp med at vi brukte bcrypt som hashing-algoritme, siden denne i tillegg var enklere å implementere på en sikker måte i Spring Boot.

I starten av utviklingsprosessen lot vi passord for både innlogging i database og krypteringsnøkler stå i klartekst i koden vår, noe som vi gjorde endringer på så snart applikasjonen ble lagt ut på internett. Kildekoden ble endret til å bruke konfigurerings-filer med passord når den opprettet kontakt med databasen, og i tråd med Datatilsynets anbefalinger om sikker lagring av krypteringsnøkler endte vi også opp med å lagre disse i ei ekstern kryptert fil. Passordet til den krypterte filen ligger som klartekst i backend, men man har ikke tilgang til den krypterte filen med mindre man får lastet den ned eksternt. Denne lå passordbeskyttet inne på Google Drive under utvikling, og var ikke tilgjengelig for andre enn medlemmer av bachelorgruppen. Den krypterte filen blir også levert som et vedlegg i bacheloroppgaven, slik at den til enhver tid holdes utilgjengelig for uønskede brukere. Krypteringsnøklene for både kryptering av JWT-token

og kryptering med AES kommer fra samme fil, men de bruker ulike deler av innholdet i filen som nøkkel.

Når det gjelder hvor sikkert man skal lagre brukerdata finnes det uendelig mange måter å gjøre dette på, slik som å bruke en lengre nøkkel for kryptering, flere iterasjoner under hashing, lagre krypteringsnøkler på eksterne harddisker eller spesialisert software, kombinere krypterings- og hashing-algoritmer for dypere sikkerhet og andre kjente teknikker. Vi valgte å følge Datatilsynets anbefalinger der vi så at det var gunstig, og fulgte eksperters råd der hvor vi følte at anbefalingene var manglende. På denne måten fikk vi kombinert flere ulike metoder, både kjente og ikke så kjente - slik at vi fikk sikret applikasjonen på en tilstrekkelig måte.

5.1.2 Hvilke metoder er best egnet for sikker autentisering av brukere på forskjellige deler av systemet?

Diskusjon rundt resultatene

Dette forskningsspørsmålet er svært relevant for vårt prosjekt, da vi utvikler et system som skal være sikkert og pålitelig. Et viktig steg for et sikkert produkt er sikker autentisering av brukere som skal bruke systemet.

Autentiseringene som er presentert i delkapittel 4.1.2 er resultater rundt forskningen på hvilke metoder som er best egnet for sikker autentisering. Metodene har ulike egenskaper, som påvirker hvor sikre de er og hvilket bruksområde de innehaver. Den første metoden som gjennomgås i resultatdelen er autentisering ved bruk av passord. Dette er en metode som fortsatt brukes flere plasser på internett, selv om det finnes tydelige angrepsmetoder som kan knekke den. Effekten av phishing-angrep kan for eksempel være stor, da flere brukere gjenbruker samme passord på flere ulike nettsteder (se delkapittel 4.1.2). Mengden phishing-angrep har også økt mye de siste årene (se delkapittel 4.1.2), og vi kan anta at flere og flere er i faresonen for å bli rammet av et slikt angrep. Det kan i tillegg være mange som velger å ha passord som er enkle å huske, uten å vurdere sikkerheten for et slikt passord. På en side er autentisering ved bruk av passord enkelt å implementere og vedlikeholde i et systemutviklingsprosjekt, mens på en annen side er ikke dette tilstrekkelig for å beskytte mot den brede varianten av angrep som finnes i dag. Dette er sannsynligvis en av grunnene til at flere nettsteder og applikasjoner velger å implementere andre former for autentisering. Dersom man veier de positive sidene mot de negative sidene vil man se at passord-autentisering ikke er en veldig sikker måte å autentisere brukere på, men kan gjøres betydelig sikrere om den kombineres med andre metoder.

Blant disse metodene finner vi blant annet multifaktorautentisering, biometrisk autentisering og sertifikatbasert autentisering. Som vist i punkt 4.1.2, gjør multifaktorautentisering at angriperen må ha tilgang til enda en identifikator for brukeren. Ved bruk av blant annet sikkerhetskoder for å fullføre autentiseringen, kan dette bidra til å gjøre autentiseringen enda sikrere. På denne måten kan et angrep mot multifaktorautentisering være vanskelig å gjennomføre, selv om det likevel kan skje dersom en angriper har tilgang til både innloggingsinformasjon og andre identifikatorer.

Biometrisk autentisering er også en type autentisering som kan anses som relativt sikker, og tar utgangspunkt i en brukers biometri. For en angriper vil det være langt

vanskeligere å få tilgang til slik informasjon, sammenlignet med andre typer identifikatorer. En brukers biometri er unik, og gjør at denne typen autentisering er særdeles sikker mot eksterne angripere.

På den ene siden kan man argumentere for at det ikke er like vanlig å bruke denne formen for autentisering, da den er vanskeligere å implementere enn andre typer autentisering. På den andre siden kan man se hvor sikker slik autentisering er ved å se på *hvilke* aktører som velger å implementere denne typen autentisering. Store applikasjoner som nettbanker, Facebook og Vipps er eksempler på slike aktører, noe som gir et godt grunnlag for å si at biometrisk autentisering er en relativt sikker autentiseringsmetode. Biometrisk autentisering er fortsatt relativt nytt i teknologihistorisk sammenheng, så man vil nok se at flere og flere applikasjoner tar i bruk denne teknologien over tid.

Sertifikatbasert autentisering er også en form for autentisering som kan brukes i sammenheng med andre metoder. Som nevnt i delkapittel 4.1.2 inneholder et sertifikat en nøkkel som er kryptert på en sikker måte, som gjør at sertifikater er sikre mot flere typer angrep. Ved å kombinere denne typen autentisering sammen med andre former, kan man oppnå en sikker brukerautentisering.

Den siste formen for autentisering som ble forsket på i punkt 4.1.2 er token-basert autentisering. Denne typen autentisering kan være mer relevant for identifisering av brukere etter at de har logget seg inn på plattformen. Gjennom bruk av tokens kan man få et sikkert system ved at tokenet kan brukes for å identifisere hvilken rolle en bruker har på systemet. På denne måten kan systemet også forhindre at en angriper får tilgang til endepunkter som kun er reservert for administratorer eller vanlige brukere. Dette kan være endepunkter som henter ut brukere og deres personopplysninger. På en annen side kan en erfaren angriper unngå disse kontrollene dersom systemet har for dårlig implementasjon av token-basert autentisering. Dette vil naturligvis sette hele systemet og dets brukere i fare. Er man forsiktig og forsikrer seg om at man implementerer tokens på en sikker måte, vil man sannsynligvis unngå de fleste angrep.

Diskusjon rundt systemets implementasjon av autentiseringsmetoder

I denne versjonen av systemet har vi valgt å bruke vanlig passord-autentisering som autentiseringsmetode. I og med at denne typen autentisering kan være den minst sikre av de som er presentert, vil man naturligvis stille spørsmål til sikkerheten rundt systemet vårt. Medlemmene i gruppen er enige om at dette ikke er en ideell brukerautentisering, da det finnes bedre alternativer. Grunnen til at vi likevel har valgt å gå for denne metoden er for at versjon 1.0 i all hovedsak kun skal benyttes av Atea IRT. Antallet brukere som får tilgang til plattformen vil derfor være begrenset. Hadde systemet vært tilgjengelig hvor som helst på internett ville dette kanskje påvirke prioriteringen rundt brukerautentisering, men vi har besluttet at det ikke finnes et umiddelbart behov for en sikrere autentiseringsmetode i denne versjonen av produktet. Hadde vi hatt bedre tid mot slutten av utviklingsprosessen, ville vi derimot sett på andre alternativer. Et alternativ vi vurderte å implementere var multifaktorautentisering gjennom Microsoft Authenticator eller sikkerhetskode via epost. Dersom produktet skal videreutvikles vil vi mest sannsynlig implementere en av disse.

Systemet bruker derimot token-basert autentisering, og sikrer dermed at kun autoriserte brukere har tilgang til spesifiserte deler av systemet. Vi har tatt i bruk JWT-tokens, da vi allerede hadde kjennskap til teknologien og visste hvordan vi kunne bruke slike tokens for å verifisere brukere etter innlogging. Ved omtrent alle tjenerkall vil man være nødt til å oppgi en JWT-token. Et unntak er blant annet endepunktet som svarer til innlogging av en bruker. Backenden som er knyttet til frontend bruker Spring Boot-rammeverket (se punktet om REST API i delkapittel 3.2.2). Spring Boot gjorde det relativt enkelt for oss å implementere JWT-tokens, da begge kan relateres til Java og det finnes støtte for dette i rammeverket. Ved tidligere erfaringer har vi opplevd at JWT-tokens er en type autentisering som er sikker hvis den implementeres riktig, noe som førte til at vi også valgte å implementere denne typen autentisering på plattformen vår.

5.1.3 Hvordan sikre effektiv og pålitelig drift av systemet?

Grunnen til at vi ønsker å svare på forskningsspørsmål nr. 3 (tittelen til dette delkapitlet) er at definert i problemstillingen vår (definert i kap 1.5) står det blant annet at vi ønsker å svare på hva som må til for at systemet skal bli så pålitelig som mulig. Effektivitet er en del av spørsmålet forbundet med problemstillingen, da vi fant tidlig ut at det går hånd i hånd med pålitelighet.

Forholdet til effektivitet og pålitelighet kan sees ved at hvis man for eksempel tester store deler av en applikasjon med hensikt å øke effektiviteten til applikasjonen, vil man jo også indirekte også øke påliteligheten til applikasjonen som presentert i kap 4.1.3. Grunnen til det er jo at testene kan avdekke feil eller problemer i systemet i selv om man benytter de til å effektivisere systemet. Videre så kan man jo se på bruk av tråd-optimaliseringsmetoden som også er nevnt i kap 4.1.3. Her kan man jo støte på flere problemer og ende opp med et mindre pålitelig system om tråd-optimaliseringen som er implementert ikke er implementert på riktig måte. Et eksempel på dette kan være at man ikke passer på om flere tråder skriver til samme fil samtidig i systemet, noe som vil føre til en systemkrasj.

På den andre siden så kan man argumentere for at tråd- optimalisering også kan være med på å bidra til at systemet blir mer pålitelig. I web-scraperen til AIRS har vi et godt eksempel på det; en simpel beskrivelse av web-scraperen sin implementasjon av tråder er at feedene som scrapes kjører parallelt i hver sin tråd. Dermed, hvis det oppstår et problem i en av trådene som fører til krasj, vil den krasjen bli begrenset til kun den tråden det gjelder. Dette gjør at selv om en av trådene krasjer og den respektive feeden ikke ble scrapet, så fortsetter selve web-scraperen å scrape andre feeder i de andre trådene. Man krasjer altså ikke hele applikasjonen, men bare selve scrapingen for en spesifikk feed, noe som gjør applikasjonen mer pålitelig da det ikke blir nedetid på grunn av et problem med en enkelt feed.

Det første som ble gjort i prosessen, som ledet fram til resultatene presentert i kap 4.1.3, var å definere selve forskningsspørsmålet. Selve spørsmålet ble som tidligere nevnt lagd med tanke på å konkret kunne svare på en del av hovedproblemstillingen, altså delen som angår pålitelighet. Dette ser vi på selve formuleringen av spørsmålet da det direkte spør etter hvordan man kan sikre effektiv og pålitelig drift av systemet. Man kan jo også si at siden spørsmålet spør etter både effektiviteten og påliteligheten til systemet, at spørsmålet henger mindre sammen med problemstillingen. Dette grunnet at effektivitet ikke er nevnt i hovedproblemstillingen. Alikevel, siden spørsmålet om

pålitelighet henger såpass tett sammen med effektivitet som diskutert i det forrige avsnittet, ble det avgjort at det hører hjemme som en del av forskningsspørsmålet.

Den neste delen av prosjektet som ledet til resultatene var systemutviklings-delen. Altså delen som er den største delen av prosjektet da den gikk over ca tolv uker. Måten systemutviklingen bidro til å finne svar på hvordan man kan sikre effektiv og pålitelig drift av systemet, var det store fokuset på nettopp det under utviklingen av systemet. Dette betyr at vi måtte forske på hva som er viktig å tenke på ved implementasjon av de forskjellige delene til AIRS-systemet. Et eksempel på en implementasjon i AIRS som krevde forskning rundt temaet pålitelighet er bruken av Docker for å containerisere. Det ble implementert nettopp for å simplifisere distribusjonen av AIRS på forskjellige enheter samt gjøre det mer pålitelig. Funksjoner som tråd-optimalisering av forskjellige deler av de to backendene er også noe som kan nevnes bidro med kunnskap rundt pålitelighet og effektivitet.

Etter de tolv ukene med systemutvikling så kom vi til selve forskningsdelen av prosjektet, som også var den siste delen av prosjektet. Denne delen gikk ut på å følge forskningsmetoden som ble definert i kap 3.3, noe som for det meste gikk ut på dokument-forskning. Når vi ser tilbake på rekkefølgen til prosessene som ledet til resultatene så kan vi argumentere for at den skulle vært annerledes. Rekkefølgen vi endte opp med var; definere forskningsspørsmål, systemutvikling, forskning. En mer ideell rekkefølge kunne vært; definere forskningsspørsmål, forskning, systemutvikling. Ved å ha gjort forskningsdelen før systemutviklings-delen kunne vi ha hatt mer tid til å implementere de punktene vi allerede fant ut det er viktig å benytte seg av for å produsere et pålitelig system. Altså, forskningsdelen hadde blitt et forarbeid som legger et bedre grunnlag for systemutviklinga slik at under utviklingen så kunne vi ha brukt mer energi på og kun implementere metoder for å øke påliteligheten samt effektiviteten til systemet. Derav ville vi måtte bruke mindre tid på forskning akkurat i den fasen. Enda en konsekvens av å gjøre ting i den rekkefølgen er at det hadde blitt enklere å bygge systemet opp rundt effektiv og pålitelig drift da vi hadde sittet på den kunnskapen fra start, noe som har kunne bidratt til et mer pålitelig sluttprodukt. Vi kan isåfall anta at det for eksempel hadde blitt mer fokus på automatiske tester samt bruk av tråd-optimalisering fra starten av systemutviklings delen.

5.1.4 Hvilke effekter gir bruken av det nye systemet sammenlignet med tidligere manuelt system?

Det første vi skal se på er mengden resultater og måten vi gikk fram for å produsere resultatene på. Vi hadde opprinnelig planlagt å svare på dette spørsmålet ved hjelp av en kvantitativ metodikk, da systemet vi har utviklet ikke bare påvirker oppdragsgiveren men et helt team som jobber under ham. Dette ville reflektere meningene til hele teamet som målgruppe for AIRS og forhåpentligvis hjelpe med å svare på hvilken grad vi klarte å oppfylle behovet som oppdragsgiver spesifiserte i oppgaveteksten til prosjektet. I realiteten viste dette seg å være vanskeligere å gjøre enn først antatt. Teamet til oppdragsgiver og oppdragsgiveren selv hadde daglige oppgaver som måtte gjøres og vi hadde relativt kort tid på å forske på effekten til AIRS. Dette førte til at kun 2 personer fra Atea hadde mulighet å komme til presentasjonen vi holdte av systemet før vi ga dem tilgang til demoen. Noe vi kunne ha gjort for å øke sjansen for flere deltagere på presentasjonen vi hadde, er å planlegge møtedato og sende ut invitasjoner mye tidligere

i prosessen. Det kunne rett å slett vært planlagt som en milepæl i forprosjektplanen (vedlegg D - Forprosjektplan) fra start.

Den neste delen av å generere resultatene innebar at de skulle prøve selve systemet og gi tilbakemelding gjennom et spørreskjema. For å unngå at de som skulle teste systemet brukte tid på å selv hoste systemet, valgte vi å distribuere systemet ved hjelp av Netlify, slik at de kunne teste det ved å gå inn på et forhåndsbestemt webområde i nettleseren. På denne måten kunne vi ha siden oppe og kjøre slik at de kunne gå inn på siden når som helst og teste den slik som de ville. Spørreskjemaet de svarte på var også tilgjengelig via nettleseren, slik at de til enhver tid kunne teste systemet og gi tilbakemelding på spørreskjemaet når de hadde tilstrekkelig med tid.

Vi ble nødt til å ta ned den distribuerte versjonen av applikasjonen tidligere enn antatt, og endte derfor opp med å få langt færre svar på spørreundersøkelsen enn vi først hadde anslått. Grunnen til at vi måtte ta ned den distribuerte versjonen var fordi oppdragsgiver uttrykte bekymring for at systemet ble indeksert av Google. Dette kunne teoretisk sett lede til at en ekstern angriper kunne funnet siden og brute-force seg inn på nettsiden. Siden brukerne som var registrert i databasen hadde svært enkle passord hadde det ikke vært noe problem å brute-force seg inn, og vi kunne dermed risikere at Atea blir offentlig eksponert for å ha dårlig passord-sikkerhet og dårlig sikret kundedata. Data som lå inne i systemet hadde ikke noe som helst slags signifikant betydning da det bare var fiktiv demo-data vi fylte inn i distribusjonen, men vi ville fortsatt ikke utsette Atea for kritikk ved å la nettsiden stå åpen.

En løsning vi vurderte til dette problemet var å distribuere siden på nytt uten relasjoner til Atea (for eksempel logoer og titler), men det viste seg å være for tidkrevende i forhold til tiden vi hadde igjen på å gjennomføre forskning rundt effektene av AIRS. Det ble derfor avgjort at vi heller skulle ta de resultatene vi fikk, og endre fra kvantitativ til kvalitativ forskningsmetodikk for å svare på forskningsspørsmålet.

Diskusjon av resultater på spørreundersøkelsen

Del 1

Del 1 av spørreundersøkelsen spør om den generelle opplevelsen av systemet. I spørsmål 1 ser vi at gjennomsnittet av svarene heller mot at de tror de kan ha svært god nytte av applikasjonen som et verktøy i arbeidsdagen sin. I tillegg så ser vi i spørsmål 3 at de synes at menyene og funksjonene var enkle å forstå og bruke. Ved å bruke forhåndsbestemte alternativer for svarene får ikke deltakerne begrunnet om en spesifikk funksjon eller meny virket uforståelig, så vi la vi til et alternativ hvor man kunne definere et selvskrevet svar. I tillegg var spørsmål 5 en frivillig kortsvarsoppgave hvor de kunne legge til noe i responsen angående oppsettet på AIRS.

En konsekvens av å inkludere den frivillige kortsvarsoppgaven var at vi fikk en tilbakemelding som poengterte at oppsettet ble opplevd som brukervennlig, og at det er viktig med mulighet for praktisk bruk av funksjonene siden brukergruppen er kompetent. Man kan tolke dette som at det kanskje er overveldende mange funksjoner på enkelte steder i systemet, men de bør likevel være der siden de tilbyr viktige funksjonaliteter for å navigere systemet.

Del 2

Del 2 av spørreundersøkelsen spør om deltakerne tror at AIRS versjon 1.0 øker eller forbedrer enkelte prosesser i arbeidsdagen til Atea IRT. Prosessene det spørres om kommer direkte fra oppgaveteksten som ligger til grunn for hele prosjektet. Vi har også inkludert to spørsmål som omhandler funksjonalitet rundt CVE og det å holde seg oppdatert på sikkerhetsfeil.

Ved å se på resultatene ser vi at deltakerne generelt sett tror at systemet kommer til å automatisere og systematisere viktige prosesser, selv ved en så tidlig implementasjon av systemet. De eneste prosessene de tror at AIRS ikke gir en stor forbedring på er de prosessene som omhandler å se hvilke teknologier de forskjellige kundene benytter, og hvilke CVE-er kundene blir påvirket av. Her kan vi anta at de allerede har systemer på plass som enkelt løser disse oppgavene. Selv om AIRS ikke nødvendigvis gir noen forbedring for disse prosessene, kan vi anta at systemet gir tilsvarende god effekt som deres nåværende metoder.

Spørsmål 13 og 14 hadde som mål å sammenlikne Atea IRTs nåværende metode for varsling av kunder, med AIRS 1.0 sin varsling av kunder. Resultatet for disse spørsmålene var overraskende, da det viser at deltakerne tror at de har stor forbedring på denne prosessen ved å benytte AIRS 1.0. AIRS versjon 1.0 har ingen funksjonalitet for varsling av kunder så man kan anta at de trodde at de svarte for en fremtidig versjon av systemet som har all ønsket funksjonalitet implementert. Hvis vi derimot antar at de leste beskrivelsen til del 2 av spørreundersøkelsen vet de at de skulle svare opp i mot funksjonaliteten som var implementert i AIRS versjon 1.0, som kan bety at koblingene systemet lager mellom sikkerhetsfeil, teknologi og kunder gir såpass god oversikt over hvilke kunder som blir påvirket av en spesifikk sikkerhetsfeil, at de kan benytte denne informasjonen for å mer effektivt sende ut varsling til kundene sine.

Del 3

Det siste vi skal se på er del 3 av spørreundersøkelsen, som har som mål å kartlegge fremtiden til systemet. For det første kan vi argumentere for at fremtiden ser lys ut for systemet, da vi fikk 100% positiv tilbakemelding på at de synes systemet bør videreutvikles på spørsmål 18. Dette gjør spørsmål 16. mer relevant, et spørsmål som bidrar med å gi oss som utviklere en prioritetsrekkefølge på fremtidig funksjonalitet. Her ser vi stor enighet i hva som har største prioritet og litt uenighet i de andre prioritetene. Vi ser at funksjonalitet rundt å visualisere/eksportere statistikk får soleklart høyeste prioritet, noe som viser til hvor viktig regelmessig kommunikasjon med brukergruppen av systemet er under utviklingen av det. Dette kan vi argumentere for da funksjonalitet rundt å visualisere/eksportere statistikk var et ønske som kom underveis i utviklingen av systemet, under MVP-presentasjon. Det endte faktisk opp med høyere prioritet enn varslingssystem for kunder som i motsetning til statistikk sto spesifisert i den originale oppgaveteksten. Dermed, ut ifra dette så kan vi faktisk også styrke opp under at den smidige utviklingsmetodikken som vi fastsatte i prosjektet fungerte.

5.2 Diskusjon rundt ingeniørfaglige resultater

5.2.1 Systemets egenskaper

Produktets funksjonelle egenskaper

Overvåkingsprogram for innhenting av nye sikkerhetsfeil

Ett av punktene vi har måttet endre på i løpet av utviklingen av applikasjonen er om den automatisk skal tildele teknologier til sikkerhetsfeil som blir hentet inn eller ikke. Atea IRT ønsket i hovedsak at dette skulle gjøres automatisk, men vi fant ut relativt tidlig i prosessen at dette var ekstremt vanskelig å programmere. Grunnen til dette er blant annet fordi det finnes ekstremt mange forskjellige teknologier, og at de ikke nødvendigvis var listet opp på samme sted på nettsiden for hver sikkerhetsfeil. For at web-scraperen automatisk skulle funnet alle teknologiene som blir påvirket av en sikkerhetsfeil måtte den enten ha implementert maskinlæring eller brukt et avansert søkesystem til å finne teknologier blant alle registrerte teknologier hos Ateas kunder. Gjennom samtaler med Atea IRT fant vi ut at vi heller skulle la denne prosessen være manuell, slik at det ikke finnes rom for feil ved koblingen mellom teknologier og sikkerhetsfeil. For å gjøre denne prosessen enklere for brukere har vi implementert et listesystem hvor man kan søke på registrerte teknologier, og enkelt legge dem til på en sikkerhetsfeil.

Oversikt over nyoppdagede sikkerhetsfeil

Hvert sikkerhetsfeil-kort viser mye relevant informasjon, da man teknisk sett ikke skal ha behov for å åpne en sikkerhetsfeil for å vite de viktigste delene om en sikkerhetsfeil. Kortene er enkle å utvide, da man kan ha behov for å legge til mer informasjon. I sikkerhetsfeil-feeden vises maksimalt 25 kort. Dette er for å effektivisere innlastingen, spare ressurser og vise minimalt med informasjon i frontend på grunn av sikkerhetsmessige grunner. Vi ble enige med oppdragsgiver og Atea IRT om at dette ville være en god løsning. Innlastingen vil naturligvis være raskere på en server-maskin, men vi har uansett optimalisert fremvisningen av sikkerhetsfeil-feeden på denne måten. I tillegg la vi til nedtrekksfunksjonalitet for enkel hopping mellom sider. Grunnen til dette er at man skal slippe å bla mellom alle sidene for å komme seg til sidetall X. Vi tenkte også at det kunne være fint for brukere å se hvor mange sikkerhetsfeil som passet for et gitt filter. I sikkerhetsfeil-feeden har vi gjort det vi kan for å gjøre siden så oversiktlig, brukervennlig og funksjonell som mulig.

I sikkerhetsfeil-popupen finnes det noen funksjoner som kan videreutvikles. Et eksempel er oversikten over kunder og teknologier. En ideell oversikt vil kanskje implementere samme kort som vises i teknologi- og kunde-feeden. Akkurat nå står det kun navn i oversiktslistene, da vi fant ut at det ville være bra nok for versjon 1.0.

Oversikt over Ateas kunder og deres teknologier samt eventuelle sikkerhetsfeil i gitte teknologier

Teknologi- og kunde-feeden følger litt av samme funksjonalitet som sikkerhetsfeil-feeden. Etter implementeringen av nedtrekksliste for sidetall og et felt for antall elementer for filtreringssøket i sikkerhetsfeil-feeden, gjorde vi det samme for teknologier og kunder. Vi har ikke testet med så mange kunder eller teknologier under utviklingsprosessen, slik at vi ikke har fått direkte bruk for disse funksjonene. Likevel tenker vi at det er en brukbar løsning for fremtidig bruk.

Et punkt som nevnes i resultatdelen angående kunde-popupen er at det ikke vises hvilke sikkerhetsfeil en kunde har inne i kunde-popupen. Man har likevel mulighet til å se relaterte sikkerhetsfeil, dersom man trykker seg innpå relaterte teknologier. En tydelig oversikt over sikkerhetsfeil er noe som kan legges til dersom produktet skal videreutvikles. Man trenger bare å opprette endepunktet for det i backend og deretter bruke samme endepunkt i frontend ved åpning av popup.

CVSS - Metode for å klassifisere hvor kritisk en sikkerhetsfeil er

I utgangspunktet så vi for oss at man skulle manuelt klassifisere hvert enkelt sikkerhetsfeil som kritisk eller ukritisk etter hvert som de ble hentet inn og sjekket av en bruker. Etter et møte med Ateas Incident Response Team fastslo vi at denne klassifiseringen kom til å ta utgangspunkt i en CVSS-score som blir angitt på de fleste nettsider. Atea IRT hadde allerede brukt CVSS-score som klassifisering. Derfor var det naturlig å inkludere det i vår løsning.

CVE - Kategorier for sikkerhetsfeil

CVE var noe som vi i utgangspunktet ikke hadde implementert i systemet, frem til vi hadde et møte med Atea IRT hvor de spesifiserte at disse var viktige for å kategorisere en sikkerhetsfeil. På grunn av dette omstrukturerte vi koblingen mellom en CVE og en sikkerhetsfeil, slik at en sikkerhetsfeil nå kunne registreres med en spesifikk CVE, samtidig som den blir gitt en attributt som forteller hvilken produsent feilen kommer fra. CVE fikk også sin egen feed, slik at brukerne av systemet enkelt kan gå inn på hver enkelt CVE og se hvilke sikkerhetsfeil de inneholder. Før denne endringen hadde hver sikkerhetsfeil en liste over CVE'er som en egen attributt i databasen, men denne løsningen gjorde det vanskelig å hente ut informasjon om hver enkelt CVE.

Ved å inndele hver enkelt sikkerhetsfeil inn i sin respektive CVE gir det en enkel måte å kategorisere hver sikkerhetsfeil på, samtidig som det gjør det enklere for brukerne å vite hvilken type sikkerhetsfeil det er snakk om. Siden det finnes en egen nettside som identifiserer, definerer og katalogiserer CVE'er kan disse effektivt brukes for å identifisere hvilken type feil det er snakk om, og hvor omfattende feilen er.

Sortering, filtrering og søk

Vi startet med å implementere en felles filtermeny, som ville fylles med ulike filtre basert på hvilken feed som var åpnet. Underveis i prosessen fant vi ut at filtrering var viktigst i sikkerhetsfeil-feeden, da hver sikkerhetsfeil har tydelige attributter man kan filtrere på. Filtrering er derfor kun implementert i sikkerhetsfeil-feeden, da vi ikke så et stort behov for det i første versjon av applikasjonen. Hver feed har tross alt søkefunksjonaliteten implementert, så det vil være en form for filtrering på hver feed uansett. En grunn til at filtrering ikke er tatt med i flere feeder er derfor på grunn av prioriteringer. Vi valgte å fokusere på annen type funksjonalitet som var viktigere.

I punktet om resultater rundt sortering, filtrering og søk i delkapittel 4.2.1 nevner vi at sortering kun fungerer på sikkerhetsfeil-feeden. I løpet av utviklingsprosessen fikk vi laget funksjonalitet for sortering av alle feeder. Sorteringen fungerte fint noen uker før innlevering, før vi gjorde om systemet til en skybasert løsning. Vår teori er derfor at det må ha skjedd noe ved sammenfletting av git-grener, som kan ha ødelagt eller gjort om på noen av metodene for sortering.

Brukergrensesnitt for å legge til nye kunder med programvarer/teknologier

Å ha en enkel måte å koble kunder opp mot spesifikke teknologier er en av hoved-funksjonalitetene ved applikasjonen, og er noe vi fokuserte på å gjøre så enkelt som mulig. Grunnen til at denne funksjonaliteten er viktig, er fordi denne koblingen også videreføres når man kobler en sikkerhetsfeil opp mot en teknologi. Man kan altså få en fin oversikt over hvilke kunder som påvirkes av en sikkerhetsfeil ved at den kobles opp mot ulike teknologier. Dette var også noe som Atea IRT påpekte at de ville ha et enkelt system for, slik at prosessen ikke ble like manuell.

Ved at man kan søke gjennom en liste med teknologier og kunder for å legge dem til på deres respektive sider, slipper man å bruke unødvendig lang tid på å lete etter de riktige elementene å legge til. Søketeksten har også auto-complete, som kommer med forslag om hvilke elementer man leter etter underveis i søket. Å fjerne en registrert teknologi på en sikkerhetsfeil, eller en kunde på en teknologi er også svært enkelt, men mangler søkefunksjonaliteten som man har når man skal legge til nye elementer. Dette gjør at hvis det er svært mange registrerte elementer er det litt mer tidkrevende å finne frem til riktig element før man får fjernet dem.

Side for innlogging

Visuelt sett er innloggingssiden en fin side, hvor bruker enkelt kan logge seg inn i systemet. Når det gjelder type autentisering, så har vi begrunnet dette i punktet om diskusjon rundt systemets autentisering i delkapittelet 5.1.2. Vi synes at innlogging for web-applikasjonen var et viktig punkt å ha med og er fornøyde med resultatet for sluttproduktet. Ved videreutvikling vil vi vurdere en annen type autentisering som er mer sikker enn passord-autentisering.

Brukerroller

Grunnen til at vi valgte å dele brukerrollene inn i tre ulike roller var fordi vi ville begrense hva ulike brukere av systemet kunne gjøre inne på applikasjonen. Hvis man ønsker mer kontroll over hva som skjer inne på plattformen er det derfor greit å ha en administratorrolle som har tilgang til all funksjonalitet, en autorisert brukerrolle som kan utføre det meste av viktig funksjonalitet og en uautorisert brukerrolle kun kan utføre et begrenset utvalg av funksjonaliteter. Dette er også et system som er enkelt å utvide ved fremtidige implementasjoner, hvis man for eksempel ønsker et system hvor eksterne kunder kan logge seg inn for å se på sikkerhetsfeilene sine.

Utloggingsknapp

Utloggingsfunksjonen forsikrer oss om at brukers nåværende JWT blir slettet ved utlogging. Siden JWT er slettet, må bruker logge inn på nytt for å komme seg forbi innloggingssiden. Sikkerhetsmessig er dette en god løsning, da man er sikker på at JWT-en forsvinner fra sessionStorage og applikasjonens minne, slik at ingen angripere kan få tilgang til den selv om den relaterte brukeren er logget ut. I tillegg blir statusen i Vuex store fjernet. I utgangspunktet ble ikke statusen nullstilt. Et problem vi da støtet på var at man kunne logge seg ut av en bruker, logge inn med en ny bruker, og likevel ha samme status på begge brukere. Hvis vi ikke hadde lagt til nullstilling, men likevel implementert bruk for kunder, kunne en kunde teknisk sett ha tilgang til en annen kundes status. Dette er ikke sikkert, da en kundes informasjon om relaterte sikkerhetsfeil er svært sensitiv. Vår løsning er derfor både et sikkert tiltak. Vi kom i mål

med utloggingsfunksjonaliteten, og var fornøyde med vår løsning for sletting av JWT og applikasjonens minne.

Mulighet for en innlogget administrator fra Atea til å godkjenne/forkaste/endre nyoppdagede sikkerhetsfeil fra systemet før det settes inn i systemet.

I utgangspunktet tenkte vi at sikkerhetsfeil med en lav CVSS-score automatisk skulle bli filtrert ut av systemet, siden de ofte ikke skaper noe stor sikkerhetstrussel. Etter samtale med Atea IRT ble vi enige om å hente inn alle sikkerhetsfeil uavhengig av CVSS-score, og heller gjøre det mulig å filtrere og sortere manuelt på CVSS-score. Grunnen til dette var at de tidligere hadde opplevd å finne sikkerhetsfeil som var klassifisert med lavt trusselnivå, som i virkeligheten påvirket langt flere systemer enn det som ble implisert av CVSS-scoren den ble gitt. Ved å velge en slik løsning er man sikker på at alt av sikkerhetsfeil blir registrert på nettsiden, uavhengig av hvor alvorlig utstederen har klassifisert sikkerhetsfeilen som. Vi har i tillegg gjort det mulig å gjøre endringer på registrerte sikkerhetsfeil, slik at man selv kan angi en mer passende CVSS-score, eller legge til kritisk informasjon i beskrivelsen av feilen.

Ikke-funksjonelle egenskaper

Universelt design iht. WCAG 2.1 standarden

I forhold til å følge WCAG 2.1 standarden har vi vurdert opp mot hvem som kommer til å bruke plattformen i hvor stor grad vi har valgt å implementere den. Den endelige utgaven av webapplikasjonen kommer kun til å bli brukt av interne brukere i Atea IRT, og de viktigste prinsippene å følge ble dermed prinsipp 1 (perceivable), prinsipp 2 (operable) og prinsipp 3 (understandable).

Behandling av personvernopplysninger

Når det gjelder behandling av personvernopplysninger og sensitiv data ble det gjort klart relativt tidlig i prosjektet at sikker lagring av slik data skulle være høyt prioritert, siden sikkerheten til kundene er viktig for Atea IRT. Det viktigste for deres del var dermed å sikre at kundedata blir lagret på en sikker måte. Basert på dette har vi vært nøye på hvordan personvernopplysninger og sensitiv data blir behandlet i applikasjonen, og har sørget for tilstrekkelig sikkerhet i de forskjellige delene av systemet (se punkt om behandling av personvernopplysninger i delkapittel 4.2.1).

Generell "web-service security" i systemet er minimum iht. OWASP sine retningslinjer for "Web Service Security"

Gjennom utviklingen av webapplikasjonen har vi brukt OWASP sine retningslinjer for sikring av webapplikasjoner for å sjekke den for svakheter. (*Web Service Security*, n.d.) Vi har ikke fulgt alle tips og råd, da vi har vurdert trusselbildet opp mot hvor mye ressurser vi skal bruke på å sikre applikasjonen mot forskjellige typer angrep. Mye av sikkerheten rundt webapplikasjonen kommer også ved bruk av rammeverk som Spring Boot og Vue, som har integrerte sikkerhetsrutiner. Det er derfor ikke like viktig å gå systematisk gjennom alle sikkerhetstiltak. Ved videre utvikling av applikasjonen kan det derimot være viktig å beskytte seg mot spesifikke typer angrep, og er noe som burde diskuteres sammen med Atea IRT for å sikre at den dekker opp under deres krav for sikkerhet.

Egenskaper som ikke er inkludert i visjonsdokumentet

Web-scraper

Helt fra starten av prosjektet visste vi allerede at innhenting av sikkerhetsfeil ville være et av de viktigste punktene i systemet vårt. Vi ville derfor finne en effektiv måte som kunne gjennomføre automatisk innhenting. Vi startet først med å utvikle en web-scraper som leste fra én og én kilde-feed. Etter kort tid fant vi ut at denne metoden ikke var effektiv nok, og vi begynte derfor å implementere tråder for å tråd-optimalisere web-scraperen. Dette endte opp med å bli en stor forbedring.

Som nevnt i punktet om web-scraperen i delkapittel 4.2.1 har vi også utviklet et rammeverk, der hensikten å forenkle implementeringen av nye innhentingskilder. Rammeverket er produsert for at man skal unngå å kode nye klasser for hver nye innhentingskilde. Dette effektiviserer implementeringen, og gjør at utviklere som ikke er kjent med systemet også kan implementere nye kilder uten å sette seg inn i alt av kode web-scraperen har.

Trådoptimaliserte backends

Hovedgrunnen til at vi implementerte tråd-optimaliserte funksjoner i begge backendene, er for å effektivisere arbeidet. Det som angår web-scraperen er beskrevet i begge punktene om web-scraping i delkapitlene 4.2.1 og 5.2.1. For system-backenden sin del gjelder tråd-optimaliseringen for funksjoner som filtrering og sortering av sikkerhetsfeil. Etter å ha skrevet kode for filtrering i backend og brukt disse funksjonene i frontend, fant vi ut at filtreringen tok for lang tid. Grunnen til dette var at filtrerings-metoden instansierte veldig store objekter med svært mange koblinger, og filtrerte gjennom denne informasjonen for hver eneste sikkerhetsfeil i databasen. Dette ble et problem med bare et par hundre sikkerhetsfeil, så vi visste at denne også kom til å skalere dårlig når antallet sikkerhetsfeil i databasen økte. Siden vi allerede hadde implementert tråd-optimalisering i web-scraperen, implementerte vi den samme metodikken i filtrerings-metoden for å korte ned tidsbruken betraktelig.

CVE og produsent

Diskusjon angående kategorisering ved hjelp av CVE er skrevet i punktet om kategorisering tidligere i dette delkapittelet. Når det gjelder CVE-feeden, valgte vi å gi brukere en oversikt over alle CVE-er som finnes i databasen, samt vise hvilke sikkerhetsfeil hver CVE har. Dette var også et ønske av Atea IRT. I samarbeid med Atea IRT fant vi ut at CVE-er ville fungere bra som kategoriserisering av sikkerhetsfeil, samtidig som man kunne se en oversikt over alle disse kategoriene.

Når det gjelder produsent var dette et punkt som også ble tatt opp under et av våre møter med oppdragsgiver og Atea IRT. I likhet med CVE, var det også et behov for å visualisere alle produsenter, sammen med sikkerhetsfeilene som relateres til hver produsent.

I denne versjonen kan man ikke se hvilke kunder og teknologier som knyttes til CVE eller produsent. Likevel, kan man teknisk sett finne alle relaterte teknologier og kunder ved å trykke seg inn på en sikkerhetsfeil som vises i listen inne i CVE- og produsent-popupen. Vi fant derfor ut at det ikke var et like stort behov for å visualisere kunder og teknologier for hver CVE og produsent dette i denne versjonen.

Dockerisering

For å øke sikkerheten og påliteligheten (se kapittel 4.1.3) til AIRS og samtidig simplifisere bygging og kjøring av systemet, i skyen, men også lokalt benytter som sagt begge backends av systemet Docker. Et dilemma som vi støtte på rundt akkurat dette er at Docker i seg selv kan være sårbar for sikkerhetshull. Dette kan for eksempel skje gjennom å benytte usikre Docker images som kanskje sender påloggingsinformasjon til en database eller annet til en potensiell angriper. For å unngå slike problemer kan man benytte seg av offisielle Docker images eller images som er utgitt fra pålitelige kilder som eksempel Google. Dette skal ikke være et problem i AIRS da Docker-filene bruker offisielle images som for eksempel Maven.

Frontenden av applikasjonen er ikke dockerisert, og dette er i all hovedsak grunnet at det er lite å tjene på det. Vue kompilerer koden ned til standardiserte JavaScript-filer som kan kjøre i enhver nettleser. Altså, har du en nettleser og Node.js, så kan du kjøre applikasjonen. Dette gjør at plattformuavhengighet ikke er nødvendig for frontend. I tillegg til dette så er prosessen å publisere en Vue-applikasjon som ikke er containerisert i skyen ekstremt enkel.

Database

Hibernate, som er det rammeverket vi brukte for å opprette databasen i backend, er et mye brukt rammeverk som gjør det svært enkelt å opprette en database basert på hvordan Java-objektene blir opprettet i backend. Hibernate bruker JPA for å opprette database-objekter, og tilfører også en måte å få tilgang til og håndtere disse i databasen. Siden dette rammeverket er så godt dokumentert og at medlemmene i gruppa hadde god erfaring med det tidligere, var dette rammeverket en klar vinner når vi skulle bestemme oss for hvordan vi ville opprette databasen.

Grunnen til at vi valgte å bruke Spring Data JPA Repositories for å kommunisere med databasen var fordi dette rammeverket er spesielt tilpasset i bruk sammen med Spring Boot, som er det rammeverket vi brukte for å lage system-delen av backend. Ved å bruke Spring Data JPA sparte vi også mye tid på å manuelt skrive database queries, og kunne bruke mer av tiden på å forsikre oss om at metodene fungerte slik de skulle.

Det som kanskje var mest utfordrende var å konfigurere klassene på riktig måte med Java Persistence API, blant annet ved på grunn av problemer med Lazy-initialisation i metodene vi lagde. Dette er kort forklart prosessen ved å forhindre oppretting av objekter, kalkuleringen av verdier eller andre kostbare prosesser - for å forhindre at systemet bruker flere ressurser enn nødvendig når det håndtere ulike objekter fra databasen. Mange av metodene våre var avhengige av å få tak i verdier fra hele objekter, eller child-objektene til objektene for å fungere slik de skulle. På internett var det mange som foreslo å endre initialiserings-metodikken til Eager, som ville løst problemene vi hadde men ført til langt mer ressursbruk enn vi kunne tillate på enkelte funksjoner. Etter hvert som vi fortsatte å lete fant vi en løsning som fungerte perfekt for vårt formål; NamedEntityGraphs. Dette er en teknikk som gjør at man kan opprette objekter med spesifiserte attributter, slik at man selv kan styre hvilken informasjon man vil ha med når man håndterer objektet. Vi utnyttet denne teknikken overalt hvor det var passende, og fikk gjort kommunikasjonen med databasen så effektiv og ressursbesparende som mulig.

Siden databasen som opprettes ved å bruke JPA er fleksibel med hvilken plattform den kan implementeres på, er det mulig for Atea IRT å selv velge hvilken plattform de vil håndtere databasen med, og gjør det enkelt å gjøre forandringer på både objekter og koblinger i databasen uten å gjøre forandringer inne på selve databasen.

Vuex store og lagring av status i hver feed

I et tidligere scrum-prosjekt hadde vi brukt Vuex store. Vi valgte derfor å implementere det i bachelorprosjektet, da vi visste hvordan vi kunne utnytte verktøyet. Man kan hente ut status fra store over hele frontend. Ved å for eksempel lagre rotadressen til backend-systemet i store, vil vi ha enkel tilgang til denne adressen også andre steder i programmet. Hvert tjenerkall bruker denne adressen for å nå endepunkt i backend. Hvis man skal endre på rotadressen trenger man derfor bare å endre adressen i store. Hadde rotadressen vært skrevet inn manuelt for hvert tjenerkall, ville man være nødt til endre absolutt alle mulige tjenerkall. Dette er derfor en god løsning hvis man vil lagre verdier som skal brukes av flere komponenter, uten at man må opprette et nytt objekt for dem hver gang de skal brukes.

Underveis i prosessen fant vi ut sammen med Atea IRT at det kunne være ønskelig å lagre status for alle feeder i store. Slik kan man bruke applikasjonen på en mer effektiv måte, da man for eksempel slipper å filtrere sikkerhetsfeil-feeden på nytt hvis man endrer feed. Hver feed har en egen nettadresse. Dersom feedene hadde vært under samme nettadresse, kun som egne individuelle komponenter, ville statuslagringen ha vært satt opp litt annerledes. Ved å gjøre det på denne måten ville vi kanskje ikke trenge å lagre status i Vuex store. Da kunne hver komponent ha lagret status som private variabler innad i komponenten. Vi fant likevel ut at vår løsning i utgangspunktet vil fungere godt.

SessionStorage og lagring av brukerattributter

I utgangspunktet begynte vi å bruke localStorage istedenfor sessionStorage, da vi hadde brukt localStorage i tidligere prosjekt. Etter å ha sett gjennom OWASPs “cheat sheet series”, fant vi ut at vi burde bruke sessionStorage istedenfor localStorage, da sessionStorage er mer sikkert rundt en enkelt nettleser (OWASP, n.d.).

Vi valgte å lagre JWT i sessionStorage, da dette ville hjelpe oss med autentisering ved hvert tjenerkall. Dersom JWT ikke fantes i sessionStorage ved et tjenerkall, ville man ikke ha mulighet til å gjøre kallet i det hele tatt. Dette påvirker sikkerheten til applikasjonen på en positiv måte. Det finnes kanskje andre måter for å lagre JWT og andre brukerattributter i nettleser og frontend, men sessionStorage var den løsningen vi synes svarte best til vårt behov.

Oppretting og endring av bruker

Oppretting av nye brukere kan kun gjøres av en bruker med administrator-privilegier. En av grunnene til dette er at webapplikasjonen i utgangspunktet kun skal brukes av ansatte i Atea IRT, og det er derfor ikke nødvendig at eksterne brukere skal kunne registrere en bruker i systemet. En annen grunn er at dette er det sikreste alternativet i forhold til adgangskontroll på applikasjonen. Ingen brukere uten autentisering skal ha mulighet til å bruke applikasjonen uansett.

Profilbilder

I starten av prosjektet hadde vi planlagt å gi kunder og brukere et profilbilde, slik at man kunne se logoer og lignende. Etter hvert som vi utviklet produktet fant vi også ut at det kunne være greit å vise logoer til produsenter og teknologier i tillegg. Profilbildene har en felles hovedoppgave, som er å gi en bedre visuell oversikt. I mange tilfeller vil man kanskje kjenne igjen produsenter, teknologier og kunder kun ved å se på bildet i feeden. Grunnen til at vi implementerte profilbilder var derfor å gjøre web-applikasjonen mer estetisk, samtidig som bildene bidrar til enkel oversikt.

Favoritter, auto-oppdatering og uleste sikkerhetsfeil

Underveis i utviklingsprosessen fant vi ut at det kunne være behov for markering av sikkerhetsfeil slik at man enkelt kunne finne dem igjen senere, eller få dem oppdatert med et hyppigere intervall enn vanlig. For å løse dette la vi til to ulike attributter på sikkerhetsfeilene som kunne brukes for å markere dem til disse formålene; favorite og auto-update. Ved å endre på disse verdiene (true eller false) kan man altså markere en sikkerhetsfeil som favoritt, eller få web-scraperen til å oppdatere informasjonen om denne sikkerhetsfeilen hver dag. Disse funksjonene mente Atea IRT kunne være nyttig hvis det var sikkerhetsfeil som behøver ekstra oppfølgning. I tillegg markeres alle uleste sikkerhetsfeil med et rødt merke, da man enkelt skal ha oversikt over hvilke sikkerhetsfeil som er åpnet og lest.

Testing av systemet

Grunnen til at vi valgte å ikke implementere testene på nytt etter hvert som vi gjorde endringer på systemet, eller lagde nye tester for ny funksjonalitet, var hovedsakelig på grunn av tidsmangel og dårlig prioritering av testing. For å veie opp for mangel av automatiske tester utførte vi derimot grundig manuell testing av alle metoder og funksjoner i systemet, både i Postman og ved å bruke systemet som normalt. Til tross for grundig manuell testing er dette derimot en langt mindre effektiv måte å sjekke at ting fungerer som det skal, og gjør det mye vanskeligere å oppdage feil underveis i utviklingen.

Den største utfordringen vi støtte på da vi først skulle implementere automatiske tester var at det var vanskelig å konstruere et reelt testmiljø. Grunnen til dette var at det var ekstremt mange koblinger mellom de ulike objektene i systemet, og alle disse måtte lages manuelt hvis man skulle teste dem slik vi hadde gjort tidligere. Dette problemet ble enda større da vi måtte gjøre endringer på koblingene underveis i utviklingen, som for eksempel ved implementasjon av CVE og produsent (se punkt om CVE i delkapittel 5.2.1).

For å unngå dette problemet fant vi senere ut at man automatisk kunne opprette testmiljø basert på informasjon fra en annen database. Vi kunne altså lagt inn data i vår egen MySQL-database, kopiert kommandoene som ble brukt for å opprette databasen, og lagt inn dette i den interne H2-databasen som ble brukt til testing. På denne måten hadde koblingene allerede vært etablert, og vi kunne testet på data som allerede hadde vært lagt inn.

Når vi utforsket hvordan man sørger for et pålitelig system ble det også tydeligere for vår del hvor viktig testing faktisk er for en slik applikasjon, og hvor stor effekt det kan ha under utviklingen av produktet (se punkt om Hvordan sikre effektiv og pålitelig drift

av systemet? i delkapittel 5.1.3). Hvis vi hadde sett mer på dette tidligere i prosjektet kunne det kanskje bidratt til at vi hadde prioritert det sterkere, eller gjort ting på en litt annen måte. Ved fremtidige implementasjoner av systemet burde det altså implementeres automatisk testing for å effektivt kunne oppdage feil og teste funksjonalitet under bruk og utvikling.

5.2.2 Status på systemet ved innlevering

Ved innlevering hadde systemet mye av den funksjonaliteten som både vi og oppdragsgiver ønsket, selv om enkelte deler som vi i utgangspunktet hadde satt til høyere prioritering ikke ble implementert. Et system for varsling av Atea IRTs kunder, samt intern varsling av nye endringer og diverse oppdateringer er funksjoner vi gjerne skulle likt å få implementert i systemet. Testing er også et punkt vi ville ha utfylt dersom vi hadde hatt bedre tid. Hovedgrunnen til at vi har prioritert som vi har gjort, er på grunn av tidspress. Samtidig som vi ville likt å fortsette på systemet, ville vi i tillegg skrive en god og utfyllende hovedrapport som ikke hadde mangler. Dersom vi hadde implementert flere ting i applikasjonen ville det ha påvirket hovedrapportens innhold på en negativ måte. Funksjonaliteter rundt sikkerhet og pålitelighet er stort sett solide, og svarer til våre forventninger. En ting vi garantert vil legge til dersom vi får i oppgave å videreutvikle produktet, er en form for multifaktorautentisering, da dette virker som en bedre autentiseringsmetode enn vanlig passordautentisering. Produktet vårt har derfor noen svakheter rundt manglende funksjonalitet, men de positive egenskapene veier opp for det meste.

I forhold til totale mål og resultater for systemet, kan vi si oss fornøyde med produktet. Vi fikk implementert det som var mest nødvendig, samtidig som vi i tillegg fikk lagt til funksjonalitet som effektiviserer bruken av systemet. Vi er spesielt fornøyde med tråd-optimaliseringen og hvilken effekt det ga produktet. Vi har gjort det vi kan for å effektivisere så mye som mulig, da vi vet at produktet i fremtiden skal ha mulighet til å håndtere flere tusen sikkerhetsfeil.

I vårt nest siste møte, hvor vi presenterte versjon 1.0 av produktet, stilte veilederen vår et konkret spørsmål til ett av Atea IRTs medlemmer, Geir Olav Skei. Spørsmålet lydde slik: Svarer sluttproduktet til Atea IRTs forventninger? Skei svarte ja, samtidig som han sa at han så et potensiale for mer funksjonalitet. Vi i bachelorgruppe 127 er også fornøyde med at systemet er godkjent av Atea IRT. I tillegg er det fullt mulig å videreutvikle produktet. I og med at vi fikk to av to mulige positive svar på spørsmålet om AIRS bør videreutvikles fra spørreundersøkelsen i punkt 4.1.4, kan vi også anta at Atea IRT ser et stort potensiale i produktet og vil at systemet skal videreutvikles.

5.3 Diskusjon rundt administrative resultater

5.3.1 Utviklingsmetode

Å følge en modifisert utgave av scrum-modellen var noe som fungerte svært godt for vår del, mye på grunn av at vi jobbet i en liten gruppe. Ved større prosjekter har vi tidligere sett effekten av å implementere hele scrum-modellen gjennom utviklingen, men ved å håndplukke de metodene som vi følte kom til å gi størst effekt, fikk vi laget en utviklingsmetode som beholdt de viktigste elementene ved den smidige utviklings-metodikken, samtidig som vi kunne være mer fleksible enn hvis vi skulle fulgt hele

modellen. Ved å dele inn prosessen i ulike sprinter, sette opp issue-board for å kartlegge prosessen og holde jevnlig kontakt med både veileder og oppdragsgiver for å få tilbakemeldinger på progresjonen i arbeidet, fikk vi opparbeidet en god rutine helt fra starten av prosjektet.

Noe fra den opprinnelige scrum-metodikken som kunne gitt god effekt på dette prosjektet er daglig scrum, hvor man kartlegger progresjonen mot neste sprint og planlegger fremtidig arbeid. Dette kunne gitt oss en mer oversiktlig arbeidshverdag, samtidig som det også kunne bidratt til en mer effektiv utførelse av enkelte arbeidsoppgaver. Grunnen til at vi valgte å ikke legge så stort fokus på akkurat dette er fordi vi hadde tett kommunikasjon gjennom hele prosjektet, og møtte hverandre fysisk eller snakket sammen i Discord så i si hver eneste dag. Dette førte også til at vi hele tiden var oppdatert på hvem som arbeidet med hva, og hvordan vi lå an på de forskjellige punktene. Selv om dette ikke er en like systematisk måte å jobbe på, fungerte det svært godt i en liten gruppe. En annen grunn til at dette fungerte godt for akkurat vår gruppe er fordi alle medlemmene i gruppa kjenner hverandre fra før, og tilbringer tid sammen også på fritiden. Det var derfor aldri noe fare for at kommunikasjonen skulle svikte på grunn av uenigheter eller at noen i gruppa ikke turte å ta opp viktige temaer.

5.3.2 Prosjektmøter, kommunikasjon og samarbeid med veileder og oppdragsgiver

Ved å holde møter med oppdragsgiver hver tredje uke eller for hver sprint har vi også fulgt en av de viktigste punktene ved den smidige utviklingsmetodikken, som er å justere produktet basert på tilbakemeldinger fra interessentene for produktet. Dette førte til at vi fikk gode tilbakemeldinger på alt av arbeid vi utførte, samtidig som det førte til store endringer for enkelte deler av systemet etter hvert som vi utformet produktet. Dette var noe som for vår del var både nytt og lærerikt, og førte til at vi fikk oppfylt både våre egne og oppdragsgivers ønsker. Vi lærte å være mer fleksible rundt utviklingen, og å holde jevn og god kontakt med alle som var inkludert i prosjektet.

Ved at vi i tillegg fikk tilegnet oss en kontorplass hos Atea i Trondheim følte vi at vi ble mer inkludert i prosessen, og fikk en slags tilhørighet i selskapet som gjorde motivasjonen for å skape et godt produkt større. Her fikk vi møtt både ansatte fra Atea og kom i kontakt med andre personer som interesserte seg for hva vi holdt på med, noe som gjorde det litt mer spennende å utnytte seg av kontorplassen vi ble tildelt. I tillegg til at det ble mer sosialt å jobbe sammen, hadde vi også tilgang til mer utstyr, flere ressurser og andre goder som gratis lunsj og parkering i byen. Alt dette bidro til at vi fikk en svært positiv erfaring ved å jobbe sammen med Atea.
Vi følte også at veileder under prosjektet hadde svært god oppfølging, og kom med gode tips og råd både når det gjaldt kommunikasjon med oppdragsgiver og progresjon underveis i prosjektet. Han var også tydelig på forbedringspunkter og ting som kunne bli gjort annerledes, som førte til at vi gjennomgikk ting litt nøyere enn vi ville gjort med en mer avslappet veileder. Dette var både motiverende og nyttig for vår egen progresjon, da det til tider var usikkerhet om både deler av hovedrapporten og hvor mye vekt vi skulle legge på forskjellige deler av systemet. Alt i alt føler vi at vi har hatt tilstrekkelig kommunikasjon med alle parter, og at det fungerte svært godt for både oss selv, oppdragsgiver og veileder.

5.3.3 Milepæler

Alle milepæler per 20. mai er nådd. Vi har prøvd å følge alle datoer der det er mulig. Ved å se på punktet om milepæler i delkapittel 4.3.3 kan man se at det er tre milepæler som har endret dato. Alle tre baserer seg på møter med enten veileder eller oppdragsgiver. Derfor er hovedgrunnen til at datoene har endret seg at det var visse møtedatoer som passet bedre. Siden milepælene ble satt i forprosjektsplanen, som ble ferdigstilt og levert den 28. januar, kan man forstå at det er vanskelig å fastbestemme datoer for møter langt frem i tid. Til tross for disse datoendringene har vi nådd alle milepæler.

5.3.4 Tidsplan og tidsbruk

Tidsplanen vi utformet for prosjektet var i form av et Gantt-diagram, som ga oss god oversikt over hvilke deler av prosjektet vi skulle jobbe med til enhver tid, og hvor lang tid vi skulle bruke på hver enkelt del. Å følge denne planen har vært svært nyttig, og har gitt oss en god pekepinne på hvor raskt vi måtte jobbe på hver del for å komme i mål med alt som skulle gjøres. Vi har måttet endre på deler av den underveis på grunn av forandring i planene, men har stort sett klart å følge den slik den først ble utformet.

Når det gjelder timebruk har vi ikke klart å komme i mål med 500 timer, som er det antallet vi først etablerte i arbeidskontrakten i starten av prosjektet. Grunnen til at vi ikke klarte å oppnå 500 timer er fordi tiden rett og slett ikke strakk til. Vi føler selv at vi har arbeidet tilstrekkelig med prosjektet, og har brukt tiden effektivt. Vi har også forhørt oss med veileder angående dette punktet, og ble forsikret om at antallet timer ikke var viktig så lenge produktet og hovedrapporten var utført tilstrekkelig.

En annen grunn til at antallet timer ble mindre enn det vi først hadde sett for oss var at vi hadde et emne i starten av bacheloroppgaven som kjørte parallelt med oppstarten av prosjektet, hvor vi hadde både arbeidskrav og eksamen som skulle utføres innen de tre første månedene av semesteret. På grunn av dette valgte vi å fokusere på å få dette emnet godkjent før vi begynte å fokusere hundre prosent på bacheloroppgaven. Vi fikk gjort unna en del av dokumentasjonen i starten, og begynte på utviklingen av produktet rundt 15. februar. Vi skulle gjerne ønsket å ha enda flere uker på å utføre bachelor-oppgaven, da vi følte at det var litt stressende i starten. Vi var redde for at vi ikke kom til å jobbe nok på bachelorprosjektet, samtidig som vi jobbet med INGT2300. Likevel klarte alle gruppemedlemmene å finne en god balanse. I og med at vi var i samme gruppe, både i bachelorprosjektet og gruppeprosjektet i INGT2300, kunne vi enkelt kommunisere med hverandre. Vi hadde derfor planlagt hva vi skulle gjøre hver uke, for å få utnyttet tiden vår optimalt. Alle i gruppen er enige om at det er en dårlig idé å kombinere to så store fag samtidig, i tillegg til at INGT2300 kunne bli veldig intenst til tider, på grunn av at emneslutt var i mars og ikke mai. Det kunne derfor vært fint å vurdere å ha INGT2300 i høstsemesteret før bacheloroppgaven, da det også inneholder relevante deler om entreprenørskap og ingeniørfag som kan være like relevant før oppstart som underveis i prosessen. Uansett så har gruppen klart å balansere arbeidet, og vi er derfor fornøyde med vår tidsbruk.

5.3.5 Refleksjon rundt gruppearbeidet

Som en liten gruppe på bare tre stykker, som også kjenner hverandre fra før og tilbringer tid sammen på fritiden, har vi kommunisert svært godt gjennom hele prosjektet. Vi har ikke hatt noen problemer med dårlig kommunikasjon, sykdom eller

konflikter innad i gruppa, og har klart å spille hverandre gode på de områdene vi kan best. Noe som man kanskje legger merke til hvis man ser på timelisten (se vedlegg C) er at Odin har noe mindre timer enn både Scott og Endré, som har relativt likt antall timer. Grunnen til dette er at Odin er småbarnsfar og har hatt en gravid kone gjennom utføringen av hele oppgaven. Mot slutten av oppgaven ble hans andre barn født, noe som også bidro til at han ikke fikk bidratt like mye i denne perioden.

Odin har til tider måtte dra tidligere fra arbeidet for å hente barn i barnehage, eller holde seg hjemme på grunn av sykdom hos barnet. Dette har også vært noe som vi har kommunisert om gjennom oppgaven, og gruppa har vært svært forståelsesfull for situasjonen. For å ta igjen for tapt arbeidstid har Odin også brukt mer tid med oppgaven på kveldstid, da hverdagen hans ikke har vært like fleksibel som Scott og Endré sin. Til tross for dette har han jobbet jevnt og trutt, og fulgt kjernetiden mellom 9-15 når det har vært mulighet for det.

Ser man bort ifra de små forskjellene rundt totalt antall timer, kan man se at hvert medlem har påvirket prosjektet i like stor grad. Blant annet har alle bidratt like mye, i form av arbeidsoppgaver, skriving og utvikling. Vi har hatt tydelige inndelinger av oppgaver i utviklingsprosessen, samtidig som vi har jobbet godt sammen om oppgaver. Hvis noen har lurt på ett eller annet, så har de to andre gruppemedlemmene vært hjelpsomme og støttende. På denne måten har vi jobbet veldig bra sammen som gruppe, og gruppearbeidet har fungert meget bra. Det eneste vi ser tilbake på som noe vi burde gjøre på en annen måte i fremtiden er fordelingen av hvem som programmerer hva i stacken. Vi hadde litt for adskilte ansvarsområder, hvor Scott som regel jobbet med frontend, Odin med database/backend og Endré med web-scraper/backend. Det endret seg litt mot slutten da vi måtte flette sammen alle delene, hvor alle jobbet med frontend for å gjøre ferdig og implementere alle funksjonalitetene fra backend. Denne måten å fordele arbeidsoppgavene på har sine fordeler og ulemper, men i fremtiden ønsker vi å variere mer i disse oppgavene slik at vi ikke blir totalt avhengig av en enkelt person for å ha fremdrift på en del av stacken.

Kapittel 6: Konklusjon og videre arbeid

6.1 Problemstilling

Som definert i kapittel 1.5 var problemstillingen vår i løpet prosjektet:
Hvordan sikre pålitelig, oppdatert og automatisk innhenting av sikkerhetsdata, samt sørge for sikker lagring av kundedata og autentisering av systemets brukere.

For å svare på denne problemstillingen har vi formulert 3 ulike forskningsspørsmål som skal bidra til å svare på disse spørsmålene, samt et ekstra forskningsspørsmål som tar for seg hvorvidt vi har klart å løse oppgaven vi fikk fra oppdragsgiver på en tilfredsstillende måte. Vi har brukt ulike former for kvalitative forskningsmetoder for å svare på forskningsspørsmålene, og presentert resultater som vi føler gir tilstrekkelig svar på problemstillingen vi fastslo i starten av prosjektet.

Den første delen av problemstillingen blir besvart ved at vi først presenterer ulike måter å sikre at driften av systemet er effektiv og pålitelig. Her viser vi blant annet til viktigheten av å utnytte microservices og containerisering for å øke påliteligheten til systemet, samt bruk av tråd-optimalisering for å effektivisere selve innhentingen av sikkerhetsdata.

Sikker lagring av kundedata kan utføres på mange måter, men vi valgte å hovedsakelig følge råd fra Datatilsynet og eksperter på området for å finne ut hvordan man tilstrekkelig lagrer kundedata. Her presenterer vi ulike krypterings- og hashing-algoritmer, samt hvordan man sørger for at disse blir implementert så sikkert som mulig.

Når det gjelder autentisering av brukere, finnes det flere forskjellige måter man kan gjøre dette på. Ut fra forskningen vi gjorde, fant vi ut at passord-autentisering er en usikker metode, mens multifaktor- og biometrisk-autentisering er mye sikrere metoder. For å sikre autentiseringen ytterligere, kan man i tillegg bruke sertifikatbasert autentisering av en datamaskin eller enhet. Vi fant også ut at autentisering ved bruk av tokens er en type autentisering som er sikker, dersom man implementerer og bruker den på rett måte.

Alt i alt har vi besvart problemstillingen vår. Vi har sett på måter man kan gjøre et system pålitelig og sikkert, og hvordan man sørger for å lagre sensitiv kundedata og passord på en sikker måte. I tillegg har vi forsket på ulike metoder for brukerautentisering.

6.2 Produkt

I dette prosjektet har vi hatt som mål å utvikle et system som svarer til oppdragsgivers krav og behov. Under *hensikt* i oppgaveteksten står det at: "*Hensikten med systemet er å lage et system som gjør det enklere for Atea IRT å få en oversikt over nye sikkerhetsfeil som angår kundene deres. Systemet skal også kunne brukes til å sende ut varslinger om alvorlige sikkerhetsfeil*."

For å se på graden av måloppnåelse produktet vi produserte i dette prosjektet har opp i mot prosjektets hensikt, ser vi på hva vi fikk implementert av både funksjonelle og

ikke-funksjonelle egenskaper. I tillegg til det så vil vi benytte oss av resultatene rundt forskningsspørsmål nr 4. (se kapittel 4.1.4), og ut ifra den informasjonen trekke en konklusjon.

Av funksjonelle egenskaper som ble skissert i vedlegg F, har vi klart å implementere de aller fleste i versjon 1.0 av AIRS, bortsett fra varsling, testing, grundig logging, statistikk og noe sortering/filtrering på enkelte sider. De funksjonelle egenskapene som er implementert fungerer slik de var ment til å fungere, og har blitt gjennomgått for feil og mangler ved både manuell testing og bruk av nettsiden under utviklingen.

Av de ikke funksjonelle egenskapene har vi også overholdt mange av disse under utviklingen av produktet, men fant etter hvert ut at enkelte punkter burde bli prioritert lavere enn vi i utgangspunktet hadde tenkt. Kommunikasjon over HSTS og å overholde alle retningslinjer i WCAG 2.1 er blant de som fikk lavere prioritet, mye på grunn av at de ikke virket nødvendig å implementere i sin helhet.

Som tidligere diskutert bidrar forskningsspørsmål 4 til å finne ut hvilke effekter Atea IRT får i arbeidsdagen sin ved å benytte AIRS. Til tross for et lavt antall svar på spørreundersøkelsen (vedlegg J), og at vi måtte konvertere fra kvantitativ til kvalitativ forskningsmetodikk som en reaksjon på det, konkluderer vi med at kvaliteten på svarene var høy. Dette grunnet at vi fikk tilbakemelding på at oppgavestiller var en av de som testet demoen vi distribuerte og ga tilbakemelding på det. Videre, når vi ser på resultatene av selve spørreundersøkelsen (vedlegg K) så kan vi konkludere med at Atea IRT vil ha god nytte av AIRS. Det kan vi si siden Atea IRT svarte at mesteparten av funksjonene i AIRS øker graden av automatisering eller systematisering i de daglige prosessene sine, sammenliknet med før. Dette utenom funksjonene for oversikt over hvilke teknologier kundene deres har, og oversikt over hvilke kunder som blir påvirket av hvilken CVE. Dette er funksjonalitet som uansett ikke er hovedfunksjonaliteten til applikasjonen.

Dermed, for å oppsummere rundt produktet sin oppfyllelse av oppgavestiller sine behov som definert i oppgaveteksten, ser vi ved hjelp av resultatene til forskningsspørsmål 4 at AIRS sannsynligvis øker produktiviteten til IRT i de fleste daglige prosessene deres. Med noen små unntak. Videre når vi ser på hvilke funksjonelle og ikke-funksjonelle egenskaper som ble med i versjon 1.0 av systemet, så kommer vi frem til at det som gjenstår er kun egenskaper som står definert med middel til lav prioritet. Det eneste unntaket er automatisk varsel ved sikkerhetsfeil for Ateas kunder og HSTS. Vi må også ha i bakhodet at versjon 1.0 av systemet ikke har implementert en særlig stor grad av innhentingskilder for web-scraperen. Versjonen implementerer nok kilder for å bevise at rammeverket for tillegging av feeder fungerer. Alt i alt *konkluderer vi med at produktet i stor grad oppfyller oppdragsgiver og Atea IRT sine behov.*

6.3 Videre arbeid

Til tross for at prosjektet er avsluttet har vi vært i samtale med Atea IRT om å utvide applikasjonen etter endt bacheloroppgave, og de har vist interesse for at vi skal videreutvikle applikasjonen for å få inn mer funksjonalitet. Noe som er viktig å påpeke er at versjon 1.0 av AIRS kun implementerer tre innhentings-kilder for sikkerhetsfeil, mens Atea IRT sin daglige drift overvåker over 20 forskjellige nettsider. Det er derfor nødvendig å implementere disse i systemet hvis det skal erstatte dagens manuelle

system i sin helhet. Siden interessen har vært stor for å videreutvikle produktet, er dette noe av det første vi ville fokusert på hvis vi fikk tilbudet om å jobbe videre på det etter endt bacheloroppgave. Dette er også en av grunnene til av vi brukte god tid på å gjøre prosessen ved å legge til innhentings-kilder så enkel som mulig.

Under ser vi en tabell med mulige videreutviklinger og prioriteringsgrad ifølge resultatene vi fikk i forskningsspørsmål fire:

Mulighet for å visualisere/eksportere statistikk.	4
Varslingssystem som varsler kunder.	2.5
Notifikasjons-system innad i AIRS.	2
Tilgang for Ateas kunder til systemet.	1.5

Tabell 6.3.1: Resultater for spørsmålet om mulige videreutviklinger fra forskningsspørsmål fire.

Varsling av kunder ved oppdaging av sikkerhetsfeil

Et viktig punkt fra visjonsdokumentet, som vi rett og slett ikke rakk å implementere, er varsling av kunder. Ved videreutvikling av dette produktet, er dette et av de første funksjonalitetene som vil bli lagt til. Per nå er det mulig å se hvilke kunder som blir påvirket av ulike sikkerhetsfeil. Likevel er det et behov for å lage et eget varslingssystem i produktet vårt. Planen er å legge til en knapp med funksjonalitet i sikkerhetsfeil-popupen, som en bruker kan trykke på for å varsle kunde(r) angående den gjeldende sikkerhetsfeilen. Deretter vil et varsel sendes fra backend, til kundens e-postadresse. Dette forutsetter også at vi legger til en e-postattributt for hver kunde.

Statistikk

Statistikk er en funksjonalitet vi tenkte kunne være grei å implementere. Dette kan være statistikk over hvor mange sikkerhetsfeil som har forekommet for alle teknologier og kunder over et bestemt tidsintervall. Vi rakk ikke å implementere dette, men ved videreutvikling er dette et punkt som vil legges til.

Mulighet for å visualisere/eksportere statistikk

En funksjonalitet som var sterkt ønsket av Atea IRT ifølge resultatene for forskningsspørsmål fire, var mulighet for visualisering og eksportering av statistikk. Dette kan være eksportering av statistikk (som beskrevet i punktet over om statistikk) eller visualisering av forskjellig type statistikk for hver kunde. Gjennom varsling av kunder, kan kundene også ha mulighet til å se statistikk og en overordnet oversikt over en sikkerhetsfeil. En CSV-eksport av statistikk kunne være interessant ifølge Atea IRT.

Notifikasjonssystem innad i AIRS

Et annet punkt som ble diskutert i forskningsspørsmål fire var implementasjonen av et notifikasjonssystem innad i AIRS. Dette kan være notifikasjoner som blir sendt ut dersom en sikkerhetsfeil blir oppdatert, en bruker har gjort endringer på et element, eller at en kommentar er lagt til. Brukere av AIRS vil da ha muligheten til å se slike notifikasjoner i navigasjonsbaren. Punktet har ikke like høy prioritet, men er uansett en fin funksjonalitet å ha innad i systemet, da det bidrar til at en bruker enkelt kan finne ut hvilke endringer som er gjort.

Tilgang for Ateas kunder til systemet

Kundetilgang er også et punkt som ble tatt opp i forskningsspørsmål fire. Som vi kan se har dette punktet lav prioritet. Likevel, ved videreutvikling, skal det ikke være veldig avansert eller komplisert å implementere denne funksjonaliteten. Ved å gi kunder tilgang til systemet, må man passe på at det er kun relaterte sikkerhetsfeil, CVE-er, produsenter og teknologier som vises for den kunden. Hvis ikke vil alle kunder ha tilgang til alt i systemet, som betyr at hvem som helst kan se hvilke sikkerhetsfeil en spesifikk kunde har. Her kreves det grundig autentisering og bruk av samme e-postadresse for både kunde og bruker.

Filterfunksjonalitet for andre feeder

En funksjonalitet vi kun implementerte i sikkerhetsfeil-feeden er filtrering. Vi skulle gjerne likt å ha implementert dette i de andre feedene i tillegg, men så ikke behovet for det i første versjon av produktet, samtidig som vi prioriterte andre funksjoner istedenfor. Filtrering for eksempel på tagger kan være brukbart i alle feeder.

Fikse sortering

I punktet om sortering, filtrering og søk i delkapittel 4.2.1 og 5.2.1, nevner vi at sortering kun fungerer i sikkerhetsfeil. Funksjonaliteten for sortering ligger også i de andre feedene, men det visuelle fungerer ikke slik vi hadde tenkt. Dette er et punkt som skal være enkelt å fikse, dersom produktet videreutvikles. Det skal også være mulig å legge til flere sorteringsmetoder.

Mulighet for valg av varslingsmetode

Hvis vi først hadde implementert et varslingssystem for kunder av Atea eller brukere internt fra Atea IRT, hadde det også vært logisk å kunne velge hvilken form for varsling man ville motta. Her kunne vi gjort det mulig å få varsling på både mail, SMS og internt i systemet. Hvis vi tenker oss et system hvor kunder ikke har tilgang til plattformen, kunne vi også gitt valget til brukerne av systemet om å sette en varslingsmetode for kundene.

Grundig logging av viktige deler av plattformen

Vi rakk ikke å implementere grundig logging i systemet. Vi begynte med bruk av Logger i Java, men fant ut at vi ikke rakk å bruke det i hele systemet. Dette blir en oppgave for videre utvikling.

Sikker kommunikasjon over internett ved hjelp av HSTS-protokollen

Mot slutten av produktet fikk vi kjørt alle deler av systemet vårt i en skytjeneste. Disse skytjenestene brukte automatisk HTTPS-protokollen for sikker forbindelse mellom delene. På grunn av sikkerhetsmessige grunner måtte vi stoppe kjøringen offentlig, som gjør at versjonen teknisk sett kun kjører lokalt per dags dato. Et ønske vi hadde for å sikre systemet ytterligere, var å implementere HSTS-protokollen. På grunn av tidspress måtte vi prioritere andre funksjonaliteter, men HSTS-protokollen kan bli vurdert i en videreutvikling av produktet.

Flere tråd-optimaliseringer

Tråd-optimalisering av funksjoner og metoder i systemet hadde svært god effekt der det ble implementert, og har potensiale til å effektivisere store deler av systemet. Søk,

innhenting av store lister med elementer, fremvisning av mye informasjon er bare et par eksempler på områder hvor det kunne blitt implementert, og er noe vi burde vurdere ved senere utvikling av applikasjonen.

Videreutvikling av feeder

Det er fullt mulig å videreutvikle alle feedene, i form av funksjonalitet og estetikk. CVE- og produsent-feeden kan videreutvikles for å vise teknologier og kunder som er knyttet til hver CVE og produsent. Videre kan det være mulig å vise relaterte sikkerhetsfeil i kunde-feeden og -popupen.

Koble CVE opp mot CVE.org

En fremtidig implementasjon kan være å koble opp hver CVE i databasen til CVE-er fra CVE.org. På denne måten kan hente inn mer informasjon om hver CVE, for eksempel en beskrivelse av hvilke typer sikkerhetsfeil som er knyttet til den.

Kapittel 7: Samfunnspåvirkning

Prosjektet vårt har i utgangspunktet basert seg på å bli tatt i bruk av en liten gruppe mennesker. Likevel er det mange kunder som påvirkes indirekte av systemet vi har laget, da det til syvende og sist er kundene til Atea som skal varsles om funnene som er blitt gjort av systemet vårt. På grunn av dette vil systemet i et større perspektiv, påvirke en stor kundegruppe. Den vil også kunne ha en positiv innvirkning på Atea IRTs arbeidshverdag hvis de velger å videreutvikle produktet etter endt bacheloroppgave. For å være godt egnet for både Atea IRT og deres kunder er det derfor lagt stor vekt på at sensitiv informasjon ikke skal komme på avveie, og at systemet er effektivt og pålitelig under bruk. Dette inkluderer både at driften ikke svekkes, at autentisering blir gjennomført for ulike funksjoner i systemet og at sensitiv informasjon blir lagret på en tilstrekkelig sikker måte.

En måte vårt arbeid kan ha en samfunnspåvirkning på, er ved at flere undersøker de mulighetene og løsningene som vi har presentert i oppgaven. Her er spesielt sikker lagring av kundedata og pålitelig drift av systemet noe som flere kunne hatt god effekt av å implementere i sine egne systemer, da mange ignorerer dette for å spare tid og ressurser. Ved å presentere ulike løsninger for å enkelt integrere slik sikkerhet i systemet, kan det hende at flere undersøker muligheten for å implementere lignende løsninger (se punkt om Vitenskapelige resultater i delkapittel 4.1).

Et viktig punkt i oppgaven var derfor å komme med oppdatert informasjon om hva som blir anbefalt for å sikre plattformen på ulike måter, og å selv implementere disse i vårt eget system for å vise at det fungerer.

En etisk problemstilling vi støtte på under utviklingen av applikasjonen var om vi skulle gjøre det enkelt for Atea å koble seg opp på ulike deler av systemet, lese informasjon i databasen og håndtere informasjon på plattformen, eller å forsikre oss om at kundeinformasjon, sensitiv informasjon og driften av systemet var sikkert. Her måtte vi velge metoder som både gjør det mer tidkrevende å bruke plattformen, og mer komplisert for interne brukere av systemet å finne informasjon på grunn av kryptering og hashing av brukerdata og passord. Grunnen til at problemstillingen i bunn og grunn løste seg selv, var at også Atea IRT hadde et ønske om å gjøre plattformen så sikker som mulig, noe som også innebærer å implementere sikkerhetstiltak som gjør bruken mindre effektiv.

Dette er en problemstilling som sikkert mange utviklere støter på underveis i utviklingen, og vi håper å sette et godt eksempel ved å både grundig undersøke muligheten for- og å faktisk ta i bruk - slike sikkerhetstiltak.

En konkret måte vårt arbeid kan ha effekt på andre som ser hvordan vi har valgt å sikre informasjon på plattformen vår på, er ved at vi har gått bort i fra Datatilsynets anbefalinger og velger å implementere en nyere og mer sikker hashing-algoritme for sikring av passord; bcrypt. Siden Datatilsynet er en stor og pålitelig aktør kan det hende at andre undersøker nye muligheter for å sikre passord på plattformen sin, da de ser at det finnes andre alternativer som er minst like gode som de som har blitt anbefalt tidligere. Ved å gjøre et dypdykk inn i dette temaet har vi også fått kunnskap om prosesser som gjør oppbevaring av sensitiv informasjon sikrere, som hvor man burde lagre krypteringsnøkler eller passord for ulike deler av systemet. Her har vi også tatt opp

en del punkter som det er tydelig at mange IT-bedrifter ignorerer, nemlig klartekst-lagring av passord direkte inne på systemet eller usikret lagring av sensitiv data. Ved å liste opp metodikker for å sikre slik informasjon, og komme med statistikk på hvor mange som ignorerer å gjøre det - kan det hende at flere bedrifter blir overbevist om å sikre systemene sine på en mer gjennomtenkt måte.

Selv om AIRS 1.0 altså ikke har noen direkte samfunnspåvirkninger, kan det få ringvirkninger for hvordan andre velger å gå frem for å sikre plattformen sin, eller hvilke kilder de velger å hente informasjon fra når de skal velge en måte å gjøre det på. Internett er stort, og det finnes mange som sitter på ekstremt mye og god kunnskap som kan utnyttes av andre. Vi valgte å ta i bruk den informasjonen vi fant og kombinerte den med allerede godt etablert kunnskap, og føler selv at vi har kommet med gode løsninger og argumenter for hvorfor vi har løst oppgaven på måten vi har gjort - og vi håper flere vil gjøre det samme.

Referanser

Alliance Software. (n.d.). *What are Software Development Methodologies?* Alliance Software. Retrieved April 5, 2022, from https://www.alliancesoftware.com.au/introduction-software-development-methodologies/

Apache Maven. (n.d.). *Maven – Introduction*. Apache Maven. Retrieved May 3, 2022, from https://maven.apache.org/what-is-maven.html

Atea. (n.d.). *Om Atea*. Atea. Retrieved April 7, 2022, from https://www.atea.no/om-atea/

Auth0. (n.d.). *Token Based Authentication Made Easy*. Auth0. Retrieved May 9, 2022, from https://auth0.com/learn/token-based-authentication-made-easy/

bcrypt. (2022, May 10). Wikipedia. Retrieved May 11, 2022, from https://en.wikipedia.org/wiki/Bcrypt

Bellairs, R. (2019, July 11). *What Is Code Quality? Overview + How to Improve Code Quality*. Perforce Software. Retrieved May 10, 2022, from https://www.perforce.com/blog/sca/what-code-quality-overview-how-improve-code-quality

Biryukov, A., Dinu, D., & Khovratovich, D. (2016, 05 12). *Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications*. IEEE Xplore. Retrieved 05 09, 2022, from https://ieeexplore.ieee.org/document/7467361

Brett, M. (n.d.). *File:Generic Feed-icon.svg*. Wikimedia Commons. Retrieved May 18, 2022, from https://commons.wikimedia.org/wiki/File:Generic_Feed-icon.svg

Brush, K. (n.d.). *What is a RDBMS (Relational Database Management System)?* TechTarget. Retrieved May 5, 2022, from https://www.techtarget.com/searchdatamanagement/definition/RDBMS-relational-database-management-system

CISQ. (n.d.). *Software Quality Standards – ISO 5055*. CISQ. Retrieved May 10, 2022, from https://www.it-cisq.org/standards/code-quality-standards/

Constantin, L. (2021, January 13). *What is hashing: How this cryptographic process protects passwords*. CSO Online. Retrieved April 5, 2022, from https://www.csoonline.com/article/3602698/hashing-explained-why-its-your-best-bet-to-protect-stored-passwords.html

CSID. (n.d.). *CONSUMER SURVEY: PASSWORD HABITS*. Experian Partner Solutions. Retrieved May 5, 2022, from https://www.csid.com/wp-content/uploads/2012/09/CS_PasswordSurvey_FullReport_FINAL.pdf

Datatilsynet. (n.d.). *Kryptering*. Datatilsynet. Retrieved May 10, 2022, from https://www.datatilsynet.no/rettigheter-og-plikter/virksomhetenes-plikter/informasjonssikkerhet-internkontroll/kryptering/

Educative. (n.d.). *What is hashing?* Educative.io. Retrieved April 5, 2022, from https://www.educative.io/edpresso/what-is-hashing

5 Common Encryption Algorithms and the Unbreakables of the Future. (2021, June 7). StorageCraft Technology, LLC. Retrieved May 10, 2022, from https://blog.storagecraft.com/5-common-encryption-algorithms/

Fol, P. (2021, October 26). *Spring vs. the World: Comparing Spring Boot Alternatives*. JRebel. Retrieved April 5, 2022, from https://www.jrebel.com/blog/spring-boot-alternatives

Franklin, R. (2021, March 13). *AES vs. RSA Encryption: What Are the Differences?* Precisely. Retrieved May 10, 2022, from https://www.precisely.com/blog/data-security/aes-vs-rsa-encryption-differences

GitKraken. (n.d.). *GitKraken Client Features | Git GUI & CLI for Windows, Mac, & Linux*. GitKraken. Retrieved April 7, 2022, from https://www.gitkraken.com/git-client/features

Gocoding. (2019, June 26). *Hva er Restful Web Services*. Go Coding. Retrieved April 5, 2022, from https://gocoding.org/no/what-are-restful-web-services/

Go Coding. (2019, June 26). *Hva er Restful Web Services*. Go Coding. Retrieved April 5, 2022, from https://gocoding.org/no/what-are-restful-web-services/

Goldman, J. (2016, September 22). *40 Percent of Organizations Store Admin Passwords in Word Documents*. eSecurity Planet. Retrieved May 3, 2022, from https://www.esecurityplanet.com/networks/organizations-store-admin-passwords-in-word-documents/

Google. (n.d.). *Digital signatures | Cloud KMS Documentation*. Google Cloud. Retrieved March 25, 2022, from https://cloud.google.com/kms/docs/digital-signatures

Grønmo, S. (2021, May 10). *Forskningsmetode - samfunnsvitenskap – Store norske leksikon*. Store norske leksikon. Retrieved April 5, 2022, from https://snl.no/forskningsmetode_-_samfunnsvitenskap

The Hacker Recipes. (2021). *Insecure JSON Web Tokens*. The Hacker Recipes. Retrieved May 14, 2022, from https://www.thehacker.recipes/web/inputs/insecure-json-web-tokens

Hadzalic, E., Kvarving, O., & Sonen, S. R. (2022). *Vedlegg F Visjonsdokument 127* (1.3) [Visjonsdokument som hører til AIRS] [Elektronisk dokument].

Hadzalic, E., Kvarving, O., & Sonen, S. R. (2022). *Vedlegg J Spørreundersøkelse-127* [Spørreundersøkelse laget av bachelorgruppe 127 for Atea IRT] [Elektronisk dokument].

Hadzalic, E., Sonen, S. R., & Kvarving, O. (2022). *Vedlegg C Timeliste-127* [Timeliste for bachelorgruppe 127] [Elektronisk dokument].

Hadzalic, E., Sonen, S. R., & Kvarving, O. (2022). *Vedlegg K Svar spørreundersøkelse-127* [Svar for spørreundersøkelsen utført av bachelorgruppe 127 på Atea IRT] [Elektronisk dokument].

IBM. (2021, June 23). *Containerization Explained*. IBM. Retrieved March 25, 2022, from https://www.ibm.com/cloud/learn/containerization#toc-applicatio-1xzdLX6E

Imperva. (n.d.). *What is phishing | Attack techniques & scam examples*. Imperva. Retrieved May 5, 2022, from https://www.imperva.com/learn/application-security/phishing-attack-scam/

Javatpoint. (n.d.). *JVM | Java Virtual Machine*. Javatpoint. Retrieved May 5, 2022, from https://www.javatpoint.com/jvm-java-virtual-machine

JSON. (n.d.). *Introducing JSON*. JSON.org. Retrieved April 1, 2022, from https://www.json.org/json-en.html

JWT. (n.d.). *Introduction to JSON Web Tokens*. JWT.io. Retrieved April 1, 2022, from https://jwt.io/introduction

Kaspersky. (n.d.). *Brute Force Attacks: Password Protection*. Kaspersky. Retrieved May 5, 2022, from https://www.kaspersky.com/resource-center/definitions/brute-force-attack

Kelley, K. (2022, February 21). *What is GitLab and How to Use It? [2022 Edition]*. Simplilearn. Retrieved May 3, 2022, from https://www.simplilearn.com/tutorials/git-tutorial/what-is-gitlab#what_is_git

Klein, J. F. (2022, April 07). *Amdahl's law*. Wikipedia. Retrieved May 9, 2022, from https://en.wikipedia.org/wiki/Amdahl%27s_law

Kopf, B. (n.d.). *The Power of Figma as a Design Tool*. Toptal. Retrieved April 7, 2022, from https://www.toptal.com/designers/ui/figma-design-tool

Kvarving, O., Hadzalic, E., & Sonen, S. R. (2022). *Vedlegg D Incident Report Forprosjektplan* (1.4) [Forprosjektplanen benyttet under oppstart av AIRS-prosjektet (127)] [Elektronisk dokument].

Kvarving, O., Sonen, S. R., & Hadzalic, E. (n.d.). *Vedlegg E Gantt-Diagram-127* [Gantt-diagram for gruppe 127] [Elektronisk dokument].

Lord, N. (2020, September 29). *Uncovering Password Habits: Are Users' Password Security Habits Improving? (Infographic)*. Digital Guardian. Retrieved May 5, 2022, from https://digitalguardian.com/blog/uncovering-password-habits-are-users-password-security-habits-improving-infographic

Maayan, G. D. (n.d.). *User Authentication Methods & Technologies to Prevent Breach*. ID R&D. Retrieved May 5, 2022, from https://www.idrnd.ai/5-authentication-methods-that-can-prevent-the-next-breach/

MariaDB. (n.d.). *Relational Databases: Foreign Keys*. MariaDB. Retrieved March 25, 2022, from https://mariadb.com/kb/en/relational-databases-foreign-keys/

Marshall University. (2019, January 25). *New Service - Multi-Factor Authentication - Information Technology*. Marshall University. Retrieved May 10, 2022, from https://www.marshall.edu/it/2019/01/25/mfa/

McCoy, L. (n.d.). *Microsoft Azure Explained: what it is and how to use it*. CCB Technology. Retrieved May 3, 2022, from https://ccbtechnology.com/what-microsoft-azure-is-and-why-it-matters/

MDN Web Docs. (2022, March 24). *Window.sessionStorage - Web APIs | MDN*. MDN Web Docs. Retrieved May 11, 2022, from https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage

MDN Web Docs. (2022, April 28). *Introduction to client-side frameworks - Learn web development | MDN*. MDN Web Docs. Retrieved May 3, 2022, from https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction

Microsoft. (n.d.). *Godkjenning med flere faktorer – Microsoft Sikkerhet*. Microsoft. Retrieved May 7, 2022, from https://www.microsoft.com/nb-no/security/business/identity-access-management/mfa-multi-factor-authentication

Microsoft. (n.d.). *Hva er Azure – Microsoft-skytjenester*. Microsoft Azure. Retrieved May 3, 2022, from https://azure.microsoft.com/nb-no/overview/what-is-azure/

Microsoft. (n.d.). *Videokonferanser, møter og anrop*. Microsoft Teams. Retrieved May 3, 2022, from https://www.microsoft.com/nb-no/microsoft-teams/group-chat-software#Key-message1

MITRE. (n.d.). *CVE - CVE ID Syntax Change (Archived)*. CVE. Retrieved May 12, 2022, from https://cve.mitre.org/cve/identifiers/syntaxchange.html

Muchmore, M., & Duffy, J. (2022, March 23). *Google Drive Review*. PCMag. Retrieved April 5, 2022, from https://www.pcmag.com/reviews/google-drive

Mullvad VPN. (n.d.). *The Basics of Encryption*. Mullvad VPN. Retrieved 05 03, 2022, from https://mullvad.net/en/help/basics-encryption/?gclid=Cj0KCQjwpcOTBhCZARIsAEAYLuVF6_n0inGLOc7sCEu1GGm5HwTr2VnI8rKsGa2KPWHld8s2RapO-ZgaAlLWEALw_wcB

MySQL. (n.d.). *MySQL Workbench*. MySQL. Retrieved April 5, 2022, from https://www.mysql.com/products/workbench/

NordPass. (2020, August 12). *Military-grade encryption explained*. NordPass. Retrieved May 10, 2022, from https://nordpass.com/blog/military-grade-encryption-explained/

Okta. (n.d.). *Hashing Algorithm Overview: Types, Methodologies & Usage*. Okta. Retrieved April 5, 2022, from https://www.okta.com/identity-101/hashing-algorithms/

Okta. (n.d.). *What Is Token-Based Authentication?* Okta. Retrieved May 9, 2022, from https://www.okta.com/identity-101/what-is-token-based-authentication/

Olenski, J. (2016, April 15). *Certificate-Based Authentication? - Blog*. GlobalSign. Retrieved May 9, 2022, from https://www.globalsign.com/en/blog/what-is-certificate-based-authentication

OneSpan. (n.d.). *What is Biometric Authentication? Use Cases, Pros & Cons*. OneSpan. Retrieved May 9, 2022, from https://www.onespan.com/topics/biometric-authentication

Oracle. (n.d.). *SecureRandom (Java Platform SE 8)*. Oracle Help Center. Retrieved May 11, 2022, from https://docs.oracle.com/javase/8/docs/api/java/security/SecureRandom.html

Oracle. (n.d.). *What Is a Relational Database*. Oracle. Retrieved March 25, 2022, from https://www.oracle.com/database/what-is-a-relational-database/

OWASP. (n.d.). *HTML5 Security*. OWASP Cheat Sheet Series. Retrieved May 14, 2022, from https://cheatsheetseries.owasp.org/cheatsheets/HTML5_Security_Cheat_Sheet.html

OWASP. (n.d.). *Password Storage*. OWASP Cheat Sheet Series. Retrieved April 5, 2022, from https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Panda, P. (2021, January 21). *Five cryptographic key protection best practices*. Security Boulevard. Retrieved May 10, 2022, from https://securityboulevard.com/2021/01/five-cryptographic-key-protection-best-practices/

Perry, M. (2022, January 7). *7 Things to Consider To Build Scalable Web Applications*. Qovery. Retrieved May 8, 2022, from https://www.qovery.com/blog/7-things-to-consider-to-build-scalable-web-applications

Prabhu, R. D. (2021, July 29). *What Is Biometric Authentication? Definition, Benefits, and Tools*. Toolbox. Retrieved May 10, 2022, from https://www.toolbox.com/it-security/identity-access-management/articles/what-is-biometric-authentication-definition-benefits-tools/

Preziuso, M. (n.d.). *Password Hashing: Scrypt, Bcrypt and ARGON2 | by Michele Preziuso | Analytics Vidhya*. Medium. Retrieved May 9, 2022, from https://medium.com/analytics-vidhya/password-hashing-pbkdf2-scrypt-bcrypt-and-argon2-e25aaf41598e

Red Hat. (2020, May 8). *Topics Understanding APIs What is a REST API?* Red Hat. Retrieved April 5, 2022, from https://www.redhat.com/en/topics/api/what-is-a-rest-api

ScienceDirect. (n.d.). *Biometric Authentication*. ScienceDirect. Retrieved May 8, 2022, from https://www.sciencedirect.com/topics/computer-science/biometric-authentication

Sharma, L. (2022, January 16). *Why is Testing Necessary and Important? | ISTQB | ToolsQA*. Tools QA. Retrieved May 10, 2022, from https://www.toolsqa.com/software-testing/istqb/why-is-testing-necessary/

Sonen, S. R., Hadzalic, E., & Kvarving, O. (2022). *Vedlegg H Systemdokumentasjon for gruppe 127* (X.X) [Systemdokumentasjon for produkt utviklet av gruppe 127] [Elektronisk dokument].

Sonen, S. R., Kvarving, O., & Hadzalic, E. (2022). *Vedlegg A Arbeidskontrakt for bachelorgruppe 127* (1.2) [Arbeidskontrakten til medlemmene i gruppe 127.] [Elektronisk dokument].

ssl2buy. (n.d.). *Symmetric vs. Asymmetric Encryption - What are differences?* SSL2BUY. Retrieved May 10, 2022, from https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences

Swoop. (2020, July 20). *User Authentication: Understanding the Basics & Top Tips*. Swoop Password-Free Authentication. Retrieved May 5, 2022, from https://swoopnow.com/user-authentication/

Titan UiO. (2016, October 30). *Dårlige passordråd fra Datatilsynet*. Titan. Retrieved May 10, 2022, from https://titan.uio.no/blogg-blogg-blogg/2016/darlige-passordrad-fra-datatilsynet

Townsend, P. (2019, March 25). *RSA vs AES Encryption - A Primer*. Townsend Security. Retrieved May 10, 2022, from https://info.townsendsecurity.com/rsa-vs-aes-encryption-a-primer

Visma. (2022, March 10). *Hva er API og API-integrasjon?* Visma. Retrieved March 25, 2022, from https://www.visma.no/blogg/hva-er-api-sporsmal-og-svar/

Visual Paradigm. (n.d.). *Free Visual Paradigm Online*. Visual Paradigm Online. Retrieved May 3, 2022, from https://online.visual-paradigm.com/diagrams/solutions/free-visual-paradigm-online/

Vue.js. (2022, Feb 5). *Security*. Vue.js. Retrieved Apr 5, 2022, from https://vuejs.org/guide/best-practices/security.html

Vuex. (2022, April 6). *Getting Started*. Vuex. Retrieved May 12, 2022, from https://vuex.vuejs.org/guide/

Web Application Testing - Techniques. (n.d.). Tutorialspoint. Retrieved May 10, 2022, from https://www.tutorialspoint.com/software_testing_dictionary/web_application_testing.htm

Web Service Security. (n.d.). OWASP Cheat Sheet Series. Retrieved May 12, 2022, from https://cheatsheetseries.owasp.org/cheatsheets/Web_Service_Security_Cheat_Sheet.html

Wikipedia. (n.d.). *Integrert utviklingsmiljø – Wikipedia*. Wikipedia. Retrieved May 5, 2022, from https://no.wikipedia.org/wiki/Integrert_utviklingsmilj%C3%B8

Wikipedia. (n.d.). *Password Hashing Competition*. Wikipedia. Retrieved May 5, 2022, from https://en.wikipedia.org/wiki/Password_Hashing_Competition

Wikipedia. (2022, Feb 9). *Web scraping*. Wikipedia. Retrieved April 5, 2022, from https://en.wikipedia.org/wiki/Web_scraping

Wikipedia. (2022, March 7). *Web crawler*. Wikipedia. Retrieved April 5, 2022, from https://en.wikipedia.org/wiki/Web_crawler

Woland, A. (n.d.). *How does certificate-based authentication work?* Network World.com. Retrieved May 9, 2022, from https://www.networkworld.com/article/2226498/infrastructure-management-simply-put-how-does-certificate-based-authentication-work.html

Yubico. (n.d.). *What is Certificate-Based Authentication*. Yubico. Retrieved May 9, 2022, from https://www.yubico.com/resources/glossary/what-is-certificate-based-authentication/

Figur- og tabelliste

Figurer:

Tabeller:

Vedlegg brukt i hovedrapporten

- A:
  Arbeidskontrakt-127
- C:
  Timeliste-127
- D:
  Forprosjektplan-127
- E:
  Gantt-Diagram-127
- F:
  Visjonsdokument-127
- H:
  Systemdokumentasjon-127
- J:
  Spørreundersøkelse-127
- K:
  Svar spørreundersøkelse-127

A Arbeidskontrakt for bachelorgruppe 127

Medlemmer: Endré Hadzalic, Odin Kvarving, Scott Rydberg Sonen

[V1.2]

[Revidert 25/3-2022, for tilpassing til jobbing hjemmefra]

Innledende tekst

Denne arbeidskontrakten bygger på et sett med typiske mål, oppgavefordelinger, prosedyrer og retningslinjer for interaksjoner for studentarbeider. Arbeidskontrakten er utfylt med *egne* fortolkninger av hva man mener med disse og hvordan man skal oppnå dette.

1. Mål for prosjektet

Effektmål

- Målet for gruppa med dette prosjektet er å lage et produkt som tilfredsstiller oppdragsgiveren sine mål og krav for ønsket produkt. Det skal være enkelt og forståelig å bruke for kundebasen som oppdragsgiver har sett for seg skal bruke produktet, og være godt dokumentert for enkel vedlikehold og feilsøking.
- Vi vil også utforske forskningsspørsmål rundt temaet som kan bidra til læring innen feltet, og knytte disse opp mot produktet vi skal utvikle.
- Vi vil også sette oss godt inn i bedriftens arbeidsmetodikk og standarder slik at produktet faller i god smak hos bedriften generelt.
- Vi vil sitte igjen med et produkt som vi er stolte av, og som kan legges ut offentlig som et eksempel på godt utført arbeid.
- Vi vil også opparbeide gode rutiner for gruppearbeid og produksjon av et større produkt som skal brukes i arbeidslivet.

Prosessmål

- Produktet fullføres innen den gitte tidsramme på minst 500 timer per medlem i gruppa.
- Prosessen underveis blir dokumentert nøye slik at hovedrapporten klarer å reflektere gruppas gjennomføring av prosjektet på best mulig måte.
- Alle frister underveis blir fulgt. Dette gjelder ikke kun dokumenter, men også frister knyttet til blant annet produktets funksjonalitet og utseende. Frister satt av gruppen selv skal være fulgt, samt alle eventuelle frister av oppdragsgiver.
- Vi vil holde god dialog med både oppdragsgiver og rådgiver underveis i prosjektet.
- Sørge for at kommunikasjonen innad i gruppen er god gjennom hele prosjektprosessen, slik at vi har god oversikt over hva som er gjort og hva som skal gjøres.
- God stemning mellom gruppemedlemmene.
- Til slutt skal vi sitte igjen med gode rutiner for gruppearbeid, produksjon og utvikling av et produkt som kan anvendes i arbeidslivet.

Resultatmål

- Produktet som oppdragsgiver sitter igjen med etter endt prosjekt skal kunne anvendes til å løse problemet oppdragsgiver ønsket å løse i utgangspunktet.
- Produktet skal også gjøre gruppen stolt, da vi selv har satt konkrete mål og krav.
- Vi vil levere en godt dokumentert kode som enkelt kan taes over av en annen gruppe ved eventuell videreutvikling, eller vedlikeholdes enkelt.

- Hovedrapporten skal besvare forskningsspørsmålet vårt og ha utfyllende punkter, samt inneholde alt av nødvendig dokumentasjon og vedlegg.

2. Roller og oppgavefordeling

A. Teamleder: Odin Kvarving

Beskrivelse: Kontaktperson for gruppa, koordinerer/planlegger aktiviteter og møter mellom veiledere/eksterne parter og gruppa. Holder kontakt med eksterne parter på vegne av gruppa.

B. Dokumentansvarlig: Scott Rydberg Sonen

Beskrivelse: Har ansvar for at frister følges og at timelisten blir fulgt opp. Det meste av dokument lagres i Google Drive. Dokumentansvarlig skal holde orden på dette.

C. Møteleder: Rulleres på av gruppemedlemmene

Beskrivelse: Ordstyrer underveis i møtet.

D. Møtereferent: Endré Hadzalic

Beskrivelse: Skriver møtereferat under møter.

E. Veileder: Olav Skundberg

Beskrivelse: Hjelper gruppen med dokumentasjon og prosess. Kvalitetssikrer dokumentasjon. Veileder gruppen i riktig retning.

F. Oppgavestiller: Vegard Kjerstad

Beskrivelse: Har utformet oppgaven for oss. Vil hjelpe til underveis i prosessen angående produkt og eventuelt dokumentasjon. Kvalitetssikrer produkt og utviklingen av det.

3. Prosedyrer

A. *Møteinnkalling*
Møteleder sender innkalling til gruppemedlemmer, veileder og oppdragsgiver. Møteinnkallinger sendes på mail oppgitt i standardavtalen, og burde senest bli sendt ut ett døgn før møtet. Møteinnkallinger skal inneholde varighet, agenda og saksunderlag (kan være lenker)

B. *Varsling ved fravær eller andre hendelser*
Dersom man kommer for sent eller ikke kan møte, varsler man ifra til Messenger-gruppen.

C. *Dokumenthåndtering*
Alt av dokumenter lagres i Google Drive. Samskriving skjer i Google Drive. Kodefiler lagres på GitHub/GitLab.
Noen dokumenter har revisjonslogg som viser når filen sist ble endret, ved skriving eller annet.

D. *Innleveringer av gruppearbeider*
Oppgaver med frister påbegynnes så tidlig som mulig, og jobbes jevnt med frem til fristen. Ferdigstilles så tidlig som mulig eller før frist. Innen 12 timer før en gitt dokumentfrist så skal innholdet gåes igjennom i plenum av alle medlemmene i gruppa. Ved inkludering av eksterne personer i dokumenter så skal de inkluderes i

gjennomgang før levering. Dokumenter skal selvfølgelig leveres innen gitte frister.

4. Interaksjon

A. ***Oppmøte og forberedelse***

Ved jobbing hjemmefra skal studentene alltid være til stede på valgt kommunikasjonsmedium over nett og jobbe sammen under kjernetiden som varer fra kl 09:00 til 13:00 alle hverdager. Utenom kjernetiden kan arbeidstimene disponeres fritt utover dagen.

Ved fysisk oppmøte avtales arbeidsdagen på forhånd av gruppemedlemmene. Arbeidsdagen varer i utgangspunktet i 7.5timer, inkludert jobbing hjemmefra. Endringer må avtales og godkjennes i plenum.

Dersom et gruppemedlem har en annen viktig avtale eller sykdom, skal dette gis beskjed om senest kl 23:59 dagen før. Ved akutt sykdom eller forsinkelse, skal dette helst gis beskjed om før avtalt oppmøtetid/kjernetid.

Dersom et gruppemedlem må dra før avtalt tid eller slutten av kjernetiden, skal dette gis beskjed om senest i løpet av arbeidsdagen.

Hvis noe skal forberedes, for eksempel dokumentasjon, skal dette være gjort før man møter opp.

B. ***Tilstedeværelse og engasjement***

Gruppemedlemmer skal i all hovedsak fokusere på gitt arbeidsoppgave under arbeidstimer. Ved passende anledninger og enighet i gruppa kan medlemmer ta seg inntil én 5-min pause per time (utenom lunchpause). Toalettpause er alltid tillatt.

I starten av arbeidsdagen er det satt av 10 min til en felles quiz i gruppa. Dette for å "varme opp" hodet.

C. ***Hvordan støtte hverandre***

Om man merker at et medlem i gruppa er deprimert/mindre motivert og ikke tar dette opp selv, bør resterende medlemmer i gruppa tilstrebe å arrangere et møte med mål å bedre situasjonen for nevnt medlem.

D. ***Uenighet, avtalebrudd***

I gruppen følger vi demokrati. Unntak kan forekomme, dersom en idé er god nok til å diskuteres. Uenighet håndteres derfor hovedsakelig demokratisk. Dersom alle gruppemedlemmene har forskjellige meninger om en sak, skal dette videreformidles til oppdragsgiver/veileder. Valget blir avgjort av oppdragsgiver/veileder.

Avtalebrudd skal håndteres med passende straff. Dersom det forekommer 3-5 avtalebrudd for én enkeltperson, skal vedkommende spandere lunsj på resten av gruppen.

Signaturer:

ENDRÉ HADZALIC

ODIN KVARVING

SCOTT RYDBERG SONEN

Incident Response System
C Timeliste

Endré Hadzalic
Odin Kvarving
Scott Rydberg Sonen

NTNU

Kunnskap for en bedre verden

Timeliste Bachelorprosjekt

UKE	Odin			Endré			Scott		
		Totalt:	371,50		Totalt:	390,50		Totalt:	406,50
	Dato	Timer	Beskrivelse	Dato	Timer	Beskrivelse	Dato	Timer	Beskrivelse
2	13.01.2022	2,00	Oppstart	13.01.2022	2,00	Oppstart	13.01.2022	2,00	Oppstart
	14.01.2022	4,00	Videre arbeid med dokumenter	14.01.2022	4,00	Videre arbeid med dokumenter	14.01.2022	4,00	Videre arbeid med dokumenter
3	18.01.2022	2,00	Forberedelse til oppstartsmøte	18.01.2022	2,00	Forberedelse til oppstartsmøte	18.01.2022	2,00	Forberedelse til oppstartsmøte
	19.01.2022	1,00	Forberedelse til oppstartsmøte	19.01.2022	1,00	Forberedelse til oppstartsmøte	19.01.2022	1,00	Forberedelse til oppstartsmøte
	20.01.2022	4,00	Forberedelse og oppstartsmøte	20.01.2022	4,00	Forberedelse og oppstartsmøte	20.01.2022	4,00	Forberedelse og oppstartsmøte
	21.01.2022	3,00	Visjonsdokument	21.01.2022	3,00	Visjonsdokument	21.01.2022	3,00	Visjonsdokument
4							24.01.2022	2,00	Arbeid med forprosjektsplan
	25.01.2022	4,00	Arbeid med forprosjektsplan	25.01.2022	4,00	Arbeid med forprosjektsplan	25.01.2022	4,00	Arbeid med forprosjektsplan
	26.01.2022	1,00	Fullføring av forprosjektsplan				26.01.2022	1,00	Fullføring av forprosjektsplan
	27.01.2022	4,00	Fullføring av forprosjektsplan og arbeid med visjonsdokument	27.01.2022	4,00	Fullføring av forprosjektsplan og arbeid med visjonsdokument	27.01.2022	4,00	Fullføring av forprosjektsplan og arbeid med visjonsdokument
	28.01.2022	2,00	Ukesrapport og jobbing med visjonsdokument	28.01.2022	2,00	Ukesrapport og jobbing med visjonsdokument	28.01.2022	2,00	Ukesrapport og jobbing med visjonsdokument
5	01.02.2022	2,00	Visjonsdokument, issueboard, forskningsmetode, møteinnkallinger	01.02.2022	2,00	Visjonsdokument, issueboard, forskningsmetode	01.02.2022	2,00	Forskningsmetode, visjonsdokument, issueboard
	02.02.2022	5,00	Møte med veileder, arbeid med visjonsdokument, forskningsmetode og issueboard	02.02.2022	5,00	Møte med veileder, arbeid med visjonsdokument, forskningsmetode og issueboard	02.02.2022	5,00	Visjonsdokument, issueboard, forskningsmetode
					2,00	Research på mulige frontend rammeverk og hvordan "innhenting av sikkerhetsdata" kan fåregå.			
	03.02.2022	4,00	Forskningsmetode og spørsmål, issueboard, kravdokumentasjon og wireframe	03.02.2022	3,00	Forskningsmetode og spørsmål, issueboard, kravdokumentasjon og wireframe	03.02.2022	4,00	Forskningsmetode og spørsmål, issueboard, kravdokumentasjon og wireframe
	04.02.2022	5,00	Wireframe	04.02.2022	5,00	Wireframe	04.02.2022	5,00	Wireframe
6	07.02.2022	4,00	Møte oppdragsgiver, visjonsdokument, wireframe	07.02.2022	4,00	Møte oppdragsgiver, visjonsdokument, wireframe	07.02.2022	4,00	Møte oppdragsgiver, visjonsdokument, wireframe
	10.02.2022	3,00	Domenemodell, møteinnkallinger, ukerapport	10.02.2022	3,00	Domenemodell, møteinnkallinger, ukerapport	10.02.2022	3,00	Domenemodell, møteinnkallinger, ukerapport
7				14.02.2022	2,00	Færdigstilling av wireframes og domenemodell	14.02.2022	2,00	Wireframe, domenemodell, git-repo
	15.02.2022	2,00	Oppstart med progging	15.02.2022	2,00	Oppstart med progging	15.02.2022	2,00	Git-repo og frontend-oppstart
	16.02.2022	4,00	Arbeid med MVP, møte med veileder	16.02.2022	5,00	Arbeid med MVP, møte med veileder	16.02.2022	6,00	Arbeid med MVP, møte med veileder
	17.02.2022	5,00	Arbeid med MVP (Database)	17.02.2022	4,00	Arbeid med Jsoup og ThreadPools	17.02.2022	7,00	Arbeid med frontend i MVP
	18.022022	5,00	Backend service, repo & test				18.02.2022	4,00	Arbeid med frontend i MVP
8				23.02.2022	2,00	Implementerte ThreadPools og fjærna Spring			
	24.02.2022	6,00	Databasejobbing	24.02.2022	6,00	Jobba med Hashing og starta på database i observer	24.02.2022	4,00	Frontend og JSDoc
				25.02.2022	2,00	Satt opp kobling mot database i obtain	25.02.2022	1,00	La til flere komponenter i frontend
9	28.02.2022	5,00	Entities og testing backend	28.02.2022	6,00	Entities og webscraping	28.02.2022	5,00	Videre arbeid med frontend
	01.03.2022	2,00	testing						
							02.03.2022	4,00	Frontendarbeid
	03.03.2022	4,00	Databaseoppkobling MySQL	03.03.2022	5,00	Mekka støtte for dynamiske nettsteder.	03.03.2022	5,00	Frontend
				04.03.2022	4,00	Begynte på webscraping	04.03.2022	4,00	Frontend
				06.03.2022	12,00	Store framskritt i observe_backend	06.03.2022	1,00	Gjorde ferdig frontend for MVP og fylte ut ukesrapporten

Timeliste Bachelorprosjekt

UKE	Odin			Endré			Scott		
		Totalt:	371,50		**Totalt:**	390,50		**Totalt:**	406,50
10	07.03.2022	10,00	Prøvde å utbedre feil i backend for databasen til MVP-presentasjon	07.03.2022	4,00	Forberedelse til MVP og jobbing med scraper.	07.03.2022	5,00	Gikk over frontend, gjorde litt endringer og la til manglende JSDoc. Lagde presentasjon for møte 08.03.22. Prøvde å løse et problem i backend sammen med Odin
	08.03.2022	6,00	Fikset feil i backend for database	08.03.2022	7,00	MVP presentasjon, bugfixing og refaktor av entitetopsett.	08.03.2022	7,00	MVP presentasjon, bugfixing, refaktor av entitetopsett, videre frontendarbeid
				10.03.2022	3,00	Jobba med å integrere Provider			
							11.03.2022	4,00	Videreutvikling av frontend
11	15.03.2022	6,00	Implementerte nye klasser etter tilbakemelding fra Atea	15.03.2022	10,00	Jobba med implementasjon av RSS-feed	15.03.2022	7,00	Videreutvikling av filtrering i frontend samt ny funksjonalitet
	16.03.2022	5,00	Fikse filtrering og sortering i backend.	16.03.2022	5,00	Fortsatte med implementasjon av RSS	16.03.2022	6,00	Frontend
12	23.03.2022	1,00	Veiledningsmøte med Olav	23.03.2022	1,00	Veiledningsmøte med Olav	23.03.2022	1,00	Veiledningsmøte med Olav
	24.03.2022	4,00	Arbeid med poster og presentasjon	24.03.2022	6,00	Arbeid med poster og presentasjon	24.03.2022	6,00	Poster og presentasjon
	25.03.2022	5,00	Arbeid med hovedrapport	25.03.2022	5,00	Arbeid med hovedrapport	25.03.2022	5,00	Arbeid med hovedrapport
13	28.03.2022	9,00	Jobbet videre med filtrering i backend og tester	28.03.2022	7,00	Impementasjon av RSS feed-scraping	28.03.2022	8,00	Arbeid med login og jwt token. La til komponenter i frontend og nødvendige klasser i backend
	29.03.2022	6,00	Prøvde å fikse filtrering backend, mangler filtrering på dato	29.03.2022	7,00	Plages me spring security	29.03.2022	7,00	Videre arbeid med frontend. Omfaktorering av komponenter. JSDoc der det manglet. Fikset sidebytting mm.
	30.03.2022	5,00	Fikset filtrering med Scott og merget med web-scraper branch	30.03.2022	5,00	Fikk implementert JWT og autentisering i backend	30.03.2022	6,00	Fikset filtrering i backend med Odin og jobbet med popups i frontend. Gjorde nettsiden mer oversiktlig
	31.03.2022	6,00	Fikset MySQL-database men fikk problemer med H2-databasen	31.03.2022	7,00	Jobbing med poster-pres og hjelping av Odin	31.03.2022	6,00	Forberedelser til poster-presentasjon og litt arbeid med frontend.
	01.04.2022	5,00	Jobbet videre med H2-databasen tester	01.04.2022	5,00	Kobling av backends opp mot database, og bruker-endepunkter i backend	01.04.2022	5,00	La til nye punkter på teoridelen i hovedrapporten og fortsatte med frontend.
				03.04.2022	2,00	Startopp med dockerizing, fiksa profiler i obtain			
14	04.04.2022	9,00	Jobbet sammen med Scott og Endré for å manuellt merge sammen fungerende versjoner av prosjektet. Fikk på denne måten implementert både H2-database og MySQL-database, samt oppdaterte versjoner av både frontend og backend	04.04.2022	6,00	Samarbeidet med Odin for å implementere H2-database og MySQL-database.	04.04.2022	7,00	Fikk implementert jwt i frontend, med bruk av sessionStorage og store. Støtet på noen problemer rundt login. Fikk merget manuelt med Odin og Endré.
	05.04.2022	9,00	Jobbet videre på teori- og metodedelen i hovedrapporten sammen med Scott og Endré. Lage Controller-klasser i backend.	05.04.2022	6,00	Jobbet videre på teori- og metodedelen i hovedrapporten sammen med Odin og Scott	05.04.2022	7,00	Jobbet videre på teori- og metodedelen i hovedrapporten sammen med Odin og Endré
	06.04.2022	9,00	Testet controller-metodene med Postman og fikset noen metoder som ikke funket i backend. Hadde store problemer med filter-metoden i SecurityIssueService, men fikk til slutt fikset det.	06.04.2022	6,00	Jobbing med dockerizing av obtain	06.04.2022	7,00	Fikk fikset login og JWT/session cookie etter litt om og men. Samarbeidet med Odin om sortering og filtrering i backend. Fikk gjort ferdig nødvendig funksjonalitet i alle popups og feeds. La til mer JSDoc
	07.04.2022	8,00	Jobbet videre på hovedrapporten og deltok i veiledningsmøte. Fortsatte med controller-metoder i backend.	07.04.2022	9,00	Jobbet videre på hovedrapporten og deltok i veiledningsmøte, deretter så fortsatte jeg med dockerizing	07.04.2022	5,00	Jobbet videre på hovedrapporten og deltok i veiledningsmøte
	08.04.2022	6,00	Fikset filtrerings- og sorteringsmetoden i backend og gjorde noen endringer på modell-klassene for å oppdatere databasen	08.04.2022	6,00	cleanup i observe, og dockerizing av begge backends er ok	08.04.2022	6,00	Jobbet videre på frontend. Fikk laget flere connections til backend. Fikk displayet alle security issues i feeden :D. La til nye icons og gjorde nettsiden mer estetisk.
							10.04.2022	1,00	Jobbet med å implementere filtrering fra backend.
15	11.04.2022	2,00	Møte med oppdragsgiver	11.04.2022	2,00	Møte me Vegard	11.04.2022	3,00	Møte med oppdragsgiver og fortsettelse på frontend.
							14.04.2022	1,00	Fortsatte litt på finjustering av security issue feeden og funderte videre på filtrering

Timeliste Bachelorprosjekt

UKE	Odin			Endré			Scott		
		Totalt:	371,50		**Totalt:**	390,50		**Totalt:**	406,50
16	19.04.2022	5,00	Møte med oppdragsgiver og medlem av Atea Incident Response Team. Jobbet videre med HTTPS	19.04.2022	5,00	Hadde møte med vegard og Geir olav og fortsatte deretter med systemet	19.04.2022	7,00	Gjorde forberedelser og hadde møte med Vegard og Geir Olav. Fortsatte på funksjonalitet rundt filtrering, søk, popups og andre ting. Ukerapporter. Fikk implementert lagring av status fra sikkerhetsfeed, som sidetall og valgt feil.
	20.04.2022	7,00	Oppdaterte modell-klasser, fikk implementert HTTPS og fikset på filtrerings-metoden.	20.04.2022	7,00	Jobba med opplasting til azure.	20.04.2022	8,00	Fikk ordnet filtrering og sortering. Refaktorerte en del av koden. Provider- og CVE-feedene skal bli ordnet.
	21.04.2022	6,00	La til metoder i forskjellige service-klasser og oppdaterte noen av metodene	21.04.2022	6,00	Fiksa opplasting og statisk serving av bilder i backend.	21.04.2022	6,00	Hadde møte med Morten hos Atea for å få tilgang til kontor/møterom. Fortsatte med refaktorering, og ble noenlunde ferdig med å implementering alt av GET requests til feeds, cards og popups.
	22.04.2022	6,00	Omstrukturerte CVE-struktuten i databasen og opprettet modell, service, controller og repo klasser for CVE	22.04.2022	6,00	Jobba med implementasjon av CVE-entitet i obtain og skrev ferdig doc for obtain.	22.04.2022	7,00	Tok i bruk mer funksjonalitet fra backend og modifiserte feeds, popups og cards.
17	25.04.2022	9,00	Jobbet med merging av brancher og oppdaterte ulike metoder og modell-klasser	25.04.2022	11,00	jobba med merging av database inn i obtain, og litt frontend	25.04.2022	8,00	La til de fleste metoder for servicene i frontend, modifiserte/la til nye metoder i backend, la til annen manglende funksjonalitet
	26.04.2022	7,00	Fikset routing-problemer i frontend og begynte på sorterings og søk-funksjonalitet	26.04.2022	10,00	Jobba med implementering av kommentarfelt	26.04.2022	7,50	Jobbet videre på frontend og la til manglende funksjonalitet. Kom et godt stykke videre. Fikk ordnet AddCustomer, men mangler å legge til teknologier og profilbilde.
	27.04.2022	7,00	Fikset søk og sortering på alle sidene bortsett fra kunde-siden	27.04.2022	8,00	Jobba med chat, og lagring av nye brukere	27.04.2022	8,00	Jobbet med profilbilde-funksjonalitet og litt annen type smallfix
	28.04.2022	10,00	La til nye endpoints i flere av controller-klassene og fikset sortering og filtrering i alle tab-er i frontend	28.04.2022	10,00	Jobba med passordendring og profilbildendring for bruker	28.04.2022	9,50	Endret på ting og tang hvor det fantes bugs. Fikk merget new_frontend inn i develop sammen med Odin (litt av en merge). Fikset add og update av customer og technology, med bruk av bilde osv.
	29.04.2022			29.04.2022	9,00	Trådoptimaliserte filterfunksjonen	29.04.2022	8,00	Ferdig med editing og adding av nye entiteter i frontend (utenom User, for den tok Endré hånd om). Fikk koblet sikkerhetsfeil, teknologi og kunder sammen i frontend. Bugfixing her og der.
18	02.05.2022	10,00	Jobbet med å gjøre filtrerings-metode i backend raskere	02.05.2022	11,00	Justeringer på frontend	02.05.2022	12,00	Mye bugfixing her og der. La til småting som manglet. Mangler en fix i TechnologyPopup som omhandler kundeadding og removing.
	03.05.2022	9,00	Jobbet videre med resultat-delen av hovedrapporten og la til JavaDoc i backend	03.05.2022	7,00	Jobbing med spørreskjema	03.05.2022	9,00	Videre skriving på rapporten. La til ting på prosjekthåndboka. Fikk fikset siste bug som manglet. Gikk over applikasjonen for å se at alt fungerte som det skulle, og all funksjonalitet fungerte som det skulle :D
	04.05.2022	9,00	Hadde møte med Olav. Fortsatte med hovedrapporten og la til mer JavaDoc i backend	04.05.2022	7,00	Møte med Olav + Dockerizing og javadoc av obtain	04.05.2022	9,00	Møte med Olav. Fortsatte på rapporten. Skrev ferdig manglende JSDoc i frontend. Gjorde endringer i rapporten basert på tilbakemeldinger fra Olav.
	05.05.2022	7,00	Utforsket teori om sikker lagring av passord og brukerdata. Prøvde å implementere kryptering av data i databasen	05.05.2022	4,00	Så videre på hosting i Azure og opplasting av Docker.	05.05.2022	7,00	Fortsatte å fylle inn på hovedrapporten. Gjorde ferdig kravdokumentasjon og domenemodell, og endret litt på prosjekthåndboka.
	06.05.2022	11,00	Fikset kryptering av kundeinformasjon i database, skrev JavaDoc i backend	06.05.2022	9,00	Fiksing av CD og doc	06.05.2022	10,00	Fortsatte på rapporten og fikset siste bugs på frontend angående image og logout. Fikset zoom bug
	07.05.2022	3,00	Skrev JavaDoc for backend	07.05.2022	2,00	Skrev på resultadel	07.05.2022	1,50	Fortsatte på resultatdelen av rapporten og fylte ut midlertidig ukerapport for denne uka.
	08.05.2022	2,50	Resultatdel og forberedelser til møte.	08.05.2022	2,00	Resultatdel og forberedelser til møte.	08.05.2022	2,50	Fortsatte litt på resultatdelen av rapporten og gjorde forberedelser til møte for mandag.

Timeliste Bachelorprosjekt

UKE	Odin			Endré			Scott		
		Totalt:	371,50		**Totalt:**	390,50		**Totalt:**	406,50
19	09.05.2022	7,00	Rapport og forberedelser til møte. Diskuterte	09.05.2022	5,00	Rapport og forberedelser til møte. Samt møte med produktpresentasjon for Atea og Olav utpå dagen.	09.05.2022	7,00	Rapport og forberedelser til møte. Skrev møtereferat og diskuterte. Kom langt på min første del i resultatdelen.
	10.05.2022	6,00	Rapportskriving og siste veiledningsmøte.	10.05.2022	9,00	Starta dagen med rapportskrving. Siste møte med veileder Olav midt på dagen og fortsatte så med rapportskrig utover dagen. Gjorde ferdig Resultatdel for forskningsspørsmål 3.	10.05.2022	9,00	Skrev på resultatdelen i rapporten og hadde siste veiledningsmøte med Olav. Fortsatte med rapport etter møtet. Leste over deler av rapporten. Kom godt i gang på delen om ingeniørfaglige resultater.
	11.05.2022	6,00	Skrev på resultatdel av hovedrapporten	11.05.2022	8,50	Begynte på diskusjonsdel til forskningsspørsmål 3	11.05.2022	8,00	Fortsatte på ingeniørfaglige resultater og refaktorerte litt av teksten. Diskuterte angående noen punkter i gruppa. Fortsatte på ingeniørfaglige resultater på kveldstid.
	12.05.2022	7,00	Fortsatte på resultatdelene av hovedrapporten	12.05.2022	7,00	Skrev ferdig diskusjon for spørsmål 3	12.05.2022	8,00	Gjorde nesten ferdig ingeniørfaglige resultater. Gjorde så og si ferdig administrative resultater. Prosjekthåndboka
	13.05.2022	5,50	Gjorde ferdig resultatdelen på hovedrapporten	13.05.2022	5,00	Jobba med resultatdel for spørsmål 4	13.05.2022	9,50	Gjorde så og si ferdig resultatdelen. Skrev mye på diskusjonsdelen også. Fikk skrevet ferdig diskusjon rundt forskningsspørsmål 3.
	14.05.2022	3,00	Gjorde ferdig store deler av diskusjons-delen i hovedrapporten	14.05.2022	7,00	Gjorde ferdig resultater for spørsmål 4, samt mye av diskusjonsdelen	14.05.2022	5,00	Fortsettelse på diskusjon rundt ingeniørfaglige resultater. Skrev videre på diskusjonsdelen på kveldstid.
	15.05.2022	5,00	Skrev videre på diskusjonsdelen.	15.05.2022	5,00	Ferdigstilte diskusjon for spm 4.	15.05.2022	5,00	Fortsatte på siste deler av diskusjonsdelen og fikk startet på systemdokumentasjon. Lagde en ny databasemodell.
20	16.05.2022	5,00	Skrev ferdig diskusjonsdel, konklusjon, sammendrag, abstract og samfunnspåvirkning.	16.05.2022	5,00	Skrev ferdig diskusjonsdel, konklusjon og forord.	16.05.2022	5,00	Skrev ferdig diskusjonsdel, konklusjon og samfunnspåvirkning.
	17.05.2022			17.05.2022			17.05.2022		
	18.05.2022	12,50	Så gjennom og rettet opp på deler av hovedrapporten. Lagde README for backend-system og fikset JavaDoc-generering. Jobbet med Systemdokumentasjon.	18.05.2022	4,00	Så på hovedrapport og systemdokumentasjon.	18.05.2022	11,00	Så kjapt over hovedrapporten. Skrev videre på systemdokumentasjon. Skrev refleksjonsnotat. Så over alle dokument og gjorde de klar for innlevering
	19.05.2022	12,00	Så over hovedrapport og all annen dokumentasjon. Finpuss.	19.05.2022	12,00	Så over hovedrapport og all annen dokumentasjon. Finpuss.	19.05.2022	12,00	Så over hovedrapport og all annen dokumentasjon. Finpuss.

Incident Response System
D Forprosjektplan

Versjon 1.4

Endré Hadzalic
Odin Kvarving
Scott Rydberg Sonen

NTNU

Kunnskap for en bedre verden

Innholdsfortegnelse

1. Mål og rammer

1.1 Orientering

Tanker rundt oppgave

Gruppens ønske for oppgave, baserer seg på tidligere erfaringer og interesser. Vi hadde et felles fag i høstsemesteret 2021; IDATT2503 Sikkerhet i programvare og kryptografi. Vi tenkte derfor at et prosjekt innenfor sikkerhet og kryptografi hadde vært interessant. I vårsemesteret 2021 var vi på samme gruppe i et scrum-prosjekt. Dette prosjektet innebar utvikling av en fullstack-applikasjon. Derfor hadde vi en idé om et sikkerhetssystem implementert i en slik applikasjon.

Interesse for samarbeid

Interessen vår for et samarbeid med en ekstern bedrift var og er stor. Vi tenkte at et slikt samarbeid var en fin måte å gjennomføre bacheloroppgaven på, da man får god innsikt i hvordan en ekstern bedrift samarbeider med studenter, i tillegg til at samarbeidet kan gi oss innsikt over hvordan yrkeslivet som systemutvikler kan være. Vi tok kontakt med noen relevante bedrifter og etablerte tidlig kontakt med Atea. Videre kom vi i kontakt med oppdragsgiveren vår, som la frem en interessant og god oppgave.

1.2 Problemstilling / prosjektbeskrivelse og resultatmål

Vår problemstilling, også kalt utfordring, **er å sørge for pålitelig innhenting av sikkerhetsoppdateringer for Ateas kunder.** Oppgaven består i å lage en plattform som i hovedsak tar for seg innhenting av nye sikkerhetsfeil, svakheter og exploits for teknologier i en fastsatt kundebase. Hvis det blir registrert en ny sikkerhetsfeil som kan være til trussel for en gitt kunde, skal det sendes en varsling til kunden om hvilken teknologi det gjelder og hva sikkerhetsfeilen innebærer. Innhenting av denne dataen skal foregå ved å overvåke en rekke nettsteder fra både utviklere av de gitte teknologiene og forumer eller tredjepartsprogrammer som gir ut slik informasjon.

Sikker behandling av kundedata skal stå sentralt, da den inneholder informasjon om alle kundene til Atea, og hvilke teknologier som blir brukt. Usikker håndtering av slik data kan føre til at en angriper får tilgang til svært sensitiv informasjon, så håndtering av denne dataen skal utføres nøye. Plattformen skal først og fremst være et hjelpemiddel for Ateas Incident Response Team for å gjøre det enklere å innhente nye sikkerhetsfeil og sende ut varsel til kundene sine, noe som tidligere har vært en manuell prosess. Hvis det blir tid, og plattformen egner seg til det - kan den også utvides til en plattform hvor kundene til Atea kan logge seg inn, få oversikt over nye sikkerhetsfeil og se nye varslinger på egenhånd. Tilgangsstyring slik at kun autorisert personell kan få tilgang må være implementert.

Resultatmål

1. Et viktig resultatmål er at produktet skal fullføres innen den gitte tidsrammen på minst 500 timer per medlem i gruppen. Totalt skal vi derfor ha omtrent 1500 arbeidstimer i gruppen.
2. Et annet viktig resultatmål er at prosessen blir dokumentert nøye underveis. Dette skal gjøres slik at hovedrapporten også kan vise til refleksjon, rundt gruppens gjennomføring av prosjektet, på best mulig måte. Dette er ikke kun for at vi skal ha mer informasjon til å gå ut fra ved produksjon av hovedrapport i

tilknytning bachelor, men også for at eventuelt fremtidige overtakere av prosjektet skal kunne benytte seg av det.
3. Videre skal produkteier sitte igjen med et produkt som kan anvendes til å løse problemet som produkteier i utgangspunktet ønsket å løse.
4. Et annet resultatmål er at alle frister blir fulgt underveis. Dette gjelder ikke kun dokumenter, men også frister knyttet til blant annet produktets funksjonalitet og utseende. Disse fristene kan bli satt av gruppen selv og oppdragsgiver.
5. Det er også viktig å holde god dialog med både oppdragsgiver og veileder underveis i prosjektet.
6. Til slutt er det viktig for gruppen at det er god stemning mellom gruppemedlemmene. For å sikre et godt samarbeid skal vi sørge for at kommunikasjonen innad i gruppen er god gjennom hele prosjektprosessen, slik at vi har god oversikt over hva som er gjort og hva som skal gjøres.

1.3 Effektmål

1. Vi vil lage et produkt som tilfredsstiller oppdragsgiveren sine mål og krav for ønsket produkt. Det skal være enkelt og forståelig å bruke for kundebasen som oppdragsgiver har sett for seg skal bruke produktet, og være godt dokumentert for enkel vedlikehold og feilsøking.

2. Produktet skal være så sikkert og pålitelig som mulig, og det skal bidra til å effektivisere prosessen ved å innhente nye sikkerhetsfeil automatisk. Det skal være tydelig forbedring i hvor lang tid man bruker på å få oversikt over nye sikkerhetsfeil, og systemet skal kunne håndtere store mengder data.
3. Vi vil også utforske forskningsspørsmål rundt temaet som kan bidra til læring innen feltet, og knytte disse opp mot produktet vi skal utvikle.
4. Vi vil også sette oss godt inn i bedriftens arbeidsmetodikk og standarder slik at produktet faller i god smak hos bedriften generelt.
5. Vi vil sitte igjen med et produkt som vi er stolte av, og som kan legges ut offentlig som et eksempel på godt utført arbeid.
6. Vi vil også opparbeide gode rutiner for gruppearbeid og produksjon av et større produkt som skal brukes i arbeidslivet.

1.4 Rammer

Tid

Første punkt er tid. Ifølge emnebeskrivelsen gitt av fagansvarlige, skal hvert gruppemedlem ha omtrent 500 arbeidstimer i løpet av prosjektprosessen. Prosjektet har endelig innleveringsfrist 20. mai, altså uke 20. Vi har totalt 17 uker å gjennomføre prosjektet på. I uke 4 til uke 11 vil vi kombinere bachelorprosjekt sammen med et annet emne ved navn INGT2300 Ingeniørfaglig systemtenkning. Dette emnet er et intensivt emne, som betyr at vi blir nødt til å prioritere emnet fra mandag til onsdag. Dette gir oss torsdag og fredag til rådighet for bachelorprosjektet i uke 4 til uke 11.

Midler

Når det gjelder utstyr og penger, så kan vi få bruk for midler dersom vi må betale for ulike lisenser og slikt. Det kan hende at vi får bruk for programvare i prosjektet, som ikke er gratis, og da vil vi som studenter gjerne ha støtte fra oppdragsgiver.

Møterom

Vi i gruppen har et stort ønske om å jobbe sammen fysisk. Det mest ideelle hadde vært å få tilgang til et grupperom hos Atea og deres lokaler. På grunn av covid-situasjonen vi er i nå, så er det dessverre vanskelig, da Atea har sendt de fleste ansatte hjem i hjemmekontor. Dette gjør det også mindre sannsynlig for at de tar imot studentgrupper akkurat nå. Selv om det ikke blir noe fysisk sammenkomst med det første, så har vi etablert oss godt digitalt, slik at vi kan jobbe effektivt uavhengig om vi sitter i samme rom eller hver for oss.

2. Organisering

Vår oppdragsgiver er Vegard Kjerstad, leder for Ateas Incident Response Team (IRT). Dette betyr at Atea, IRT og Vegard Kjerstad er aktører i prosjektet. Noen har større roller enn andre, men alle er involvert. Videre er NTNU og veileder Olav Skundberg involvert i prosjektet. Underveis i prosessen vil vi holde en god dialog med Kjerstad og Skundberg. Disse to er de viktigste aktørene og interessentene i prosjektet.

3. Gjennomføring

3.1. Hovedaktiviteter

Dokumentasjon

Den første hovedaktiviteten er dokumentasjon. I starten er dette den viktigste aktiviteten, da vi får en overordnet oversikt over produkt og prosess. Hovedrapporten er det viktigste dokumentet vi skriver i dette prosjektet. Likevel er det andre viktige dokumenter som skal ferdigstilles relativt tidlig i prosessen. Dette er dokumenter som visjonsdokument, arbeidskontrakt, standardavtale og forprosjektplan. Alle i gruppen bidrar like mye til dokumentering av prosess og produkt. Dokumentasjon er derfor en viktig aktivitet, da alle i gruppen er med på å forme prosess og produkt. Dokumentasjon skal også tidvis fremvises til veileder av prosjektet for å kvalitetssikre deler som har blitt ferdigstilt.

Utviklingsfasen

Videre så skal vi over på utviklingsfasen av prosjektet. I denne fasen så jobber gruppen med å løse fastsatte delproblemer av hovedproblemet vi tolket i enighet med oppgavestiller i starten av prosjektet. Del-problemene som løses vil i hovedsak være softwarebasert og gruppen løser de ved hjelp av en spesifisert utviklingsmetodikk som for eksempel kan være scrum. Denne fasen av prosjektet varer sannsynligvis fra alle avtaler og forprosjekt-planlegging er fastsatt, til slutten av prosjektet som er 20.mai 2022. Dokumentasjon som skal bli produsert i tilknytning utviklingsfasen av prosjektet vil i hovedsak være tilknyttet til koden som produseres. Dette innebærer systemdokumentasjon, direkte kommentarer i kode (for eksempel javadoc) og brukermanual. Parallelt med dette så vil jo også ukesrapporter leveres ukentlig samtidig som at timeliste og hovedrapport blir det jobbet regelmessig med.

Forskningsspørsmål

En annen vesentlig del av prosjektet er å jobbe med forskningsspørsmålene som vi har stilt til oppgaven. Her skal vi utnytte etablert kunnskap om en spesifikk del av industrien for å svare på et forskningsspørsmål. Under prosessen skal vi utnytte en valgt forskningsmetode for å komme frem til et resultat, og deretter dokumentere dette tilstrekkelig. Dette skal også være en del av hovedrapporten, og er spesielt viktig da utviklere har hatt en tendens til å være veldig god på dokumentasjon av prosessen under utvikling, men ikke like god når det gjelder vitenskapelighet og videreutvikling av teorier. For å effektivt kunne jobbe med forskningsspørsmålene er det viktig at vi får innsikt fra noen i industrien som jobber med eller har erfaringer med det vi skal undersøke. Her har vi mulighet til å nå ut til flere personer innenfor Atea, og forhåpentligvis andre aktører som sier seg villig til å bidra til arbeidet. Arbeidet med forskningsspørsmålene vil gå parallelt med utviklingsfasen av prosjektet, hvor vi kommer til å sette av deler av tiden til jobbing med akkurat dette.

Avslutningsfasen

Til slutt har vi avslutningsfasen, hvor alt av dokumentasjon og kode skal ferdigstilles. Dette er fasen hvor vi som gruppe har kommet i mål med produktet. De siste ukene/dagene vil brukes effektivt for å ferdigstille hovedrapporten. Resterende dokumentasjon, som blant annet kodedokumentasjon og vedlegg på hovedrapporten, skal også finpusses. I avslutningsfasen er det detaljering og perfeksjonisme som er

viktig, da alt av dokumentasjon og produkt så og si er ferdig. I denne fasen vil hele gruppen jobbe sammen om finjusteringer av kode og dokumentasjon. Det er også viktig med en god dialog mellom oppdragsgiver og veileder, da de spiller en viktig rolle i denne fasen. En av deres oppgaver er å kvalitetssikre produkt og dokumentasjon. Kvalitetssikring er selvfølgelig viktig i hele prosessen. Dersom man ikke har gjort en slik vurdering underveis, vil det i denne fasen være viktig å fokusere på det.

3.2. Milepæler

- 28.01.22 - Innlevering av forprosjektplan
- 04.02.22 - Godkjenning av visjonsdokument med oppgavestiller
- 10.02.22 - Wireframes presentasjon
- Uke 13 - Presentasjon av problemstilling på engelsk
- 28.03.22 - Innlevering av poster (plakat)
- 28.02.22- MVP presentasjon
- 20.05.22 - Ferdigstilling og innlevering av prosjekt
- 27. mai 2022 - Presentasjon av prosjekt

4. Oppfølging og kvalitetssikring

4.1 Kvalitetssikring

Veileder kvalitetssikrer dokumentasjon

Gjennom prosjektet skal vi regelmessig vise fram dokumentasjon til veileder. Veilederen vil deretter kvalitetssikre arbeidet vårt, samt gi oss innspill på hva vi kan endre på. Alt av kritikk, både positive og negative tilbakemeldinger, vil være ekstremt viktig for gruppen, da dette garantert vil hjelpe oss med å få god og akademisk dokumentasjon.

God dialog med oppgavestiller

For å kvalitetssikre produktet vårt vil vi ha en god dialog med oppdragsgiver underveis. Oppgaven vår er relativt åpen, der vi som utviklere har frie tøyler til å velge hvilke teknologier vi skal bruke underveis i utviklingen av produktet. Oppdragsgiver har naturligvis et ønske om et ferdig produkt til slutt, men valg av blant annet programmeringsspråk, rammeverk og utviklingsplattform er det gruppen som bestemmer, dette er valg som selvfølgelig også vil begrunnes.

Testing av produkt

Et annet vittig verktøy vi kan bruke for kvalitetssikring, er testing av produkt. Produktet vi skal utvikle må være sikkert, da det håndterer sensitive data som en angriper ikke skal ha tilgang til. Vi må teste produktet tilstrekkelig for å vite at alt av funksjonalitet og sikkerhet er bra nok. Forskjellig typer testing vi kan gjennomføre er brukertesting og JUnit-testing. Disse testene tester forskjellige ting. Brukertesting omhandler design og funksjonalitet, da det er eksterne brukere som tester ut en enkel app. Denne appen er gjerne en slags wireframe med et førsteutkast for design. Gjennom brukertester får vi god kritikk på hva som er bra og dårlig med utvalgt design. JUnit er et rammeverk for testing som tester deler av koden. Disse testene omhandler funksjonalitet og sikkerhet. Testing av sikkerhet i produktet er viktig for vår og oppdragsgivers skyld.

Gruppen kvalitetssikrer hverandres arbeid

Vi som gruppe skal også kvalitetssikre hverandres arbeid. All kode og dokumentasjon skal være konsekvent. For koding betyr dette at vi blant annet skal bruke samme type variabelnavn (forkortelser eller enkle ord) og linjeskift hvor det skal være linjeskift. Dette gjør koden mye mer oversiktlig, både for utviklere og lesere. Når det gjelder dokumentasjon er det også viktig å være konsekvent med hvilke ord man bruker, bøyning, og linjeskift. Man må også være konsekvent med font-stiler.

4.2 Rapportering

Rapportering av progresjon skal skje til både veileder Olav Skundberg og oppdragsgiver Vegard Kjerstad *ved behov*. Ideelt sett vil vi ha møter med veileder minst hver 2. - 3. uke for å sjekke at utført arbeid tilfredsstiller de kravene som er satt til oppgaven, og for generell veiledning underveis. Møter med oppdragsgiver er noe mindre forutsigbart da han har en vanlig arbeidshverdag med egne kunder. Han har imidlertid opplyst om at han kan sette av tid dersom vi har behov for et møte, så dette skal være uproblematisk for vår del.

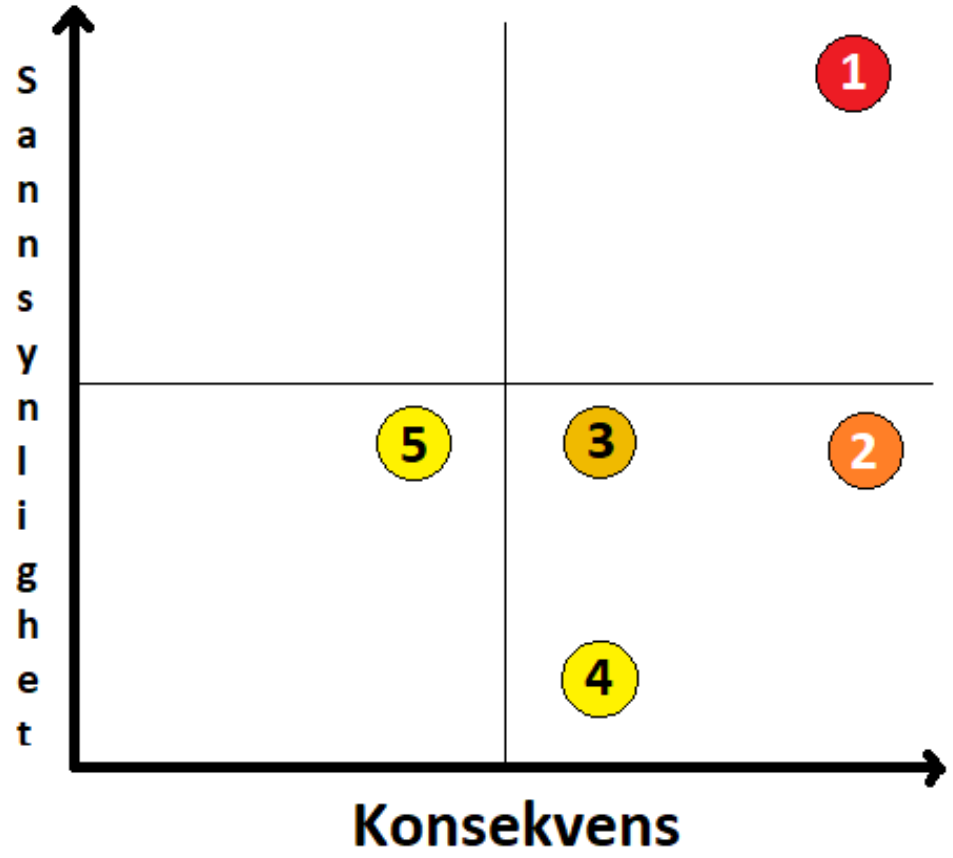
5. Risikovurdering

Enkelte deler av prosjektet, spesielt under utviklingsfasen - medbringer større risiko enn andre. Plattformen som vi skal utvikle skal inneholde og behandle sensitiv informasjon om Ateas kunder og samarbeidspartnere, og mye av denne informasjonen kan direkte påvirke kundenes sikkerhet hvis de skulle havne i feil hender. Det er dermed særdeles viktig at vår behandling av denne informasjonen er så sikker som mulig, både når det gjelder lagring, overføring og behandling. Vi må altså sørge for at hvert lag av plattformen egner seg til og er tilstrekkelig sikker.

1. **Sikkerhetsbrudd eller utilstrekkelig sikker behandling av kundedata.** Dette er et svært viktig punkt å vurdere risikoen for, da det i hovedsak er dette som er viktig for oppdragsgiver for at produktet skal kunne brukes av bedriften i ettertid. Det er derfor viktig at vi konstant er oppmerksom på hvor sikker plattformen er med tanke på lagring og overføring av kundedata. Tiltak her skal være å undersøke hva som er industristandard for behandling av data på ulike områder på plattformen, samt grundig testing og gjennomgang av sikkerheten generelt.
2. **Dårlig testing av programvare** kan føre til uforventede feil og mangler som kan være vanskelig å oppdage senere i prosjektet. Her skal vi blant annet bruke JUnit-tester for kildekoden og database, brukertester for interface og frontend - og eventuelle andre testmetoder for andre deler av prosjektet.
3. **For dårlig kommunikasjon med oppdragsgiver.** Dette kan blant annet føre til at produktet ikke fungerer slik som det var tiltenkt, tidkrevende justeringer for å rette opp feil og unødvendig bruk av midler på noe som ikke er brukbart.
4. **Å miste filer som er viktige for prosjektet** kan være kritisk. Hvis vi for eksempel mister deler av kildekoden eller viktig dokumentasjon kan det være både vanskelig og tidkrevende å gjenskape dem. For å unngå dette vil vi lagre alt av dokumentasjon både lokalt på egen pc og på en sikker plattform på internett.
5. **Sykdom/fravær.** Sykdom og fravær er spesielt fremtredende nå som korona-smitten er høy, og det er mye sykdom generelt. For å motkjempe at dette kan få stor effekt på prosjektet vil vi sørge for at alle medlemmer har satt seg godt inn i hver enkelt del av prosjektet, slik at man enkelt kan ta over deler som personen med fravær jobber med. Arbeidet skal også dokumenteres godt underveis, og kommunikasjon burde opprettholdes med den som er fraværende - slik at man effektivt kan fortsette å jobbe der det er behov.

Nr.	Hva kan gå galt	Hvor ofte	Konsekvensen	Klassifisering	Tiltak
1	Sikkerhetsbrudd	Svært ofte	Svært alvorlig	Særdeles alvorlig	Testing, undersøking og overvåkning
2	Dårlig testing av program	Sjelden	Svært alvorlig	Svært alvorlig	Testing av kode
3	For dårlig kommunikasjon med oppdragsgiver	Sjelden	Alvorlig	Alvorlig	Regelmessig kommunikasjon
4	Miste filer	Svært sjelden	Alvorlig	Mindre alvorlig	Dobbeltlagring
5	Sykdom/fravær	Sjelden	Mindre alvorlig	Mindre alvorlig	Overlappe arbeid

Figur 1: Tabell for risikoer og deres sannsynlighet og konsekvens

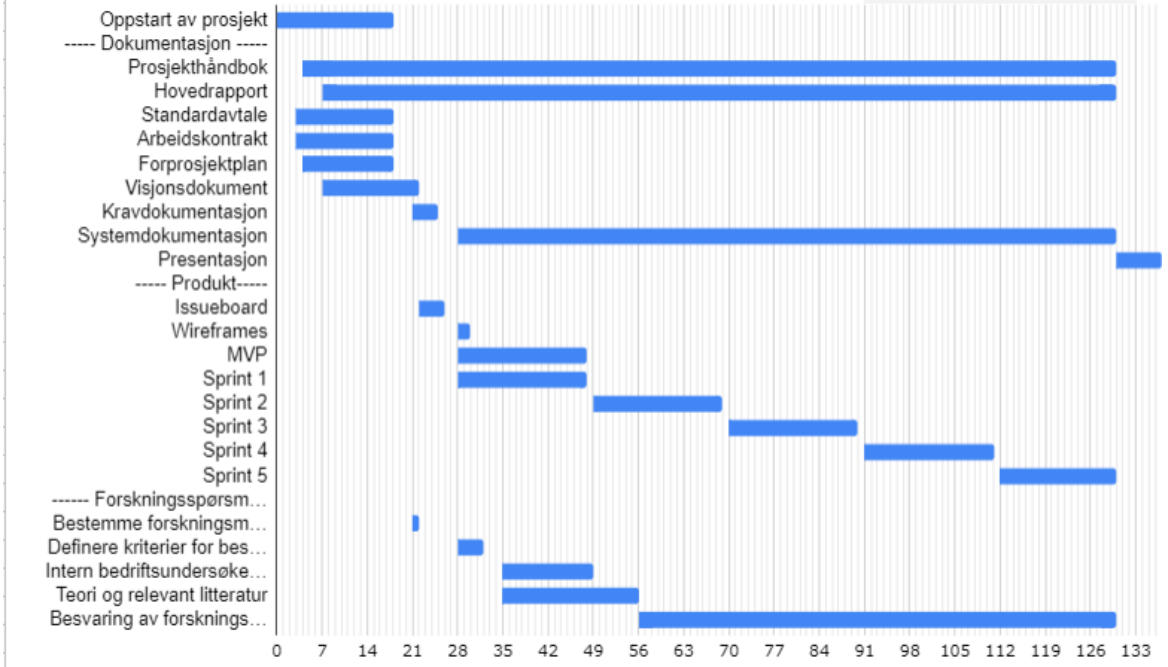


Figur 2: Diagram for sannsynlighet og konsekvenser over risikoer

6. Vedlegg

6.1 Tidsplan

Oppgaver	Framgang	Startdato	Frist	Startdag*	Varighet*
Oppstart av prosjekt	90%	10.01.2022	28.01.2022	0	18
----- Dokumentasjon -----					
Prosjekthåndbok	50%	14.01.2022	20.05.2022	4	126
Hovedrapport	5%	17.01.2022	20.05.2022	7	123
Standardavtale	100%	13.01.2022	28.01.2022	3	15
Arbeidskontrakt	100%	13.01.2022	28.01.2022	3	15
Forprosjektplan	100%	14.01.2022	28.01.2022	4	14
Visjonsdokument	95%	17.01.2022	01.02.2022	7	15
Kravdokumentasjon	0%	31.01.2022	04.02.2022	21	4
Systemdokumentasjon	0%	07.02.2022	20.05.2022	28	102
Presentasjon	0%	20.05.2022	27.05.2022	130	7
----- Produkt-----					
Issueboard	0%	01.02.2022	05.02.2022	22	4
Wireframes	0%	07.02.2022	09.02.2022	28	2
MVP	0%	07.02.2022	27.02.2022	28	20
Sprint 1	0%	07.02.2022	27.02.2022	28	20
Sprint 2	0%	28.02.2022	20.03.2022	49	20
Sprint 3	0%	21.03.2022	10.04.2022	70	20
Sprint 4	0%	11.04.2022	01.05.2022	91	20
Sprint 5	0%	02.05.2022	20.05.2022	112	18
------ Forskningsspørsmål -----					
Bestemme forskningsmetode	0%	31.01.2022	01.02.2022	21	1
Definere kriterier for besvarelse	0%	07.02.2022	11.02.2022	28	4
Intern bedriftsundersøkelse	0%	14.02.2022	28.02.2022	35	14
Teori og relevant litteratur	0%	14.02.2022	07.03.2022	35	21
Besvaring av forskningsspørsmål	0%	07.03.2022	20.05.2022	56	74



Vedlegg E: Gantt-diagram

6.2 Adresseliste

Adresselisten er i et eget vedlegg på grunn av sensitiv informasjon. Se vedlegg:

L Adresseliste-127

6.3 Avtaledokumenter

6.3.1 Arbeidskontrakt for bachelor-gruppen

Se vedlegg:

A Arbeidsavtale-127

6.3.2 Standardavtale

Se vedlegg:

B Standardavtale-127

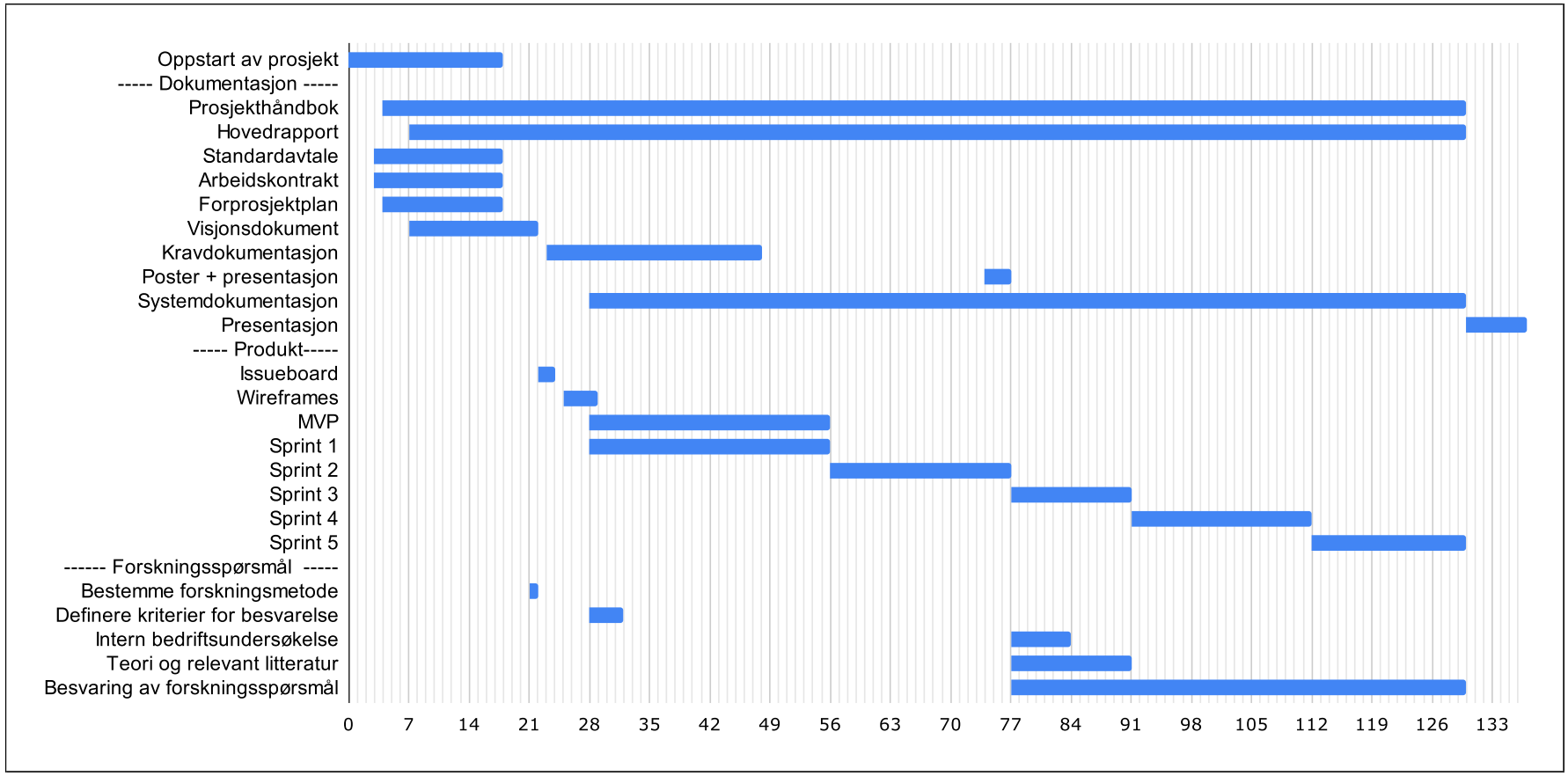
Incident Response System
E Gantt-Diagram

Endré Hadzalic
Odin Kvarving
Scott Rydberg Sonen

NTNU

Kunnskap for en bedre verden

Oppgaver	Framgang	Startdato	Frist
Oppstart av prosjekt	100%	10.01.2022	28.01.2022
----- Dokumentasjon -----			
Prosjekthåndbok	100%	14.01.2022	20.05.2022
Hovedrapport	100%	17.01.2022	20.05.2022
Standardavtale	100%	13.01.2022	28.01.2022
Arbeidskontrakt	100%	13.01.2022	28.01.2022
Forprosjektplan	100%	14.01.2022	28.01.2022
Visjonsdokument	100%	17.01.2022	01.02.2022
Kravdokumentasjon	100%	02.02.2022	27.02.2022
Poster + presentasjon	100%	25.03.2022	28.03.2022
Systemdokumentasjon	100%	07.02.2022	20.05.2022
Presentasjon	0%	20.05.2022	27.05.2022
----- Produkt-----			
Issueboard	100%	01.02.2022	03.02.2022
Wireframes	100%	04.02.2022	08.02.2022
MVP	100%	07.02.2022	07.03.2022
Sprint 1	100%	07.02.2022	07.03.2022
Sprint 2	100%	07.03.2022	28.03.2022
Sprint 3	100%	28.03.2022	11.04.2022
Sprint 4	100%	11.04.2022	02.05.2022
Sprint 5	100%	02.05.2022	20.05.2022
------ Forskningsspørsmål -----			
Bestemme forskningsmetode	100%	31.01.2022	01.02.2022
Definere kriterier for besvarelse	100%	07.02.2022	11.02.2022
Intern bedriftsundersøkelse	100%	28.03.2022	04.04.2022
Teori og relevant litteratur	100%	28.03.2022	11.04.2022
Besvaring av forskningsspørsmål	100%	28.03.2022	20.05.2022


Oppstart av prosjekt
----- Dokumentasjon -----
Prosjekthåndbok
Hovedrapport
Standardavtale
Arbeidskontrakt
Forprosjektplan
Visjonsdokument
Kravdokumentasjon
Poster + presentasjon
Systemdokumentasjon
Presentasjon
----- Produkt-----
Issueboard
Wireframes
MVP
Sprint 1
Sprint 2
Sprint 3
Sprint 4
Sprint 5
------ Forskningsspørsmål -----
Bestemme forskningsmetode
Definere kriterier for besvarelse
Intern bedriftsundersøkelse
Teori og relevant litteratur
Besvaring av forskningsspørsmål
0
7
14
21
28
35
42
49
56
63
70
77
84
91
98
105
112
119
126
133

Incident Response System
F Visjonsdokument

Versjon 1.3

Endré Hadzalic
Odin Kvarving
Scott Rydberg Sonen

NTNU

Kunnskap for en bedre verden

Innholdsfortegnelse

1. Innledning

Hensikten med dette dokumentet er å gi en oversikt over produktet som hovedoppgaven skal produsere. Her finner man blant annet informasjon om hvordan produktet er tiltenkt å fungere, hvilket formål det har og problemer det løser, samt funksjonelle og ikke-funksjonelle egenskaper til produktet.

2. Sammendrag problem og produkt

2.1 Problemsammendrag

Problem med	Innhenting/oppsamling og varsling av sikkerhetsoppdateringer i tilknytning programvarer/teknologier utstedt til kundene til Atea AS, som fortsatt er en manuell prosedyre.
Berører	Ateas Incident Response Team og kundene til Atea.
Som resultatet av dette	Trenger de en plattform som automatisk innhenter sikkerhetsoppdateringer for teknologier til Ateas kunder.
En vellykket løsning vil	Finne alle nye sikkerhetsfeil for programvarer/teknologier som er registrert på de forskjellige kundene til Atea, og automatisk sende et varsel til kundene som er berørt av disse.

2.2 Produktsammendrag

For	Ateas Incident Response Team.
Som	Har behov for et automatisk system for innhenting av sikkerhetsoppdateringer og varsling av kunder som blir berørt av disse.
Produktet AIRS (Automatic Incident Response System)	Er en (web) plattform
Som	Automatisk vil sende ut varsel til Ateas kunder dersom plattformen støter på en eller flere sikkerhetsfeil. En annen fordel er at kundene kan få et kjappere varsel, da et automatisk system effektiviserer denne prosessen.
I motsetning til	Den manuelle prosessen som Ateas Incident Response Team gjennomgår i dag for å innhente informasjon og sende ut varsel til kundene sine.
Har vårt produkt	Automatisk innhenting av nye sikkerhetsoppdateringer og varsling til kundene det gjelder.

3. Overordnet beskrivelse av interessenter og brukere

3.1 Oppsummering interessenter

Navn	Utdypende beskrivelse	Rolle under utviklingen
Atea Incident Response Team	Vegard Kjerstad, Leder for Atea Incident Response Team. Interessent fordi han er oppdragsgiver og har vært med på å definere oppgaven.	Veiledning angående industristandarder og generell innsikt i hvilke behov og ønsker Atea har i forhold til produktet som skal lages.
NTNU	Olav Skundberg, veileder for vår gruppe under utførelsen av oppgaven, og er derfor en interessent.	Generell veiledning gjennom utførelsen av prosjektet, og fungerer som mellomledd mellom NTNU og Atea.

3.2 Oppsummering brukere

Navn	Utdypende beskrivelse	Rolle under utviklingen	Representert av
Atea Incident Response Team	Atea Incident Response Team er i dag ansvarlig for manuell innhenting og varsling av sikkerhetsfeil i deres kunders teknologier.	Siden Atea er vår oppdragsgiver, er det fullt mulig at de vil gjennomføre penetrasjonstesting og slike tester av produktet vårt, for å sjekke om produktet har noen mangler og at alt er som det skal. Vegard Kjerstad vil komme med flere innspill underveis i utviklingen.	Vegard Kjerstad og eventuelle kollegaer.
Ateas kunder	Potensielle brukere som har tilgang til plattformen for å se informasjon fra Atea om sikkerhetsfeil i programvarer/teknologier de bruker.		Vegard Kjerstad

3.3 Brukermiljøet

Systemet må forenkle/passe inn i de daglige rutinene til Atea Incident Response Team. Dette gjør at det må være et intuitivt og raskt verktøy å bruke på en daglig basis. Et tregt system med lite gjennomtenkt brukergrensesnitt kan være frustrerende å bruke.

Systemet vil være enkelt tilgjengelig på Atea Incident Response Team sin arbeidsplass uavhengig om den er på kontoret eller hjemmefra, da systemet er en web applikasjon.

3.4 Sammendrag av brukernes behov

3.4.1 Innhenting

Behov	Prioritet	Påvirker	Dagens løsning	Foreslått løsning
Automatisk innhenting av sikkerhetsfeil.	Høy	Alle brukere	Manuell innhenting	Overvåkningssystem som følger med på oppdateringer om sikkerhetsfeil og analyserer disse for å finne ut hvilke programvarer/teknologier det gjelder.
Portal med oversikt over kunder og teknologier.	Høy	Alle brukere	Manuell oversikt, trolig i excel.	System med oversikt over kundene til Atea og deres teknologier.
Metode for å klassifisere hvor kritisk en sikkerhetsfeil er. (CVSS)	Middels	Alle brukere	Manuell klassifisering	Program som kan analysere og klassifisere hvor kritisk en sikkerhetsfeil er, og automatisk angi et CVSS-nivå og hvor raskt problemet bør håndteres.

3.4.2 Datahåndtering

Behov	Prioritet	Påvirker	Dagens løsning	Foreslått løsning
Sikker behandling av sensitiv data.	Høy	Alle brukere	Kryptering av data og sikker behandling/lagring	Stegvis kryptering gjennom alle lag i webapplikasjonen (frontend, backend & database). Kombinasjon av symmetrisk og asymmetrisk kryptering med passende krypteringsalgoritmer.

Atea skal ha mulighet til å logge seg inn med admin-privilegier.	Høy	Atea	Atea har tilgang til sitt eget manuelle system	Forbedre det manuelle systemet ved å lage en automatisk web applikasjon hvor Atea kan logge seg inn og ha tilgang til alle kunders teknologier samt deres sikkerhetsfeil.
Mulighet for å logge seg inn med kunde-privilegier for kundene til Atea.	Lav	Kundene til Atea og Atea	N/A	Innlogging på med lavere privilegier systemet for kunder.

3.4.3 Varsling

Behov	Prioritet	Påvirker	Dagens løsning	Foreslått løsning
Automatisk varsel ved sikkerhetsfeil. (7-8 CVSS-score - varsling internt)	Høy	Alle brukere	Manuelt varsel	System for utsending av automatisk varsel ved nyoppdagede sikkerhetsfeil i teknologier hos kundene til Atea.
Varsling av Admin/Atea Incident Response Team.	Middels	Admin/Atea Incident Response Team	N/A	Ved potensielt kritiske sikkerhetsfeil eller lignende så varsles Atea Incident Response Team/Admin. Kravet til å få varsel kan spesifiseres av brukere av systemet.
Godkjennelse av en administrator før utsendelse av varsler	Høy	Alle brukere	N/A	En innlogget administrator må godkjenne varsling av kunder før varslene faktisk blir sendt ut.
Ukentlig rapport over oppdagede sikkerhetsfeil av systemet.	Lav	Atea	N/A	Ukentlig oversikt over de forskjellige sikkerhetsfeilene som ble oppdaget av systemet den foregående uken/dagen.

3.5 Alternativer til vårt produkt

- Manuell oversikt av kunder i f.eks. excel
- Manuell oversikt av sikkerhetsfeil i sikker database

4. Produktoversikt

4.1 Produktets rolle i brukermiljøet

Produktet skal være et verktøy som Atea sitt Incident Response Team kan bruke for å forenkle prosessen ved å finne nye sikkerhetsfeil og varsle kundene sine.

4.2 Forutsetninger og avhengigheter

En forutsetning er at Atea faktisk har bruk for dette produktet, og at interessenter ikke endrer seg underveis i utviklingen. Vi som utviklere er avhengige av at Atea er oppdragsgiver under utviklingen. Dersom oppdragsgiver endrer seg, vil vi som utviklere være nødt til å forholde oss til denne endringen samt endre på all dokumentasjon vi har skrevet så langt.

En annen forutsetning er at data om sikkerhetsfeil er enkelt tilgjengelig for tolking og kan samles inn over internett. Dersom dette ikke er mulig, vil vi støte på flere utfordringer underveis, da vi i utgangspunktet skal utvikle en web applikasjon.

En tredje forutsetning er at vi må følge WCAG 2.1 standard for universelt interaksjonsdesign.

5. Produktets funksjonelle egenskaper

Funksjonell egenskap	Beskrivelse
Overvåkningsprogram for innhenting av nye sikkerhetsfeil	Webapplikasjons-løsning som analyserer en rekke websider/feeder/annet et par ganger i løpet av døgnet, som automatisk kan oppdage nye innlegg om sikkerhetsfeil. Den skal også kunne klassifisere hvilke programvarer/teknologier innlegget gjelder, og knytte dette innlegget opp mot alle kunder som er registrert med denne programvaren/teknologien.
Varsling av kunder ved oppdaging av sikkerhetsfeil	Varsling av kunder som bruker en spesifikk teknologi fra Atea. Denne varslingen kan eksempelvis skje innad i den gitte teknologien, eller ved e-post.
Oversikt over Ateas kunder og deres teknologier samt eventuelle sikkerhetsfeil i gitte teknologier	En database og tilhørende web-portal med oversikt over Ateas kunder og deres programvarer som er utstedt fra Atea. Siden skal gi en enkel oversikt over hvilke kunder som bruker hvilken programvare, eller oversikt over de ulike programvarene en kunde bruker.
Metode for å klassifisere hvor kritisk en sikkerhetsfeil er	Et program som kan analysere informasjonen på et innlegg som er innhentet av overvåkningsprogrammet, og automatisk angi hvor kritisk sikkerhetsfeilen er. Den skal deretter kunne angi et nivå på hvor raskt man bør behandle problemet.
Kategorier for sikkerhetsfeil	Forskjellige kategorier som gjør at sikkerhetsfeil kan kategoriseres og filtreres.
Filterfunksjon for kunder	Filtrering av Ateas kunder, som gir en oversikt over hvilke teknologier hver kunde har.
Filterfunksjon for teknologi	Filtrering av kunders teknologier. Her kan vi få en oversikt over kunder som bruker samme type teknologi.
Sortering av kunder	Stigende/synkende sortering av kunder. Sortering alfabetisk, hvor utsatte de er for sikkerhetsfeil, og hvor mange sikkerhetsfeil de har.
Sortering av teknologier	Stigende/synkende sortering av teknologier. Sortering alfabetisk og for hvor mange sikkerhetsfeil hver teknologi har.
Søkefunksjon på siden	Mulighet for å søke etter teknologier/kunder på siden. Søking etter forskjellige kategorier for sikkerhetsfeil.
Høynivå statistikk.	Statistikk over hvor mange sikkerhetsbrudd som har forekommet hos alle teknologiene til Atea, eller hos en spesifikk kunde over et bestemt tidsintervall, eller hos en kategori sikkerhetsfeil.

Lavnivå statistikk	Statistikk over hvor mange sikkerhetsbrudd som har forekommet hos en spesifikk teknologi over et bestemt tidsintervall
Interface for å legge til nye kunder med programvarer/teknologier	En måte å registrere nye kunder inne på webapplikasjonen hvor man kan legge til hvilke programvarer/teknologier de bruker
Side for innlogging	En side for å autorisere brukere til bruk av systemet.
Atea skal ha mulighet til å logge inn som administrator	Dersom en bruker med admin-privilegier logger seg på, skal brukeren ha tilgang til alle funksjonaliteter og kan endre på data. Dette må gjøres på en sikker måte, slik at en angriper ikke kan tildele seg selv en admin-attributt.
Utloggingsknapp	En knapp som logger ut autoriserte brukere fra systemet.
Forskjellige privilegier for forskjellige brukere	En innlogget bruker med høyere privilegier har typisk tilgang til mer sensitiv data og kan utføre mer "skadelige" operasjoner.
Mulighet for å endre kunde og deres teknologier	Interface koblet opp mot databasen for å endre kunders navn, id, teknologer, og andre eventuelle attributter. Man må ha admin-privilegier for å gjøre slike endringer.
Mulighet for å slette kunde	Interface koblet opp mot databasen for å slette kunder. Man må ha admin-privilegier for å gjøre en slik handling.
Mulighet for valg av varslingsmetode	Interface for å velge hvilken form for varsling en spesifikk kunde skal motta (email, tekstmelding…).
Oversikt over nyoppdagede sikkerhetsfeil.	En oversikt over nye sikkerhetsfeil oppdaget av systemet.
Mulighet for en innlogget administrator fra Atea til å godkjenne/forkaste/endre nyoppdagede sikkerhetsfeil fra systemet før det settes inn i systemet.	Mulighet for å se over nye sikkerhetsfeil systemet har fanget opp i oversikten for så å godkjenne/endre/forkaste de nytilkomne forslagene av en Atea-bruker/admin. Dette gjøres for de "trer i kraft".
Grundig logging av viktige deler av plattformen	Logging gjør at det er enklere å oppdage feil på plattformen. Viktige funksjoner og egenskaper skal derfor logges i både frontend og backend, slik at det er enkelt å lokalisere hvor feilen kommer fra. Den skal også kunne brukes til å overvåke bruken av plattformen, for å identifisere eventuelle sikkerhetsangrep og identifisere uvanlige hendelser.

6. Ikke-funksjonelle egenskaper og andre krav

Ikke-Funksjonell egenskap	Beskrivelse
Universelt design iht. WCAG 2.1 standarden	Layouten til systemet er utformet iht WCAG 2.1
Behandling personvernopplysninger	I henhold til reglementet beskrevet av Datatilsynet
Prosjektmøter	Møter med veileder/oppdragsgiver foregår over Teams.
Kommunikasjon i gruppa (over nett)	Discord er den foretrukne plattformen blant gruppemedlemmene.
Generell "web-service security" i systemet er **minimum** iht. OWASP sine retningslinjer for "Web Service Security"	Generelle tiltak for å forhindre de vanligste sikkerhetsproblemene i industrien (OWASP)
Sikker kommunikasjon over internett skjer ved hjelp av HSTS protokollen.	Systemet må oppfylle spesifiserte krav

7. Referanser

- OWASP Cheat Sheet Series. (2021, 26. juli). Web Service Security. Hentet fra: https://cheatsheetseries.owasp.org/cheatsheets/Web_Service_Security_Cheat_Sheet.html
- OWASP Cheat Sheet Series. (2020, 3. juni) HTTP Strict Transport Security Cheat Sheet. Hentet fra: https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html
- W3C. (2018, 5. juni). Web Content Accessibility Guidelines (WCAG) 2.1. Hentet fra: https://www.w3.org/TR/WCAG21/

Incident Response System
H Systemdokumentasjon

Versjon 1.2

Endré Hadzalic
Odin Kvarving
Scott Rydberg Sonen

NTNU

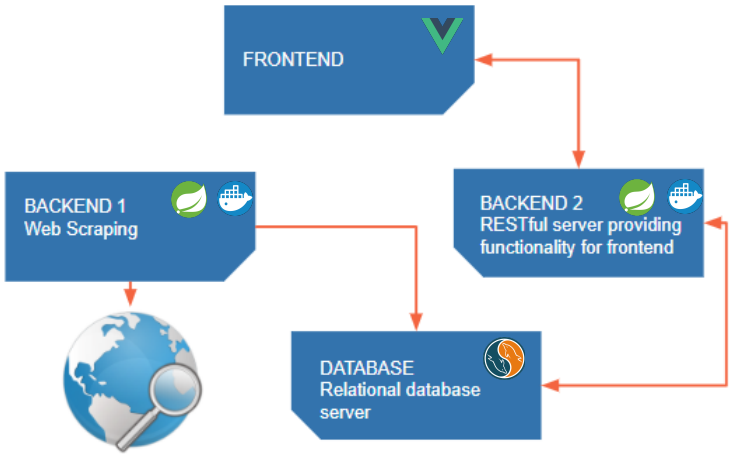
Kunnskap for en bedre verden

Innholdsfortegnelse

1. Introduksjon

Dette dokumentet er skrevet for å beskrive produktet vårt AIRS (Atea Incident Response System) og hvordan det er utviklet. I dette dokumentet vil vi først beskrive systemarkitekturen, altså oppbygningen av systemet. Deretter viser vi prosjektstruktur, klassediagram og databasemodell. Videre skal vi presentere server-tjenester og sikkerhetsmetoder vi har brukt. Mot slutten vil vi beskrive installasjon og kjøring av systemet, dokumentasjon av kildekode, og kontinuerlig integrasjon og testing.

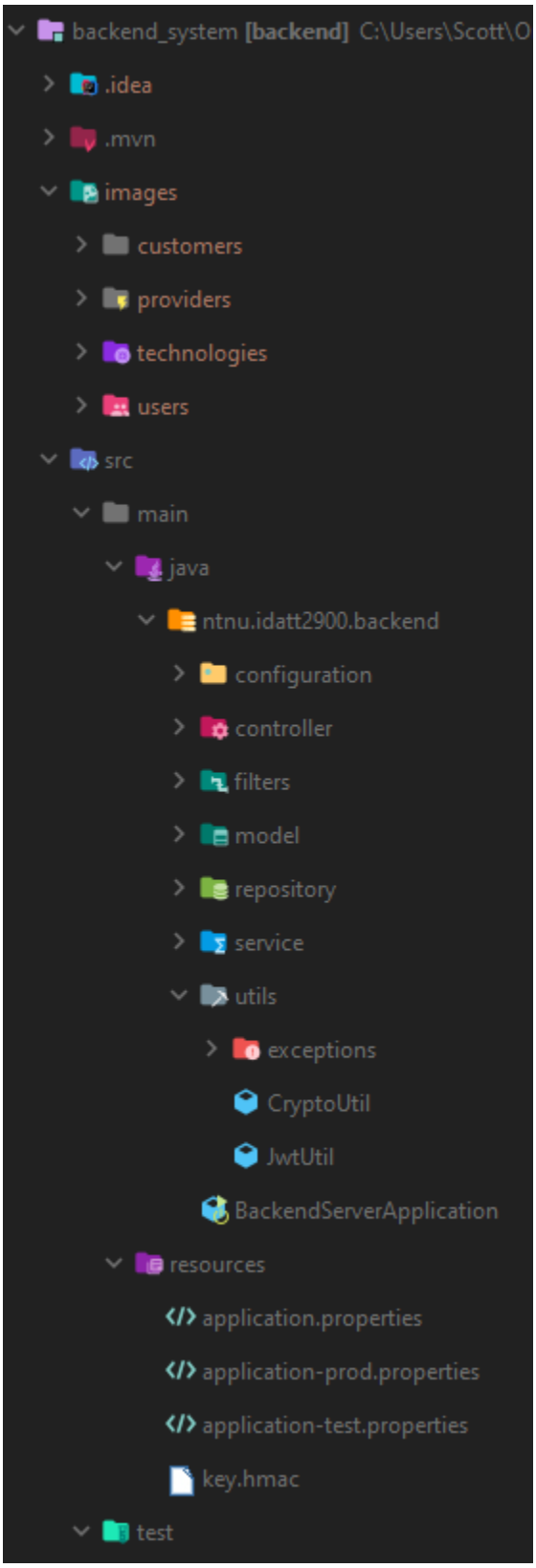
2. Arkitektur

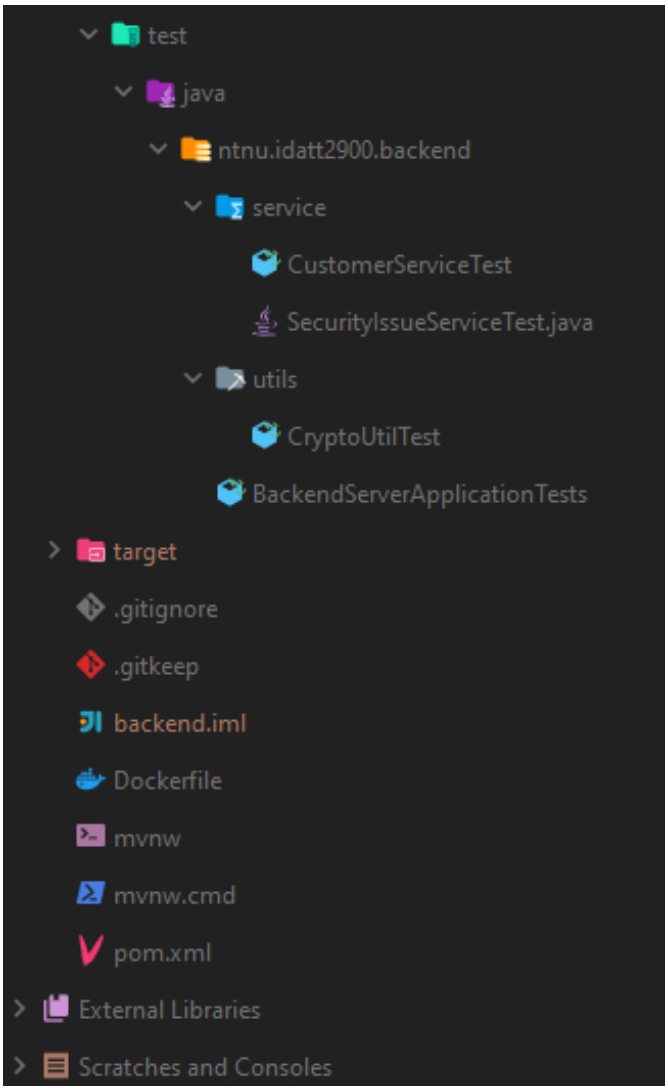


Figur 2.1: En oversikt over systemets arkitektur.

I figur 2.1 kan man se systemets arkitektur og hvordan det er delt opp i en frontend-del, to backend-deler og en database. Frontenden står for det visuelle, altså det brukeren ser. Backend 1 håndterer web-scraping, i form av at den henter inn sikkerhetsfeil fra ulike innhentingskilder som for eksempel vanlige nettsider, RSS- eller Atom-feeds. Backend 2 er i direkte kontakt med frontend og er derfor et bindeledd mellom frontend og databasen. Databasen lagrer alle viktige entiteter som skal brukes i frontend. Sammen danner de vårt produkt.

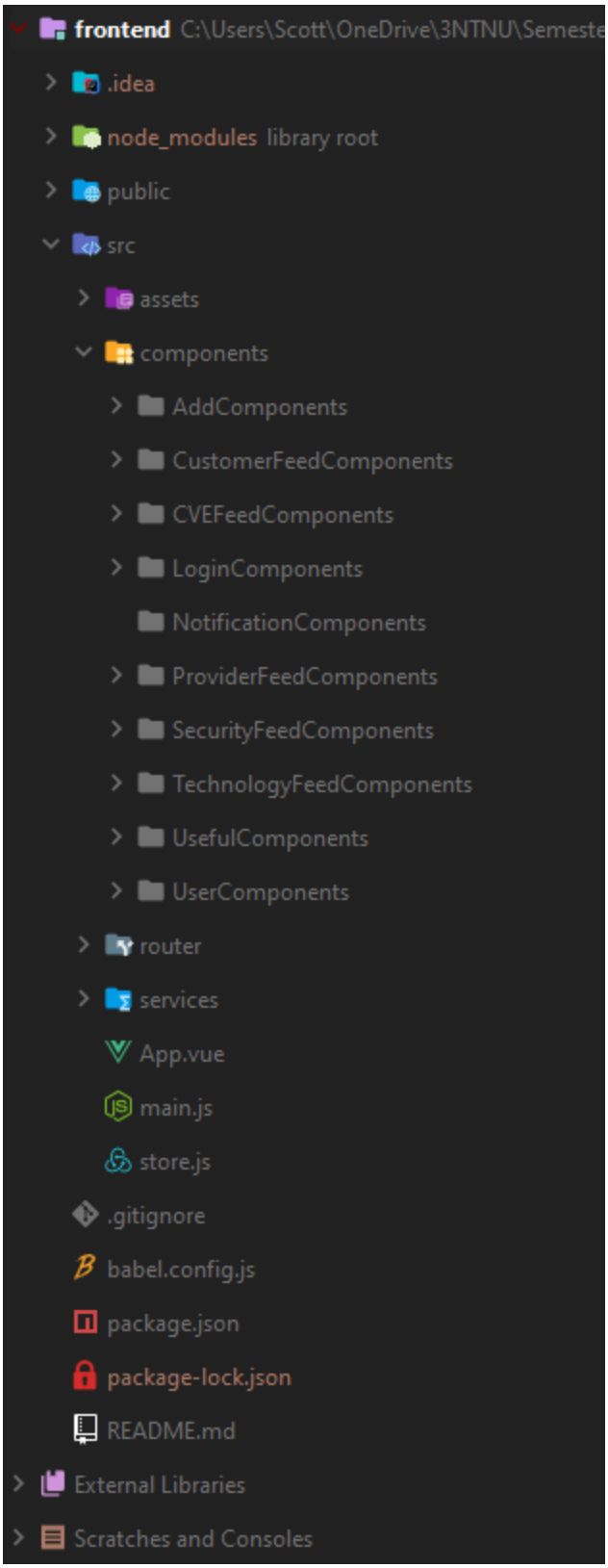
3. Prosjektstruktur





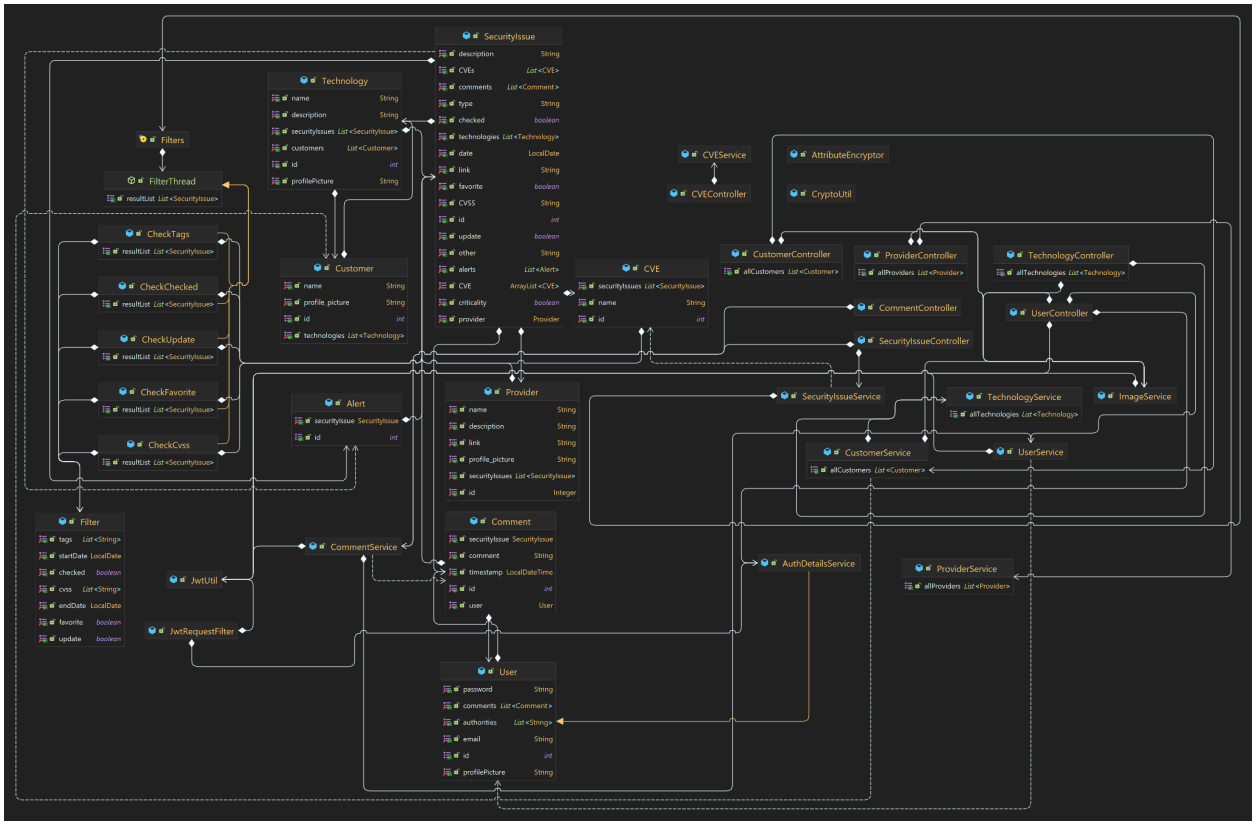
Figur 3.1 og 3.2: Bildene viser filstrukturen i system-backenden, altså backend 2. Begge bildene tilhører samme filsystem, og inneholder relevante klasser. Her finner man blant annet kontroller-, service- og modell-klasser. "images"-mappen inneholder alle profilbilder som brukes i applikasjonen. Videre har man mappene som blant annet inneholder konfigurasjoner og kontroller-klasser. I "resources"-mappen ligger filene for application properties. Disse kan man bytte mellom, avhengig av hvilket kjøremiljø man vil bruke.

Figur 3.3 og 3.4: Bildene viser filstrukturen til web-scraper-backenden. Disse bildene tilhører nok en gang samme filsystem, og inneholder klasser som brukes i web-scraperen. Mappen med navn "configuration" inneholder alle klasser som definerer ulike konfigurasjoner. "database"-mappen inneholder klasser og interfaces som brukes for å legge til nye elementer i databasen fra innhentet informasjon. I "sources"-mappen lagres alle feeder og nettsider som skal scrapes. "system"-mappen har viktige klasser for kjøring av web-scraperen. I likhet med system-backenden har man også lagret flere application properties i "resources"-mappen, for endring av kjøremiljø.

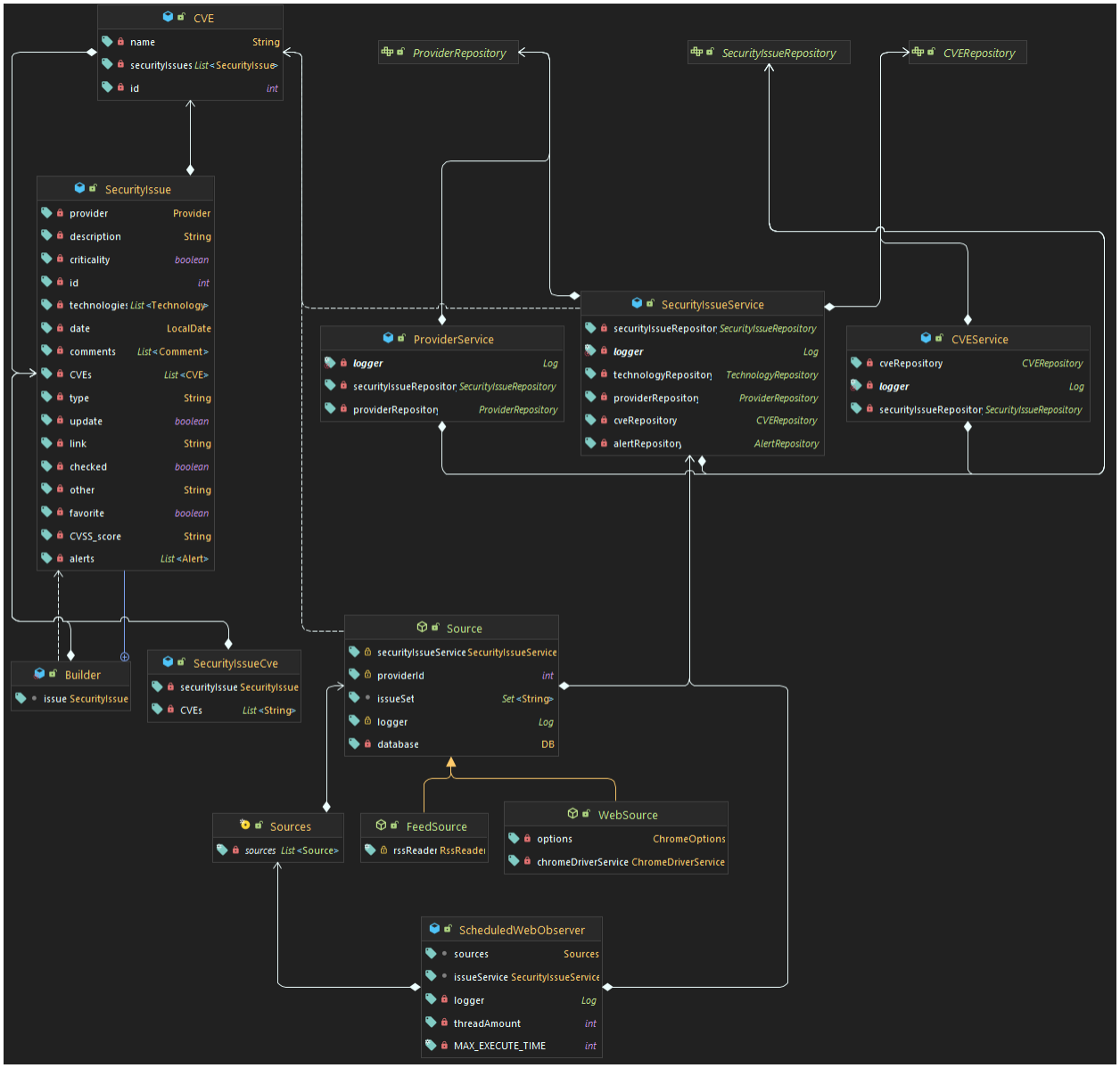


Figur 3.5: Bildet til høyre viser en oversikt over filstrukturen til frontenddelen. I "assets"-mappen ligger statiske bilder som brukes på nettsiden. "components"-mappen inneholder alle komponenter, og mappen er delt inn i flere mapper for logisk inndeling og oversikt. "router"-mappen holder alle ruter, altså en parent-komponent som er en rute eller et endepunkt i frontend. I tillegg ligger det en fil som har alle mulige ruter. "services"-mappen inneholder alle service-filer for frontend, altså filene som gjør tjenerkall til backend.

4. Klassediagram

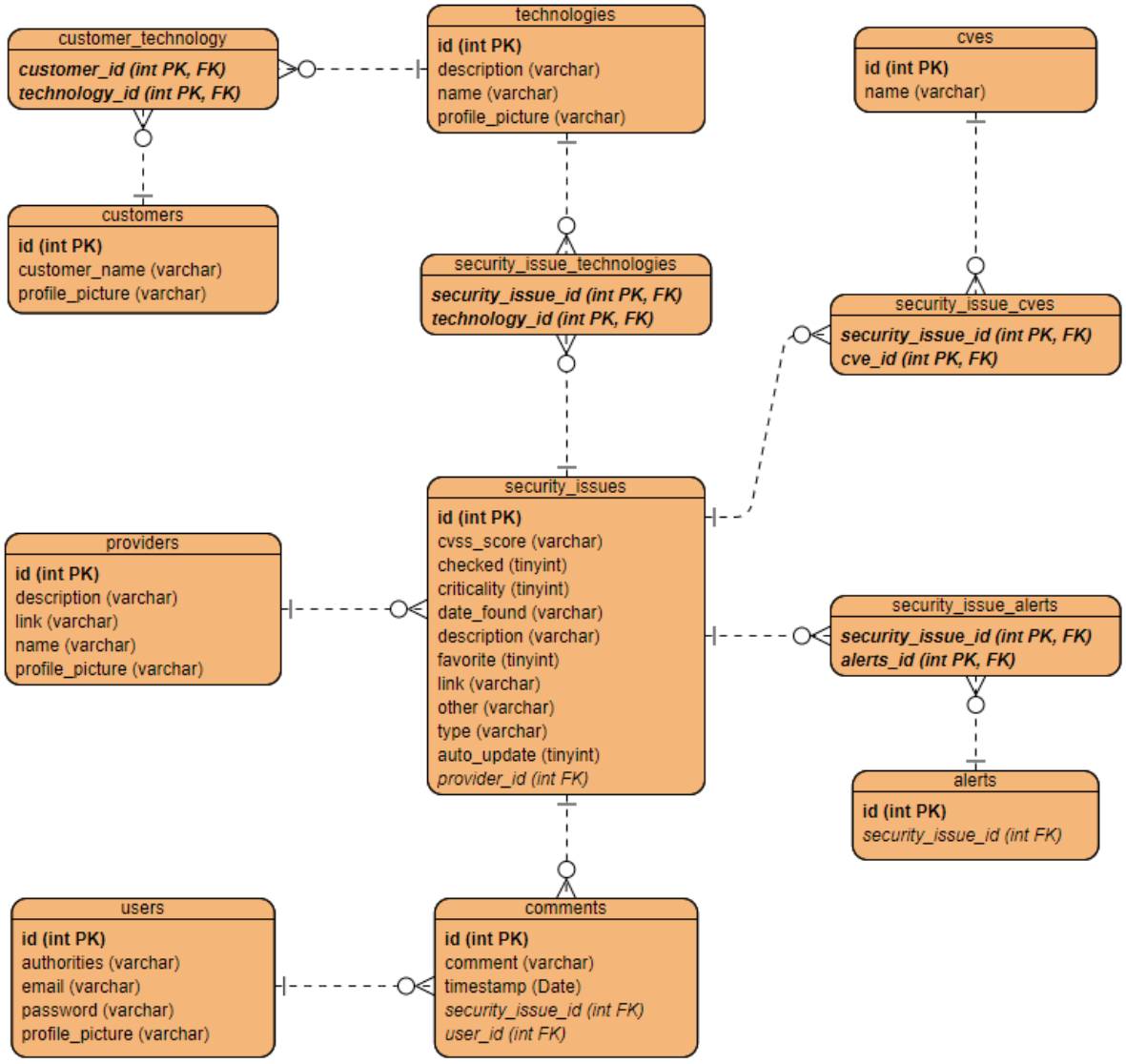


Figur 4.1: Et klassediagram over system-backenden, som gir oversikt over relevante klasser og komponenter fra både backend 2 (system-backend).



Figur 4.2: Et klassediagram over web-scraperen, hvor de mest relevante klassene knyttet til web-scraperen er visualisert.

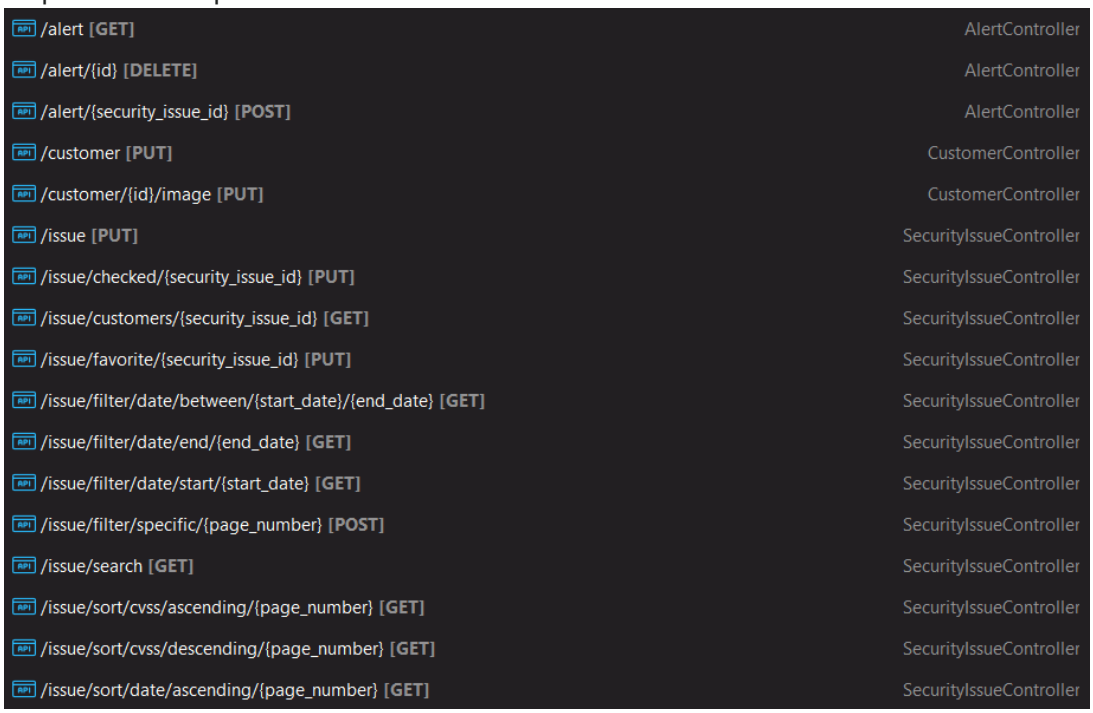
5. Databasemodell



Figur 5.1: Databasemodell med oversikt over alle tabeller i relasjonsdatabasen vår. Det finnes fire mange-til-mange-relasjoner i databasen. Disse er tatt hånd om ved hjelp av relasjonstabeller med primær- og fremmednøkkel som ID-ene fra begge relaterte tabeller.

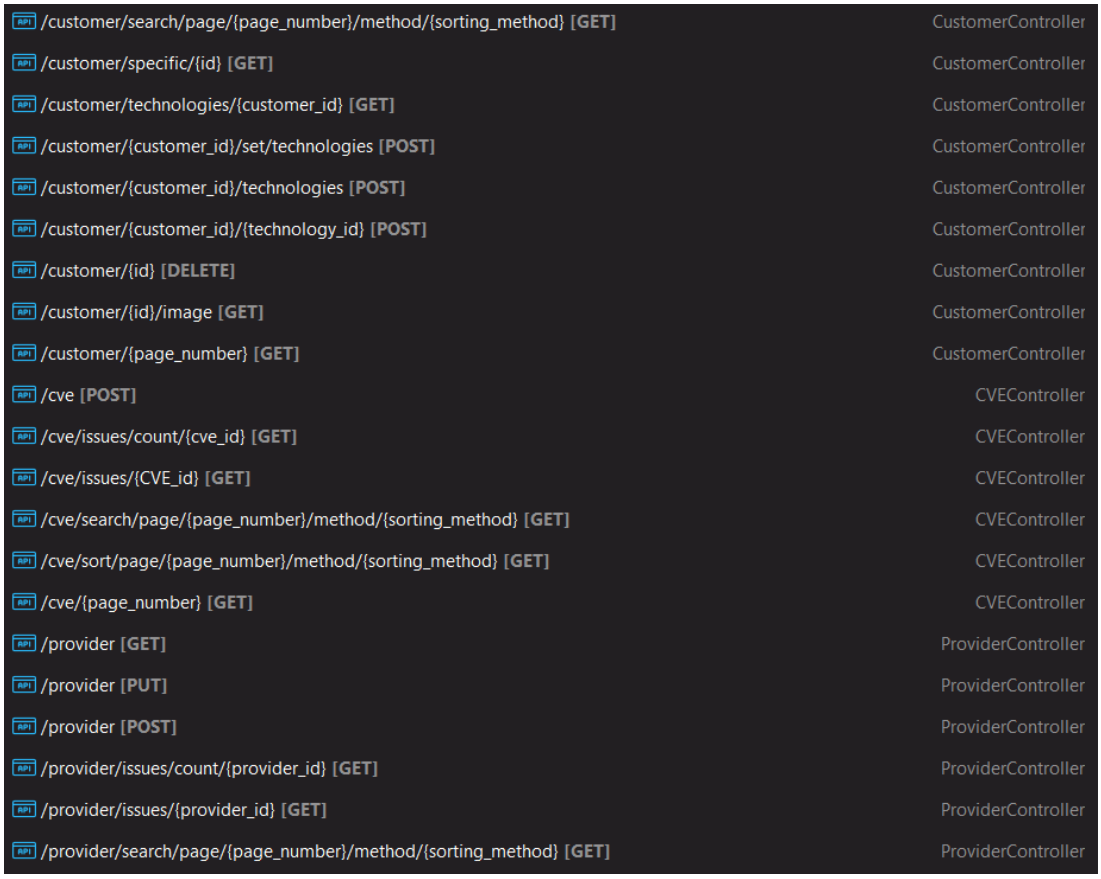
6. Server-tjenester

Nedenfor listes alle REST-endepunkter som er tilgjengelig i systemet opp. For å få en forklaring på de ulike endepunktene, se punkt 9. Dokumentasjon av kildekode for å få en instruksjon på hvordan man genererer JavaDoc, og naviger til ntnu.idatt2900.backend.controller for å få en oversikt over alle controller-klassene og deres respektive endepunkter.



/issue/technologies/{security_issue_id} [GET]	SecurityIssueController
/issue/update/{security_issue_id} [PUT]	SecurityIssueController
/issue/{id} [DELETE]	SecurityIssueController
/issue/{id}/comments [GET]	SecurityIssueController
/issue/{id}/comments [POST]	SecurityIssueController
/issue/{id}/data-amount [GET]	SecurityIssueController
/issue/{page_number} [GET]	SecurityIssueController
/issue/{security_issue_id}/cves [GET]	SecurityIssueController
/issue/{security_issue_id}/set/technologies [POST]	SecurityIssueController
/issue/{security_issue_id}/technology/{technology_id} [POST]	SecurityIssueController
/issue/{security_issue_id}/technology/{technology_id}/remove [PUT]	SecurityIssueController
/technology [GET]	TechnologyController
/technology [PUT]	TechnologyController
/technology [POST]	TechnologyController
/technology/issues/{technology_id} [GET]	TechnologyController
/technology/search/page/{page_number}/method/{sorting_method} [GET]	TechnologyController
/technology/specific/{id} [GET]	TechnologyController
/technology/{id} [DELETE]	TechnologyController

/technology/{id}/image [GET]	TechnologyController
/technology/{id}/image [PUT]	TechnologyController
/technology/{page_number} [GET]	TechnologyController
/technology/{technology_id}/customers [GET]	TechnologyController
/technology/{technology_id}/remove/customers [POST]	TechnologyController
/technology/{technology_id}/remove/{customer_id} [PUT]	TechnologyController
/technology/{technology_id}/set/customers [POST]	TechnologyController
/technology/{technology_id}/{customer_id} [POST]	TechnologyController
/user [POST]	UserController
/user/authenticate [POST]	UserController
/user/image [PUT]	UserController
/user/info [GET]	UserController
/user/is-admin [GET]	UserController
/user/password [PUT]	UserController
/user/promote [PUT]	UserController
/comments/{id} [DELETE]	CommentController
/customer [GET]	CustomerController
/customer [POST]	CustomerController



Figur 6.1, 6.2, 6.3 og 6.4: Bildene viser en oversikt over alle REST-endepunkter som finnes i systemet. Disse endepunktene gir tilgang til alle REST-ressursene i applikasjonen.

7. Sikkerhet

7.1 Hashing og håndtering av passord

Hashing av passord foregår ved bruk av bcrypt. I hovedrapporten har vi forklart hva bcrypt er: "Bcrypt er en hashing-algoritme som gjør det ekstremt kostbart å brute-force passord, og har gjort det i praksis nesten umulig. Denne kalles ofte for en CPU-tung algoritme, fordi den krever flere CPU-sykluser for å kalkulere én enkelt hash. Grunnen til dette er at man definerer en arbeidsfaktor, som avgjør hvor mange iterasjoner av algoritmen passordet skal hashes med. Her må man finne balansen mellom sikkerhet og ytelse, siden en for høy work factor gjør at verifisering av passord tar for lang tid, mens en for lav work factor reduserer sikkerheten.

Når man hasher et passord med bcrypt sender man inn et passord som er maksimalt 72 bytes langt, en numerisk kostnad og en 16-bytes (128-bit) salt-verdi. Saltet er typisk tilfeldig generert. Bcrypt-funksjonen bruker disse parameterne til å beregne en 24-byte (192-bit) hash. For å autentisere et passord som er hashet med bcrypt henter man ut saltet til det registrerte passordet. Hvert hashet passord registrert i databasen lagres med saltet som en del av tekststrengen. Dette saltet blir deretter brukt under hashing av passordet som skal autentiseres, og sammenlignes deretter med hashen til det registrerte passordet."

(Sonen et al., 2022) (*Bcrypt*, 2022).

7.2 Kryptering og behandling av annen sensitiv kundeinformasjon

I hovedrapporten har vi beskrevet hvordan vi krypterer og behandler sensitiv kundeinformasjon: "Vi har kryptert all informasjon om kundene i databasen, slik at man ikke direkte kan lese ut informasjon om kundene hvis man skulle få tilgang til databasen. For å kryptere og dekryptere kundeinformasjonen valgte vi å bruke JPA AttributeConverter, som konverterer en primitiv datatype i Java til en spesifisert kolonnetype i databasen og tilbake igjen til en valgt primitiv datatype. Som krypteringsalgoritme brukte vi AES, som er en av de sikreste og mest brukte krypteringsalgoritmene for slik type data. I tråd med Datatilsynets anbefalinger bruker vi en 128-bit nøkkel for å kryptere dataen. Denne nøkkelen blir hentet fra en kryptert fil, hvor vi har spesifisert at den skal bruke kun de 128 første bitsene fra filen som nøkkel. Den krypterte filen som nøkkelen hentes fra ligger lagret separat fra applikasjonen. Her blir både kundens navn og profilbilde-lenke kryptert, slik at man ikke kan hente ut informasjonen til en kunde i klartekst. I relasjons-tabellen mellom en kunde og en teknologi i databasen er de kun knyttet sammen av en unik ID, og man kan dermed ikke knytte en kunde opp mot en teknologi siden resten av attributtene for kunden er kryptert. Vi har også brukt samme metode for å kryptere brukere av systemets e-postadresser, slik at man i praksis ikke har tilgang til innloggingsinformasjonen til en bruker uten å dekryptere innholdet i databasen." (Sonen et al., 2022).

7.3 Bruk av tokens

I systemet bruker vi tokens for å autentisere brukere etter de har logget inn. Hvis bruker skriver inn riktig innloggingsinformasjon og får logget inn, vil de motta en JWT fra backend. Dette tokenet kan brukes for å autentisere bruker før hvert tjenerkall. Dersom det ikke finnes en JWT i frontend, vil man ikke klare å kalle metoder i backend. JWT-en lagres i sessionStorage, som fungerer som nettsidens minne, og blir slettet når session er ferdig (utlogging). Ved bruk av JWT, kan man også vite om brukeren er en admin eller ikke. Dermed kan man også gi begrenset tilgang til brukere, basert på hvilken rolle de har.

En mer forklarende beskrivelse av JWT er skrevet i hovedrapporten: "Måten en JWT-token blir tildelt på, er ved at en bruker logger seg inn med riktig påloggingsinformasjon, som blir verifisert ved å sammenligne verdiene til en allerede registrert bruker. Hvis autentiseringen går gjennom, tildeles man en unik JWT-token. Grunnen til at denne må være unik er for at ingen brukere skal ha samme JWT-token samtidig, da det er denne som bestemmer hvilke deler av nettsiden man har tilgang til. Får man for eksempel tildelt samme JWT-token som en administrator som bruker systemet, har man plutselig tilgang til alle deler av systemet som en administrator har.

Gjennom sikker oppretting av en JWT-token vil man unngå slike problemer, og sørger for at brukerens JWT-token sjekkes opp mot registrerte privilegier hver gang den utfører et kall til serveren. Dersom en angriper prøver å navigere til en side i applikasjonen gjennom et kjent endepunkt, vil ikke angriperen bli sendt til denne siden før vedkommende har autentisert

seg med en JWT-token, og sjekket at brukeren har privilegier til å gå til denne siden. JWT-token er derfor viktig for å skape sikker brukerautentisering, både for å verifisere identiteten til brukeren og skjerme nettstedet fra potensielle angripere.” (Sonen et al., 2022).

7.4 Sikker overføring av data i HTTPS

Mot slutten av utviklingsprosessen fikk vi kjørt alle deler av systemet i ulike skytjenester. Disse tjenestene tok i bruk HTTPS-protokollen for å kommunisere med de andre delene av systemet. På denne måten har vi fått implementert ekstra sikkerhet, gjennom TLS/SSL i HTTPS-protokollen.

På grunn av sikkerhetsmessige grunner, ba oppdragsgiveren vår oss om å fjerne systemet fra skytjenestene. Systemet bruker derfor ikke HTTPS ved innleveringsøyeblikket, men det er fullt mulig å implementere det på nytt.

8. Installasjon og kjøring

8.1 Installering av repoet

Her er lenken til GitLab-repoet vårt: GitLab repo

For å installere repoet bør man ha lastet ned Git Bash (guide på hvordan man laster ned Git Bash: Guide)

Når man har lastet ned og åpnet Git Bash, vil man kunne klare å klone GitLab-repoet vårt ved å kjøre disse kommandoene (NB: Det er mulig at det ikke er mulig å klone, dersom man ikke er medlem av GitLab-repoet. Derfor blir kildekoden vedlagt ved innlevering):

- Først, velg hvilken mappe du vil bruke for å klone repoet.
- Deretter skriver man: git clone https://oauth2:PERSONAL_ACCESS_TOKEN@gitlab.stud.idi.ntnu.no/scottrs/idatt2900-bachelorgruppe-127.git
    - PERSONAL_ACCESS_TOKEN er en token man genererer i GitLab:
        - Gå inn på brukermenyen øverst til høyre.
        - Trykk på "Edit Profile" i nedtrekksmenyen.
        - Velg "Access Token" i menyen til venstre.
        - Skriv inn navn på token under "Token name".
        - *Frivillig: Velg "Expiration date", altså datoen som tokenet skal slettes.
        - Velg scopes under "Select scopes" for å velge rettigheter.
        - Trykk "Create personal access token" for å generere en personlig access token.
        - Access tokenet vil vises øverst under "Your new personal access token"
            - Tokenet begynner med "glpat-".
            - Lagre tokenet for senere bruk. Tokenet slettes dersom man refresher!

8.2 Installering av web-scraper

Se README for backend_obtain etter installering av repoet. Denne filen finner du her i mappen som ble installert:
idatt2900-bachelorgruppe-127/backend_obtain/README.md

8.3 Installering av system-backend

Se README for backend_system etter installering av repoet. Denne filen finner du her i mappen som ble installert:
idatt2900-bachelorgruppe-127/backend_system/README.md

8.4 Installering av frontend og kjøring av systemet

8.4.1 Avhengigheter/biblioteker

Node.js:
Node.js er en hendelsesdrevet, kryssplattform runtime-system som er designet for å bygge skalerbare web-applikasjoner. Node.js baserer seg på JavaScript og er asynkron. Dette betyr at Node.js ikke venter på at en operasjon skal fullføres ved tjenerkall, men heller

kjører- neste operasjon. Et viktig punkt her er at en callback-funksjon kalles når denne operasjonen er ferdig. På denne måten kan data fra en tjener håndteres når som helst i frontend-delen av applikasjonen (Node.js, n.d.).

8.4.2 Installering av frontend

Dersom gitlab-repoet er klonet, kan man åpne terminalvinduet (for Node.js) når man er inne i frontend-mappen. **NB: system-backenden må kjøres før frontend for å få brukt koblingen.** Deretter er det noen kommandoer som må kjøres for å åpne applikasjonen:

- **npm install**
  - Denne kommandoen gjør at man automatisk laster ned alle nødvendige pakker og avhengigheter som er brukt i frontend for web-applikasjonen.
- **npm run serve**
  - Kommandoen gjør at koden kompileres og åpner applikasjonen på en port i localhost.

Det finnes også noen andre kommandoer man kan bruke:

- npm run build
  - Kommandoen lager en build-mappe som inneholder en bygget versjon av applikasjonen. Kjøres som regel før npm run serve.
- npm run lint
  - Denne kommandoen kjører et program som analyserer om koden har feil eller mangler.

Basert på hvordan database-tilkoblingen er oppsatt må man gjøre endringer i frontend-kildekoden. Her er det én enkelt linje med kode som må erstattes - basert på hvilken URL backend-system bruker:

- Åpne frontend-mappen i et koderedigeringsprogram (for eksempel Visual Studio Code)
- Naviger deg til denne mappen:
  idatt2900-bachelorgruppe-127/frontend/src/store.js
- På linje 63 inne i filen, sett *state.rootUrl* = "{url-fra-backend-system}/api/"
- Frontend vet nå hvilken URL den skal bruke for å utføre spørringer mot tjeneren (backend-system).

For å logge deg inn på nettsiden kan du hente brukernavn og passord fra Vedlegg *M Innloggingsinformasjon-127* som du finner sammen med vedlagte filer for prosjektet. Skriv disse inn på innloggingssiden for nettsiden og logg inn for å teste systemet.

8.4.3 Installering og kjøring av hele stacken

1. Følg *først* instruksjoner for å klone repoet i delkapittel 8.1.
2. Hvis du har tilgang til vedlagte filer for prosjektet og ikke ønsker å opprette en egen database-tilkobling, legg **database-configuration** filen som du finner i zip-filen kalt *database konfigurasjon og krypteringsnøkkel.zip* inn i denne mappen fra det nedlastede repoet:
   idatt2900-bachelorgruppe-127/backend_system/src/main/resources

og hopp deretter over punkt 3, 4 og 5. Du vil da bruke database-tilkoblingen som hostes av oss i bachelorgruppen gjennom Microsoft Azure.

3. Sett opp en MySQL-database enten lokalt eller som hostes over nett (uten innhold). Hvis MySQL-databasen hostes over nett, noter ned URL for database-tilkoblingen, slik at du kan sette denne inn i **database-configuration** filen.
   Denne filen kan enten hentes fra vedlagte filer for oppgaven, fra zip-filen kalt *database konfigurasjon og krypteringsnøkkel.zip* eller opprettes på egenhånd, og plasseres i mappen:
   idatt2900-bachelorgruppe-127/backend_system/src/main/resources
4. Innholdet i filen skal være som følger:
   - Linje 1 skal inneholde *URL* (jdbc:mysql://localhost:8080/{navn-på-database} for lokal MySQL-database, egendefinert eller autogenerert URL fra punkt 3 for eksterne databaser).
   - Linje 2 og 3 skal inneholde henholdsvis *brukernavn* og *passord* for tilkobling til databasen. Disse velger du selv når du oppretter databasen, uavhengig av om den er opprettet lokalt eller hostes på en ekstern plattform.
5. Plasser **key.hmac** filen i mappen:
   idatt2900-bachelorgruppe-127/backend_system/src/main/resources
   Denne finner du også i samme zip-fil som i punkt 2/3, og kan kun hentes derfra.
6. Følg instruksjoner for kjøring av system-backend i delkapittel 8.3.
7. Plasser den samme **database-configuration** filen som du hentet/opprettet i punkt 3, inne i denne mappen:
   idatt2900-bachelorgruppe-127/backend_obtain/src/main/resources
8. Følg instruksjoner for kjøring av web-scraper i delkapittel 8.2.
9. Følg instruksjoner for kjøring av frontend i delkapittel 8.4
10. **FERDIG.** Databasen er konfigurert, og kjører på definert port. Web-scraperen kjører, og henter inn informasjon til databasen. Frontend er startet, og kan kommunisere med databasen.

9. Dokumentasjon av kildekode

I frontenddelen er det skrevet JSDoc for å dokumentere kode. For å se JSDoc i nettleseren er det to mulige måter å gjøre det på, avhengig av hvilke filer som blir brukt:

- For javaScript-filer kan man skrive disse kommandoene, dersom man befinner seg i frontend-mappen:
    - npm install -g jsdoc
        - Eller: npm install --save jsdoc
        - Laster ned det man trenger for å bygge JSDoc i JavaScript-filer.
        - -g betyr at man laster ned JSDoc globalt, mens –save gjør at man laster ned JSDoc lokalt.
    - jsdoc path\to\file.js
        - path\to\file.js byttes ut med en ordentlig path.
        - Eksempel: jsdoc src\router\index.js
            - src er rot-mappen, og må være med.
- Deretter vil det genereres index-filer som finnes i “out”-mappen på samme nivå som “src”.
- For Vue-filer må man skrive disse kommandoene, dersom man befinner seg i frontend-mappen:
    - npm install -g documentation
        - Laster ned documentation for bygging av JSDoc i Vue-filer.
    - documentation build path\to\file.vue -f html -o docs
        - path\to\file.vue byttes ut med en ordentlig path til en Vue-fil.
    - Eksempel: documentation build src\components\SecurityFeedComponents\SecurityIssueFeed.vue -f html -o docs
        - src er rot-mappen, og må være med.
- Deretter vil de nødvendige sidene være i en “docs”-mappe på samme nivå som “src”.

I backenddelene er det skrevet JavaDoc for dokumentering. For å se JavaDoc i nettleseren kan man gjøre det på forskjellige måter. I utviklingsprosessen har vi brukt det integrerte utviklingsmiljøet Intellij IDEA. Derfor baserer vi genereringen av JavaDoc på hvordan man kan gjøre det i Intellij:

- Øverst i programmet finner man “Tools”-menyen
- I nedtrekksmenyen finner man valget “Generate JavaDoc”
- Man kan velge hvilket scope man skal generere i. Det enkleste er å generere i “Whole Project” som er prosjekt-scopet.
- Deretter må man fylle inn “Output directory”, altså hvor man skal lagre JavaDoc-en.
- Deretter trykker man på “Generate”-knappen nederst i vinduet.
- Videre kan man gå til mappen som man valgte som “Output directory”, og åpne index.html. Man vil deretter bli videresendt til nettleseren, som åpner opp siden med JavaDoc.

10. Kontinuerlig integrasjon og testing

10.1 Kontinuerlig Integrasjon

Systemet har et kontinuerlig integrasjons oppsett om er spesifisert i *.gitlab-ci.yml* i prosjektets rotmappe.
Oppsettet for CI er ikke veldig omfattende men omfatter å bygge samt kjøre backend_system og frontend.

Løsningen blir ikke testet på noen andre plattformer enn linux-baserte docker images. Dette er grunnet at systemet er designet for å kjøre i Docker da det gir økt sikkerhet. Dette gjør at testing på forskjellige plattformer ikke blir relevant da kjøretidsmiljøet teknisk sett er konstant.

Grunnen til at backend_obtain delen av prosjektet ikke blir testet er at den krever et ganske avansert test oppsett som vi ikke rekker å implementere da den kjører forskjellige chrome-driver funksjoner, og krever virtualisering av skjerm-virtualisering etc. Dette er implementert i Dockerfilen til obtain.

Oppsett:

- CI fila blir kun kjørt ved push til main-branchen i GitLab repositoryet.
    - Pushen går kun gjennom og blir gyldig om alle stadiene av CI-fila kjører velykket.

- 2 stadier i CI fila, build og test
  Build:
    - Bygger frontend, går igjennom ved exit code 0

  Test:
    - Tester backend, går igjennom hvis ingen av testene feiler.
    - Tester frontend, sjekker altså om dist/ mappa blir opprettet etter build-stadiet.

10.2 Testing

10.2.1 Automatiserte

De automatiserte testene som er implementert i systemet er plassert under src/test i backend_obtain og backend_system. Frontend har ikke implementert noen form for automatisert testing.

Testdekningen i systemet er ganske lav da det var fokus på å få implementert så mye viktig funksjonalitet som mulig i utviklingsfasen av prosjektet.

- Obtain har kun tester for om systemet klarer å starte all viktig funksjonalitet som nettleser for web-scraping etc.
- System har kun tester for CustomerSesrvice-delen av systemet.

De automatiserte testene i backend_observe kan kjøres ved hjelp av maven goalet: *mvn test*

De automatiserte testene i backend_obtain fungerer for øyeblikket kun på et vertsoperativsystem og ikke inne i en container. Det er grunnet forskjellige egenskaper som viste seg var utfordrende å implementere med tiden vi hadde.
Testene kan kjøres med maven goalet: *mvn test.* (NB: pass på at du har google chrome Versjon 101.x.xxxx.xx installert).

10.2.2 Brukertester

Brukertesting av systemet ble gjort på slutten av utviklingsperioden, prosessen rundt det er beskrevet i hovedrapporten del 4.1.4 og diskutert i 5.1.4. Kjapt oppsummert gikk den ut på å distribuere systemet slik at brukergruppen av systemet kunne teste det, dermed gi tilbakemelding på et spørreskjema (se vedlegg J).

AIRS v1 - Anmeldelse etter demo

Et spørreskjema som svares på etter demo-gjennomgang av AIRS (Atea Incident Response System). Målet med skjemaet er å se hvilke daglige effekter Atea Incident Response Team kan oppnå ved å benytte seg av systemet.

Det er viktig at alle svar baseres på hvilke funksjoner som er inkludert i v1 og presentert/prøvd på demoen (hvis ikke det eksplisitt blir bedt om noe annet).

🧍 - Undersøkelsen er anonym
⏳ - Tar maksimalt 10minutter

*Må fylles ut

Del 1: Bruk av systemet

Har som mål å kartlegge nyttigheten av AIRS samt hvor enkelt AIRS v1 er å bruke.

1. Hvor nyttig tror du AIRS v1 som presentert og testet i demoen, kan være som et verktøy i arbeidsdagen din? *

Markér bare én oval.

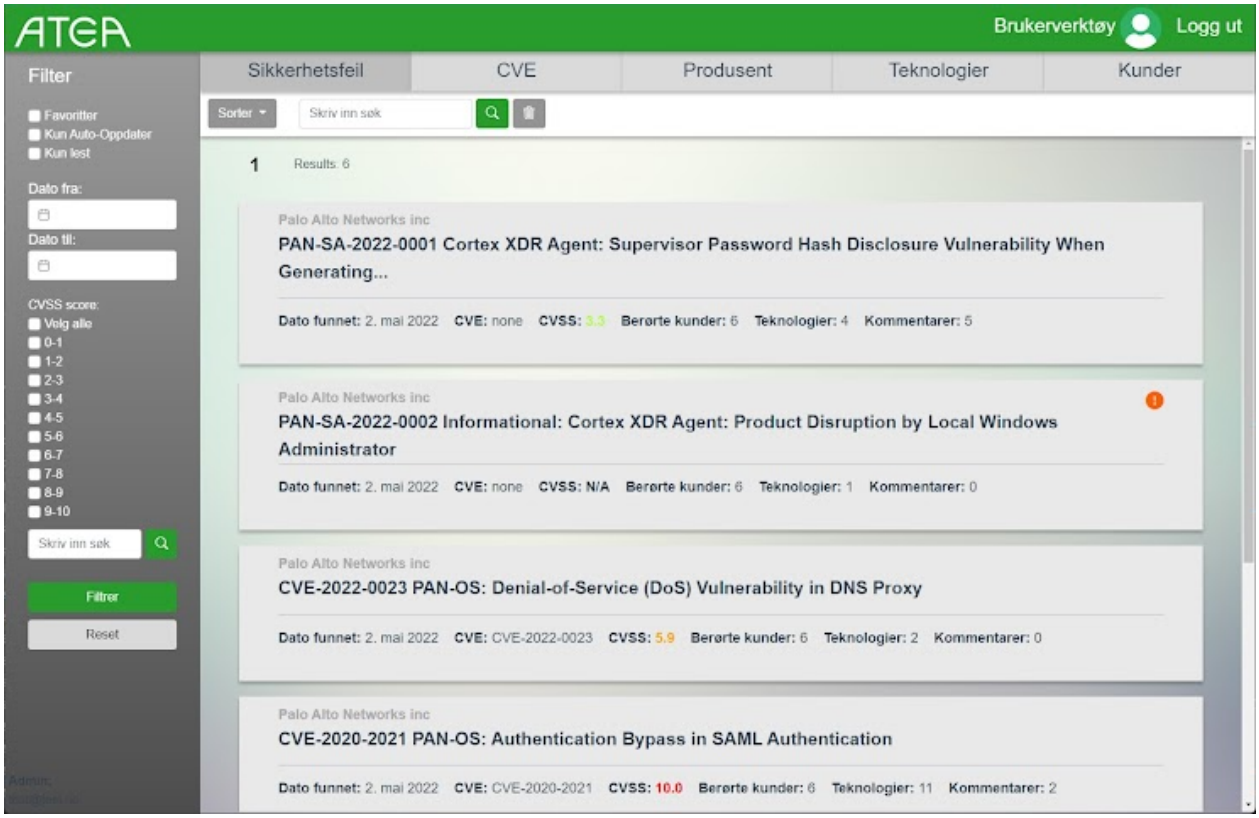
	1	2	3	4	5	
Ikke nyttig i heletatt ❌	◯	◯	◯	◯	◯	Veldig nyttig 💯

2. Hvor nyttig tror du en fremtidig versjon av AIRS (med bla.a. funksjonalitet for varsling av kunder innad i systemet, en komplett liste med kilder den henter data fra, mulighet for uthenting av statistikk, egen feed som kunder har tilgang til samt andre optimaliseringer) kan være som et verktøy i arbeidsdagen din? *

Markér bare én oval.

	1	2	3	4	5	
Ikke nyttig i heletatt ❌	◯	◯	◯	◯	◯	Veldig nyttig 💯

3. Hvordan syns du oppsettet til AIRS v1 med menyer og funksjoner(søk, kommentarfelt, filtrering og redigering av elementer) er? *



Markér bare én oval.

- ◯ Menyene og funksjonene var enkle å forstå og bruke 😍.
- ◯ Menyene var enkle å forstå og bruke, men funksjonene var ikke det.
- ◯ Funksjonene var enkle å forstå og bruke, men menyene var ikke det.
- ◯ Verken menyene eller funksjonene var enkle å forstå eller bruke 😳.
- ◯ Andre: _____

4. *Frivillig* Vil du tilføye noe ekstra som angår oppsettet av AIRS? 💬

_____

_____

_____

_____

_____

Del 2: Dagens prosesser VS AIRS v1

Denne delen har som mål å definere til hvilken grad Atea Incident Response Team mener AIRS v1 påvirker deler av dagens prosesser som medlem av teamet.

Spørsmål-oppsett:

Det generelle oppsettet på spørsmålene er at man først blir spurt om dagens prosesser (uten AIRS) deretter hvordan man tror prosessen blir med AIRS. (Spørsmålene som henger sammen har matchende emoji 🙋🏽 🙋🏽)

- Hvis man svarer med en høyere verdi på spørsmålet angående AIRS som en del av prosessen sier man at AIRS er "forbedrer" prosessene.
- Hvis man svarer likt på begge vil det si at AIRS ikke påvirker dagens prosesser.
- Hvis man svarer med en lavere verdi på spørsmålet angående AIRS som en del av prosessen sier man at AIRS er "forverrer" prosessene.

5. 🎾 Hvor manuell syns du dagens prosesser som angår innhenting/oppdaging av sikkerhetsfeil er? *

Med innhenting/oppdaging mener vi prosessen fra en ny sikkerhetsfeil har oppstått og blitt oppdaget samt hva den omfatter har blitt forstått av et menneske.

Markér bare én oval.

	1	2	3	4	5	
Helt manuell 💪🏽	◯	◯	◯	◯	◯	Helt automatisert 🦾

6. 🎾 Hvor manuell syns du AIRS v1 gjør prosessene som angår innhenting/oppdaging av sikkerhetsfeil blir? *

Markér bare én oval.

	1	2	3	4	5	
Helt manuell 💪🏽	◯	◯	◯	◯	◯	Helt automatisert 🦾

7. 📝Hvor manuell syns du dagens prosesser vedrørende det å holde seg oppdatert på endringer i sikkerhetsfeil er? *

Med dette mener vi prosessen som angår det å oppdage det at en produsent som f.eks. Cisco oppdaterer en tidligere sikkerhetsfeil som er interessant for dere til dere oppdager den eventuelt nye informasjonen.

Markér bare én oval.

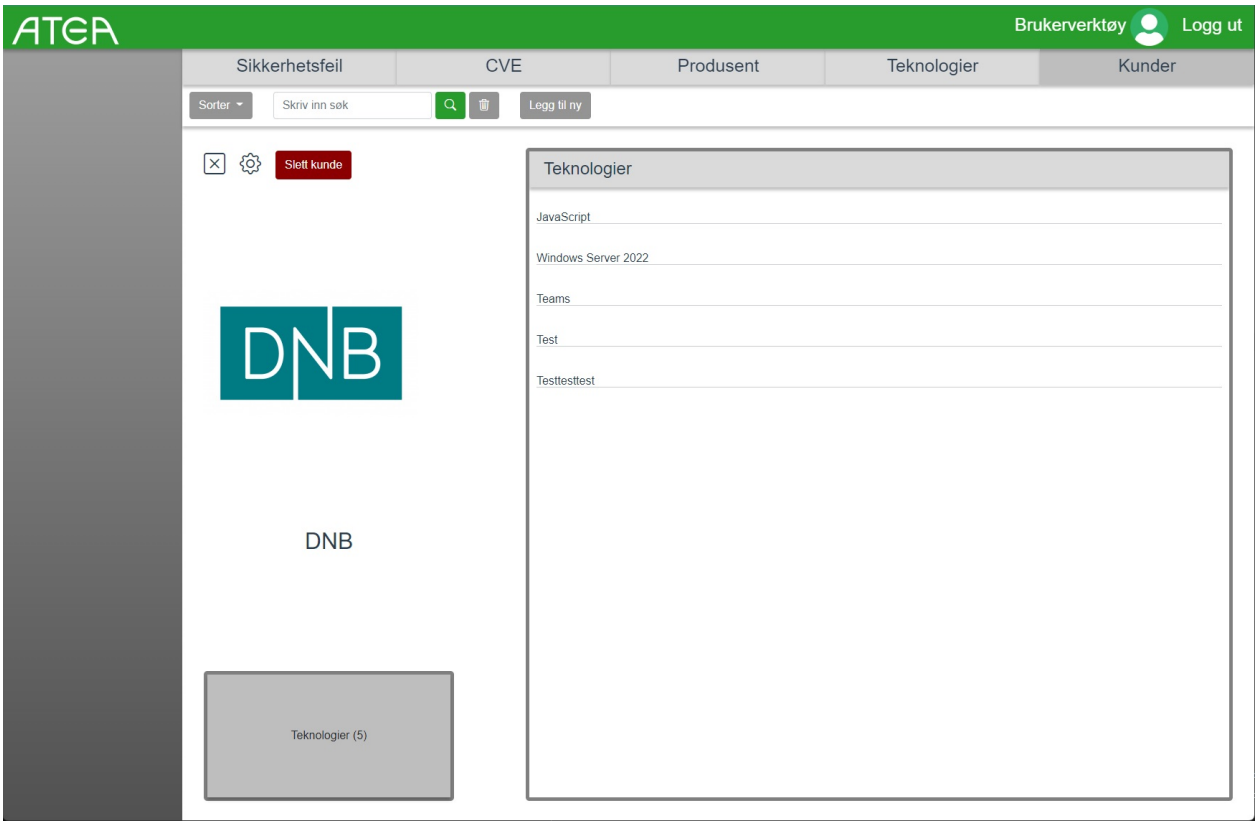
	1	2	3	4	5	
Helt manuell 💪	◯	◯	◯	◯	◯	Helt automatisert 🦾

8. 📝Hvor manuell syns du AIRS v1 gjør prosessene vedrørende det å holde seg oppdatert på endringer i sikkerhetsfeil blir? *

Markér bare én oval.

	1	2	3	4	5	
Helt manuell 💪	◯	◯	◯	◯	◯	Helt automatisert 🦾

Skjermutklipp av AIRS v1 sin oversikt over hvilke teknologier en kunde har (fiktivt):



9. 📋 Hvor effektivt syns du dagens system for å se hvilke kunder som benytter hvilke teknologier er? *

Altså oversikten over hvilke teknologier f.eks. DNB benytter seg av.

Markér bare én oval.

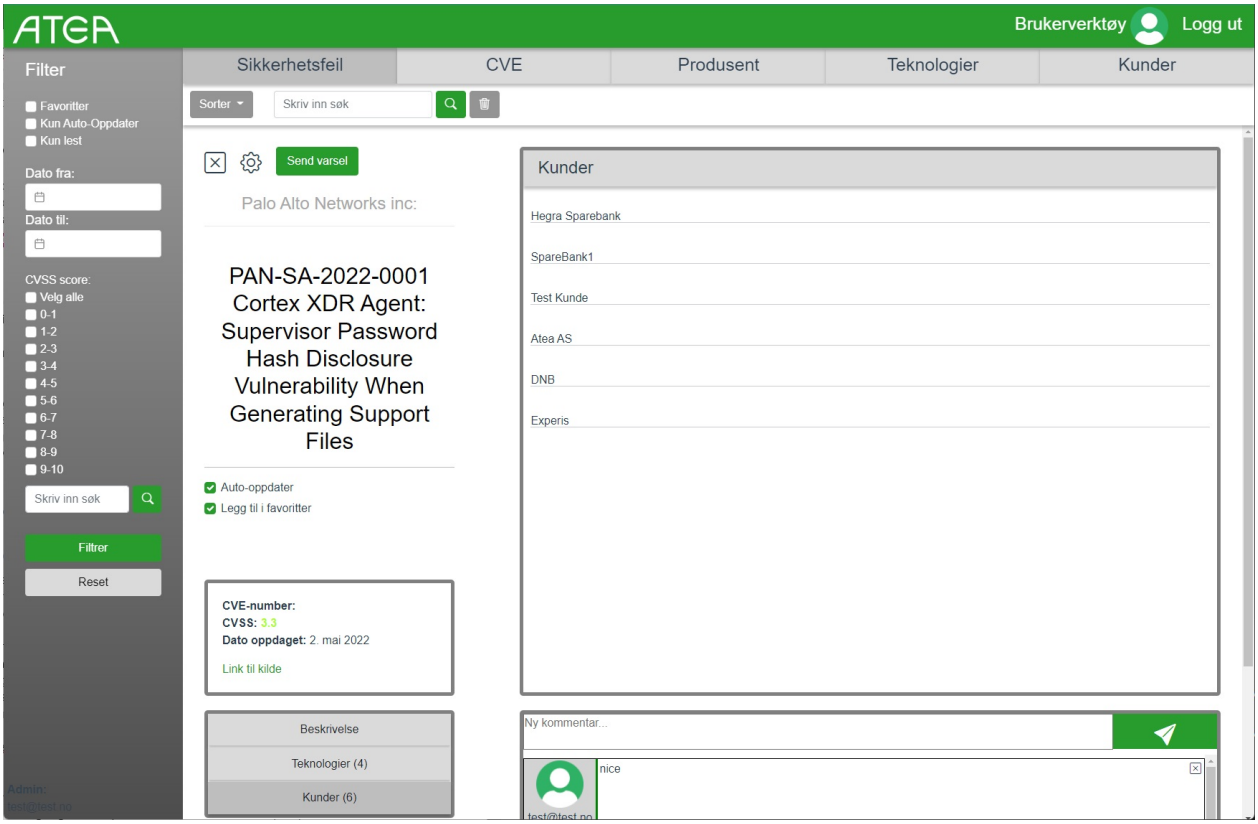
	1	2	3	4	5	
Usystematisk/treigt	◯	◯	◯	◯	◯	Systematisk/raskt

10. 📋 Hvor effektiv syns du AIRS v1 gjør systemet for å se hvilke kunder som benytter hvilke teknologier blir? *

Markér bare én oval.

	1	2	3	4	5	
Usystematisk/treigt	◯	◯	◯	◯	◯	Systematisk/raskt

Skjermutklipp av AIRS v1 sin oversikt over kunder som blir påvirket av en spesifikk sikkerhetsfeil:



11. 🗃️ Hvor effektivt syns du dagens system for å se hvilke kunder som blir påvirket av en nyoppdaget sikkerhetsfeil eller CVE er? *

Altså hvor enkelt og effektivt det er å se at en kunde blir påvirket av en nyoppdaget sikkerhetsfeil?

Markér bare én oval.

	1	2	3	4	5	
Usystematisk/treigt 🐢	◯	◯	◯	◯	◯	Systematisk/raskt 🦄

12. 🗃️ Hvor effektivt syns du AIRS v1 gjør systemet for å se hvilke kunder som blir påvirket av en nyoppdaget sikkerhetsfeil/CVE blir? *

Markér bare én oval.

	1	2	3	4	5	
Usystematisk/treigt 🐢	◯	◯	◯	◯	◯	Systematisk/raskt 🦄

13. 📢 Hvor effektivt syns du dagens system for å varsle kunder som blir påvirket av en sikkerhetsfeil er? *

Markér bare én oval.

	1	2	3	4	5	
Usystematisk/treigt 🐢	◯	◯	◯	◯	◯	Systematisk/raskt 🦄

14. 📢 Hvor effektivt syns du AIRS v1 gjør systemet for å varsle kunder som blir påvirket av en sikkerhetsfeil blir? *

Markér bare én oval.

	1	2	3	4	5	
Usystematisk/treigt 🐢	◯	◯	◯	◯	◯	Systematisk/raskt 🦄

Del 3: Fremtiden til AIRS 📈

Målet med denne delen er å kartlegge fremtiden til AIRS i forhold til funksjoner.

15. Hvordan vil du prioritere følgende fremtidige funksjoner i AIRS? (hvor 4 er høyeste prioritet) *

Markér bare én oval per rad

	1	2	3	4
Notifikasjons-system innad i AIRS	◯	◯	◯	◯
Tilgang for Ateas kunder til systemet	◯	◯	◯	◯
Varslingssystem som varsler kunder.	◯	◯	◯	◯
Mulighet for å visualisere/eksportere statistikk.	◯	◯	◯	◯

16. *Frivillig* Er det noen fremtidige funksjoner som ikke er nevnt over som du syns bør være med i en fremtidig versjon av AIRS? Gjerne gi den en prioritet.

17. *Frivillig* Er det noen funksjoner som allerede ER IMPLEMENTERT i AIRS v1 som du mener ikke trenger å være der?

18. *Frivillig* Syns du AIRS v1 bør videreutvikles?

Markér bare én oval.

- Ja
- Nei

2 svar

Tar imot svar

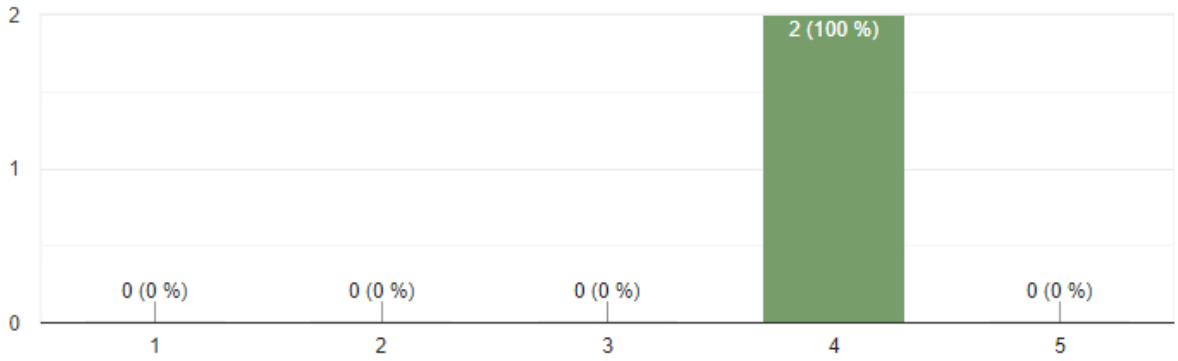
Sammendrag Spørsmål Individuell

Del 1: Bruk av systemet

Hvor nyttig tror du AIRS v1 som presentert og testet i demoen, kan være som et verktøy i arbeidsdagen din?

Kopiér

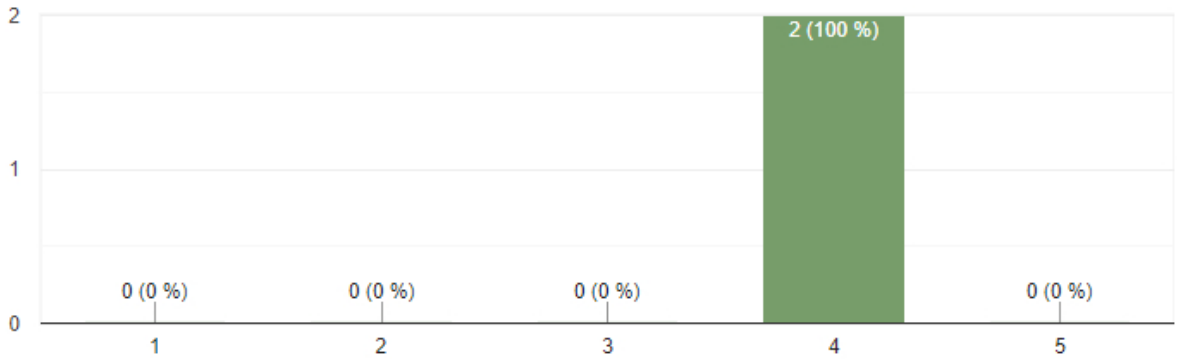
2 svar



Hvor nyttig tror du en fremtidig versjon av AIRS (med bla.a. funksjonalitet for varsling av kunder innad i systemet, en komplett liste med kilder den henter data fra, mulighet for uthenting av statistikk, egen feed som kunder har tilgang til samt andre optimaliseringer) kan være som et verktøy i arbeidsdagen din?

Kopiér

2 svar



Hvordan syns du oppsettet til AIRS v1 med menyer og funksjoner(søk, kommentarfelt, filtrering og redigering av elementer) er?

Kopiér

2 svar

Frivillig Vil du tilføye noe ekstra som angår oppsettet av AIRS? 💬

1 svar

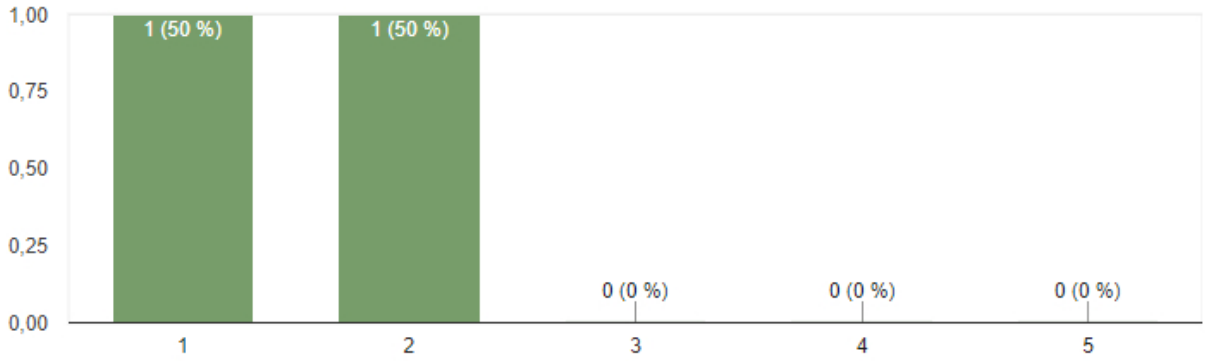
Brukervennlig grensesnitt, praktisk bruk av funksjonene vil uansett være avgjørende da det er en kompetent brukergruppe.

Del 2: Dagens prosesser VS AIRS v1

🔎 Hvor manuell syns du dagens prosesser som angår innhenting/oppdaging av sikkerhetsfeil er?

Kopiér

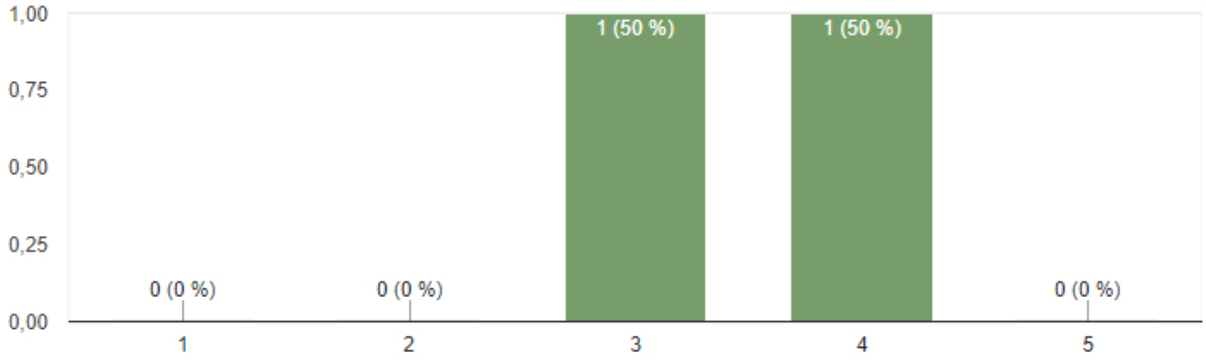
2 svar



🔎 Hvor manuell syns du AIRS v1 gjør prosessene som angår innhenting/oppdaging av sikkerhetsfeil blir?

Kopiér

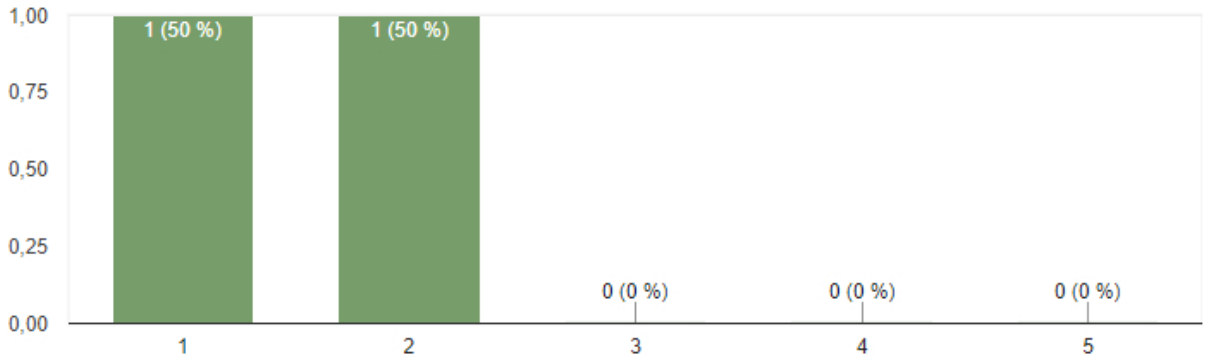
2 svar



📝Hvor manuell syns du dagens prosesser vedrørende det å holde seg oppdatert på endringer i sikkerhetsfeil er?

Kopiér

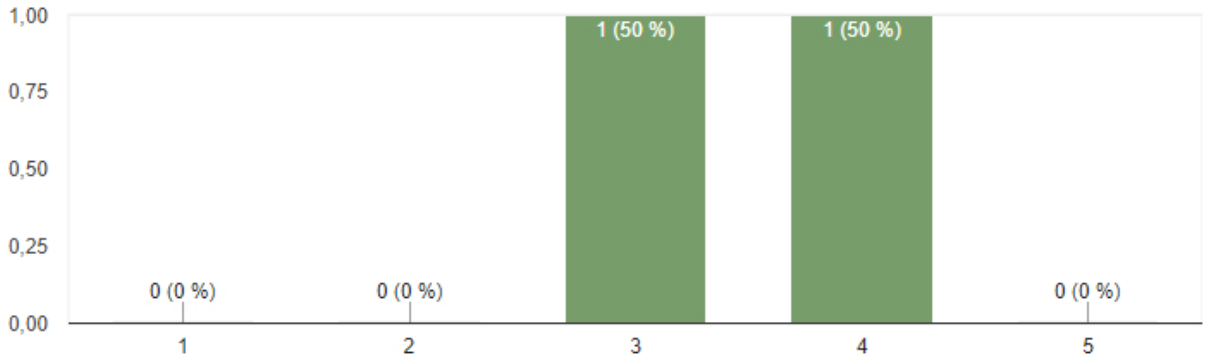
2 svar



📝Hvor manuell syns du AIRS v1 gjør prosessene vedrørende det å holde seg oppdatert på endringer i sikkerhetsfeil blir?

Kopiér

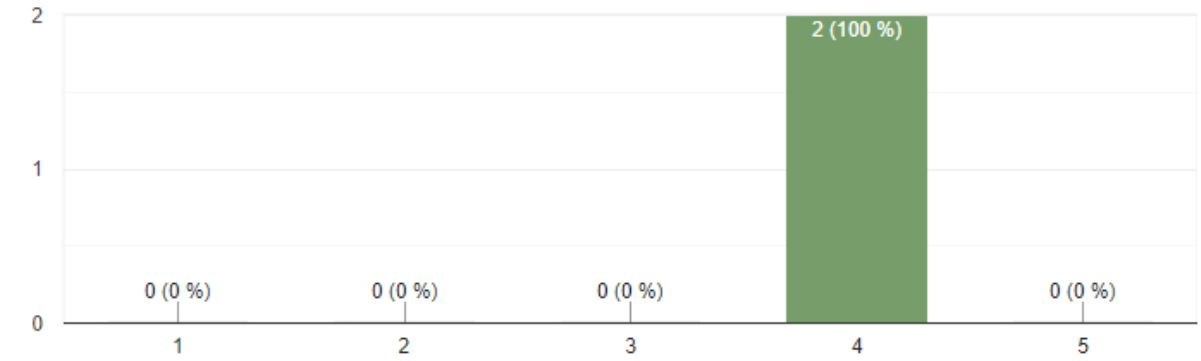
2 svar



📋 Hvor effektivt syns du dagens system for å se hvilke kunder som benytter hvilke teknologier er?

Kopiér

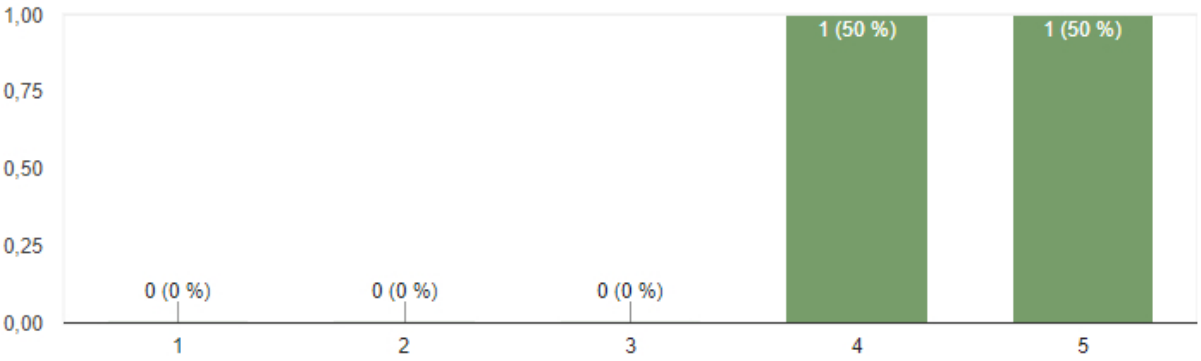
2 svar



📋 Hvor effektiv syns du AIRS v1 gjør systemet for å se hvilke kunder som benytter hvilke teknologier blir?

Kopiér

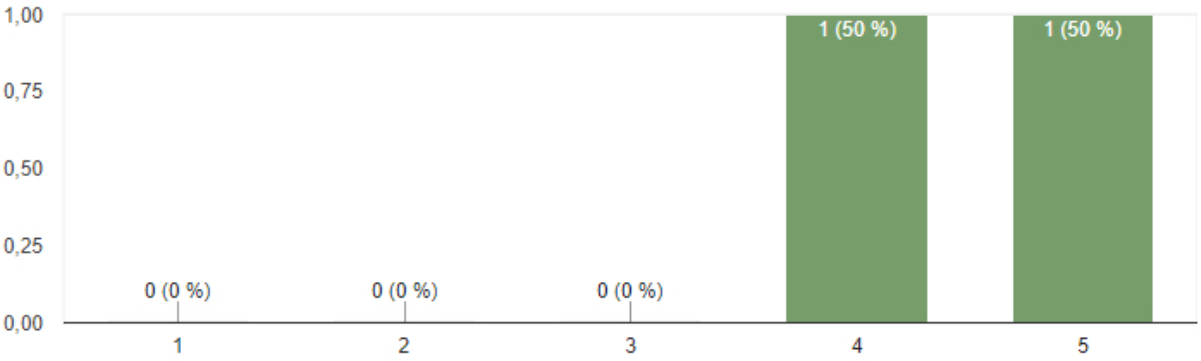
2 svar



🗂️ Hvor effektivt syns du dagens system for å se hvilke kunder som blir påvirket av en nyoppdaget sikkerhetsfeil eller CVE er?

Kopiér

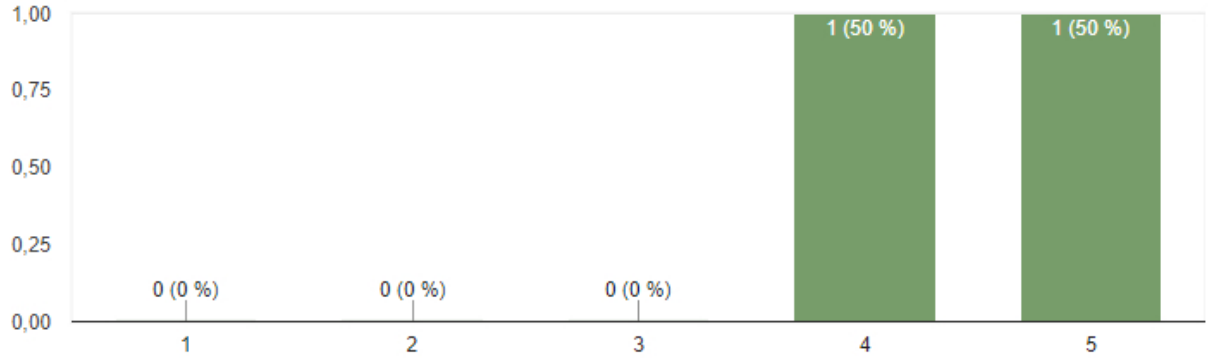
2 svar



🗂️ Hvor effektivt syns du AIRS v1 gjør systemet for å se hvilke kunder som blir påvirket av en nyoppdaget sikkerhetsfeil/CVE blir?

Kopiér

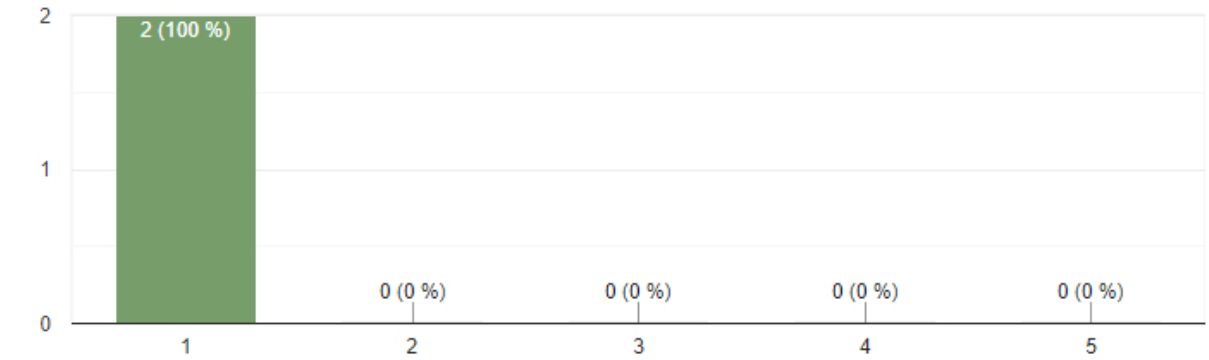
2 svar



📌 Hvor effektivt syns du dagens system for å varsle kunder som blir påvirket av en sikkerhetsfeil er?

Kopiér

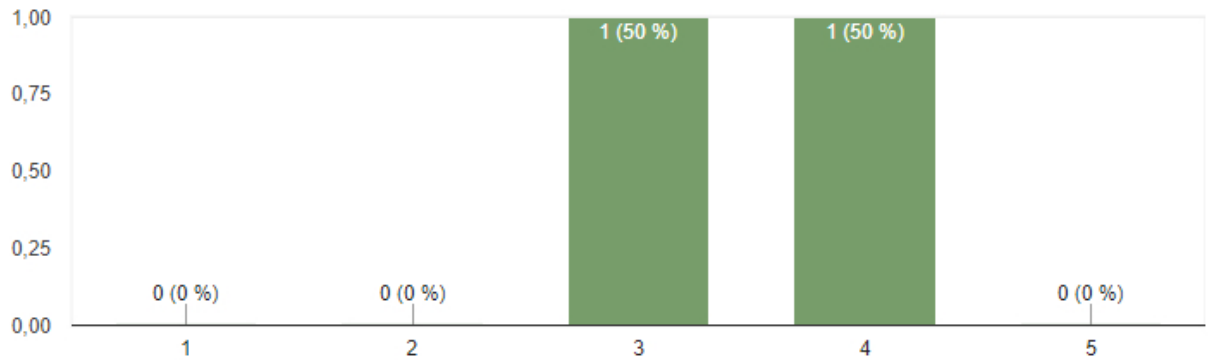
2 svar



📌 Hvor effektivt syns du AIRS v1 gjør systemet for å varsle kunder som blir påvirket av en sikkerhetsfeil blir?

Kopiér

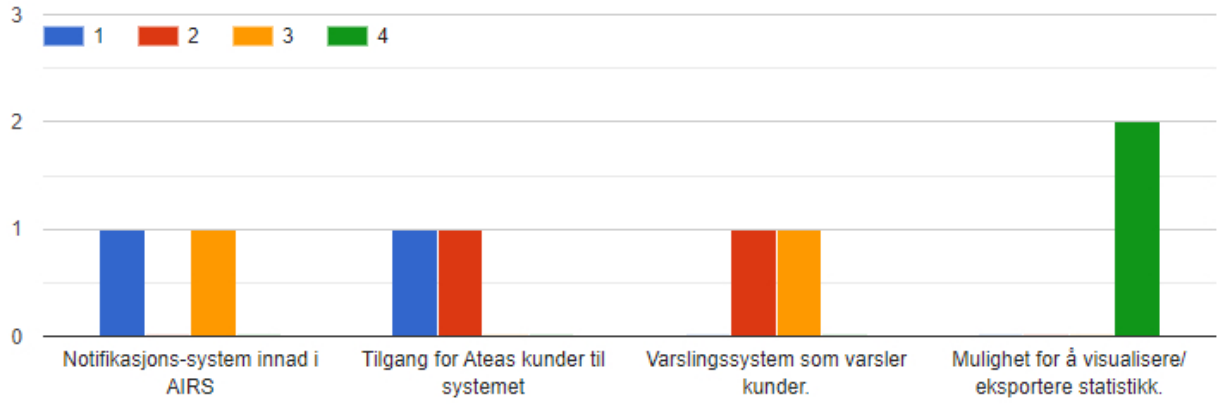
2 svar



Del 3: Fremtiden til AIRS 📈

Hvordan vil du prioritere følgende fremtidige funksjoner i AIRS? (hvor 4 er høyeste prioritet)

Kopiér



Frivillig Er det noen fremtidige funksjoner som ikke er nevnt over som du syns bør være med i en fremtidig versjon av AIRS? Gjerne gi den en prioritet.

1 svar

"Enkel" metode for adde kilder

Frivillig Er det noen funksjoner som allerede ER IMPLEMENTERT i AIRS v1 som du mener ikke trenger å være der?

1 svar

Nei

Frivillig Syns du AIRS v1 bør videreutvikles?

Kopiér

2 svar