Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Magnus Sustad
Oskar Keogh
Tobias Ellefsen
Esben Bjarnason

Automatisk rapportskriving og testing av digitale arkiv

Bacheloroppgave i Programmering
Veileder: Deepti Mishra

Mai 2021

NTNU
Kunnskap for en bedre verden

Magnus Sustad
Oskar Keogh
Tobias Ellefsen
Esben Bjarnason

Automatisk rapportskriving og testing av digitale arkiv

Bacheloroppgave i Programmering
Veileder: Deepti Mishra
Mai 2021

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Norwegian University of
Science and Technology

Abstract

The vast majority, whether it is a business or a private person, produce many types of documents such as accounts, correspondence, decisions, exam results and birth certificates. Many of these documents are very important, and must therefore be available far into the future. These documents are deposited as archive material to archival institutions for preservation and protection for posterity. An important part of this process is to validate the archive material for its authenticity, and as of today this is done manually, which can take from an hour to a month per extract. This amount of time spent on archives bundled with the staggering amount of more digital archives being deposited is a huge problem. Innlandet County Archives / IKA Opplandene came to us with a wish for a solution that will automate this process for the digital archives that are deposited. The solution is a Java application with a graphical user interface which automatically tests and validates archive extracts in relation to the Noark standard, as well as generates a report based on the results. The user can configure the tests, report contents and what happens to the extract after completing the testing, so that the solution can stay up to date with the changing laws and standards. There has been a strong focus on delivering high code quality, developing in a professional work environment and being flexible in the form of the scrum methodology.

Sammendrag

De aller fleste, om det er en virksomhet eller en privatperson, produserer mange typer dokumenter som for eksempel regnskap, korrespondanse, vedtak, eksamensresultater og fødselsattester. Mange av disse dokumentene inneholder viktig informasjon, og må derfor være tilgjengelige langt inn i fremtiden. Dokumentene blir deponert som arkivmateriale til arkivinstitusjoner for bevaring og sikring for ettertiden. En viktig del av denne prosessen er å validere arkivmateriale for sin autentisitet og per i dag blir dette gjort manuelt, noe som kan ta fra én time til én måned per uttrekk. Denne tiden brukt på arkiver samlet med den svimlende mengden flere digitale arkiver som deponeres er et stort problem. Innlandet fylkesarkiv / IKA Opplandene kom til oss med et ønske om en løsning som skal automatisere denne prosessen for de digitale arkivene som blir deponert. Løsningen er en Java applikasjon med grafisk brukergrensesnitt som automatisk tester og validerer arkivuttrekk i forhold til Noark-standarden, samt genererer en rapport ut i fra resultatene. Brukeren kan selv konfigurere testene, innholdet i rapporten og hva som skjer med uttrekket etter fullført testing, slik at løsningen kan holde seg oppdatert med skiftende lover og standarder. Det har vært stort fokus på å levere høy kodekvalitet, utvikle i et profesjonelt arbeidsmiljø og være smidige i form av metodikken scrum.

Forord

Dette er en bacheloroppgave, utstedt og gjennomført i samarbeid med Innlandet fylkesarkiv / IKA Opplandene, for Institutt for datateknologi og informatikk (IDI) ved Norges teknisk-naturvitenskapelige universitet (NTNU) i Gjøvik.

Vi ønsker først å takke Innlandet fylkesarkiv for deres tid og samarbeid igjennom hele prosessen. Kontaktperson Per Arne Stenshagen og systemansvarlig ved elektroniske arkiv Pål Mjørlund har vært veldig hjelpsomme og tilgjengelig for alle møter og brukertester.

Takk til vår veileder Deepti Mishra for god veiledning og oppfølging. Hennes kunnskaper har vært til god hjelp for prosjektet vårt.

Til slutt ønsker vi å takke familie, de ansatte ved NTNU og hverandre for godt samarbeid og en lærerik tid.

Innhold

Figurer

Kodelister

Akronymer

AIP Archival Information Package. 2, 3, 11, 20, 22, 33

API Application Programming Interface. 33, 69

CMD Command Prompt. x, 39, 40

CSS Cascading Style Sheets. 36

GNU GNU's Not Unix. 69

HTML HyperText Markup Language. x, 26, 27, 45, 50–53, 60

IDE Integrated Development Environment. 26

IKA Interkommunalt arkiv. iii–v, 1

JAR Java Archive. 20, 55, 64

JDK Java Development Kit. 60, 69

JVM Java Virtual Machine. 25, 26, 69

MVC Model-View-Controller. 4, 23, 24, 35, 70, 71

ODF OpenDocument Format. 11

QA Quality Assurance. 56

SDK Software Development Kit. 69

TAR Tape Archive. 10–12, 14, 40

UML Unified Modeling Language. 13

XML Extensible Markup Language. x, 10–14, 26, 41, 44–46, 52, 64, 70

Ordliste

branch Et konsept som viser den nyeste versjonen av koden fremst på grenen og alle tidligere versjoner er tatt vare på bakover på grenen. 56, 57, 59, 60, 67

bytecode Instruksjonssettet til Java Virtual Machine. 25, 68

code smells Et karakteristisk trekk i kildekoden som kan indikere et dypere problem. 58

cognitive complexity En måleenhet for hvor vanskelige det er å intuitivt forstå en kodeenhet. 58

injeksjon Når det sendes ondsinnet data som ikke blir validert til et system. 9

issue En sak, oppgave eller funksjonalitet som er en del av et system. 17, 18, 56, 68, 71

map Java map er en samling av data med tre grensesnitt. Grensesnittene gjør at man kan se på samlingen som et sett med nøkler, verdier eller nøkkel-verdi par. 43, 47, 48

parse Transformerer en type data til en annen type for å gjøre dataen leselig. 27, 51, 52

push Et uttrykk i versjonskontroll av kode der man laster opp en ny versjon av koden til repository-et. Det motsatte er pull. 56, 57, 60

repository En sentralisert lokasjon hvor data er lagret og håndtert. 67

scrum board Et verktøy som gir utviklerne oversikt over hva slags ting som er fullført, under arbeid eller venter på å bli gjort. 17–19, 56

sprint planning Møte på starten av en sprint. Gruppen planlegger hva som skal implementeres i sprinten. 17, 19, 68

sprint retrospective Møte på slutten av en sprint der man evaluerer gruppen og lager en plan for forbedringer som kan implementeres i neste sprint. 17, 19, 68

sprint review Møte på slutten av en sprint sammen med produkteier. Her ser man på det som har blitt levert i løpet av sprinten og får tilbakemeldinger og innspill. 15, 16, 18

wireframe Visuell guide som representerer skjelettrammen til et program. 19, 28–30

Kapittel 1

Innledning

1.1 Problemområde

Det finnes mange typer dokumenter som for eksempel korrespondanse, notater, regnskap og film. Alle virksomheter i Norge, både privat og offentlig, produserer slike dokumenter hver dag som hjelper dem til å nå målet sitt [1]. De dokumenterer hva som ble sagt, hva som er bestemt og grunnlaget for avgjørelser. Virksomhetene produserer disse dokumentene for å sikre seg selv i ettertid, for eksempel for rettslige eller velferds relaterte grunner. Derfor er det svært viktig for dem at disse dokumentene blir tatt vare på og sikret. For å gjøre bevaringen lettere blir det brukt arkivdokumenter i stedet, det vil si et dokument som bare har ett eksemplar og lagret i en arkivinstitusjon. Det er også viktig for befolkningen å kunne bevare viktig informasjon som fødselsdokumentasjon, eksamensresultater, eiendomsdokumenter og adopsjonspapirer som de kan få bruk for i ettertid [1]. Samtidig må historiske dokumenter også bli tatt vare på fordi forskere og historikere trenger dem for å løse sine problemstillinger, men ikke minst så trenger folket det, for å bevare vår identitet.

1.2 Avgrensning

Innlandet fylkesarkivet / IKA Opplandene er en arkivinstitusjon drevet av Innlandet fylkeskommune som samler inn og bevarer arkivmateriale. IKA Opplandene er et interkommunalt arkivsamarbeid for kommunene i Innlandet og noen i Viken, og blir styrt av fylkesarkivet [2]. Fylkesarkivet er dermed et arkivdepot for både fylkeskommunen og de 44 kommunene i IKA Opplandene. Det vil si at disse virksomhetene kan deponere sine arkiver dit for bevaring. En stor del av jobben ved å samle og bevare arkivmateriale er å sikre at de er gyldige og godkjente etter arkivloven. Arkivloven er styrt av Arkivverket og har som formål til å sikre en helhetlig samfunnsdokumentasjon [3]. I tillegg jobbes det hardt med å digitalisere arkivene, slik at befolkningen får tilgjengelighet til arkivene på ett felles sted. I 2020 ble det tre ganger så mange digitale arkivuttrekk som må testes, på grunn

av sammenslåingen av Oppland og Hedmark.

1.3 Oppgavedefinisjon

Fylkesarkivet ønsker en programvareløsning som kan automatisk teste og validere digitale arkivuttrekk, samt automatisk generere en rapport for resultatene. Disse arkivuttrekkene blir testet mot Noark-standarden som er en norsk standard for journalføring og arkivering av saksdokumenter [4].

Applikasjonen skal primært være tilgjengelig på Windows og det skal være et grafisk brukergrensesnitt som de ansatte kan benytte seg av når de utfører automatisk testing og rapportskriving. Per i dag benyttes flere tredjeparts verktøy for den manuelle prosessen. Verktøyene er Arkade, KOST-Val, VeraPDF, og DROID, og disse tester standarden, integriteten og bruk av godkjente filformater. Disse verktøyene vil bli integrert inn i løsningen via en kommandolinje, som verktøyene har implementert og eksponert. Rapporten som blir generert skal følge fylkesarkivets mal og inneholde data fra tredjepartsverktøyene i tillegg til resultater fra fylkesarkivets egendefinerte spørringer. Etter fullført testing av uttrekket skal det være mulig for brukeren å få en kopi av rapporten. Denne kan da videresendes gjennom arkivsystemet til fylkesarkivet, og det vil da bli gitt mulighet til å pakke uttrekket sammen med rapporten og alle delresultatene til en AIP pakke. AIP står for «Archival Information Package» og er et bevaringsobjekt, tilrettelagt for langtidsbevaring i arkivdepotet [5]. Denne pakken skal inneholde bevarings metadata for uttrekket og innholdet. Innholdet inneholder også en teknisk metadata for fremstilling og logiske metadata for å forstå innholdet.

1.4 Formål

Det er først og fremst Innlandet fylkesarkiv som kommer til å ha mest interesse i dette prosjektet. Det er fordi applikasjonen er utviklet med deres samarbeid, krav og tilbakemeldinger. Man kan også se flere steder i applikasjonen at den er utviklet for Innlandet fylkesarkiv, spesielt i rapporten, grunnet malen som er brukt. Det har derimot vakt interesse fra de andre fylkesarkivene i landet som sliter med de samme problemene som Innlandet fylkesarkiv hadde før prosjektet.

Prosjektet er ønsket fordi den vil gjøre testing og validering av arkivuttrekkene mye mer effektive. Den manuelle prosessen består av å manuelt konfigurere og sette opp flere testmiljøer og deretter kjøre disse testene en etter en. Man må altså være til stede for å kunne fortsette prosessen etter en deltest er fullført. Så må man lese og tolke testresultatene, noe som kan være veldig vanskelig å forstå, siden mesteparten egentlig bare er ment for å bli lest av en datamaskin. Etter at man har plukket ut den relevante informasjonen (som kan bli en veldig stor mengde ut fra størrelsen på uttrekket) plottes den inn i rapportmalen. Deretter

blir uttrekket pakket sammen med resultatene til en AIP pakke, som blir sendt videre i arkivsystemet. Denne prosessen er svært tidkrevende og kan vare alt fra en time til en måned.

Vi har motivasjon til å utføre dette prosjektet fordi det er snakk om en applikasjon med bruk i det virkelige liv, og ikke minst at den vil bli brukt av en arkivinstitusjon som vi støtter. Bevaring av arkiv er noe som aldri kommer til å forsvinne og det er svært viktig at materialet som blir bevart er autentisk for både den individuelle og for nasjonens identitet. Ved å utvikle denne applikasjonen vil vi derfor hjelpe til med bevaring av arkivmateriale og hjelpe de ansatte ved Innlandet fylkesarkiv, og kanskje flere etterhvert, til å gjøre jobben sin mer effektivt. De ansatte ønsker seg en automatisert prosess for nettopp denne grunnen.

1.5 Målgruppe

Brukergruppen for dette systemet kommer til å være de ansatte hos fylkesarkivet, som har kompetanse innen utvikling og programmering. Dette er folk som allerede har kunnskap i mange av bibliotekene og programmeringsspråkene vi kommer til å bruke, og jobber med testing og rapportskriving på uttrekkene.

For selve rapporten vi skriver så er denne rettet mot sensur og ansatte hos fylkesarkivet som skal drive med videreutvikling og integrering med det nye systemet de snart får. Samtidig vil det være interessant for studenter innen programmering som har interesse for integrering av tredjepartsverktøy, datainnhenting med bruk av XQuery spørringer, Java utvikling og hvordan det er å jobbe i et smidig profesjonelt arbeidsmiljø.

1.6 Egen bakgrunn og kompetanse

Alle i gruppen har studert bachelor i programmering ved NTNU på fulltid, og har generell studiekompetanse innen de fleste fagområder. De spesifikke kunnskapene vi har nytte av i denne oppgaven er kompetanse innen systemutvikling og programmeringsspråkene C++ og Java, som ga oss kompetanse innen objektorientert programmering. Vi har også alle programmert i ulike operativsystemer og er kjent med Windows samt Linux. Vi har også kunnskap innen både frontend- og backend utvikling.

Selv med et bra utgangspunkt for å takle denne oppgaven, så var det en rekke med teknologier vi ikke var kjent med på forhånd. Det var verktøy vi trengte for å løse noen deler av systemet og verktøy som bedriften allerede tok i bruk i sitt tidligere system. Så de største teknologiene vi måtte lære gjennom utviklingsprosessen var Arkade, BaseX, og Java-bibliotekene Apache POI og jsoup. Mer om disse teknologiene er beskrevet i kapittel 4.

1.7 Rammer

Selv om fylkesarkivet ga oss relativt frie tøyler til å utvikle programvaren hadde de noen rammer som var viktig at vi fulgte. Den første rammen var at programmet skulle være lett å modifisere. Malen brukt for rapporten skal være lett tilgjengelig for endringer, samt implementeringen av de forskjellige tilfellene som kommer opp i hvert kapittel. Disse tilfellene bestemmer hva slags informasjon som skal skrives i rapporten ettersom hva testresultatene er. Denne tilgjengeligheten og vedlikeholdsevnen er viktig for at automatiseringen skal kunne oppdatere seg i lik linje som arkivloven.

Den andre rammen er hvordan resultatene skulle bli håndtert. Resultatene fra de egendefinerte spørringene som de ansatte lager for å hente mer spesifikk informasjon fra testresultatene skal skrives ut i et tekstdokument per spørring. Dette gjelder også for lister eller annen informasjon i kapitlene som har et antall enheter som overskrider en satt grense på 25. Hvis dette skjer skal alle enhetene bli skrevet ut i sitt eget tekstdokument. Disse tekstdokumentene skal bli dokumentert i rapporten i form av vedlegg.

Den tredje rammen er hvilke testverktøy som skal bli brukt. Tredjepartsverktøy som blir brukt til uttrekkstestene skal være Arkade, KOST-Val, VeraPDF, og DROID eller et annet verktøy som gir en oversikt over filformater i uttrekket. Selv med de satte rammene for de påkrevde verktøyene, så hadde vi frie tøyler til å integrere flere verktøy ved behov.

1.8 Øvrige roller

Prosjektet hadde på grunn av sin arkitektur og natur selvstendige deler som man kunne jobbe på parallelt. Vi valgte å dele opp prosjektet i en frontend- og backend del. Dette følte vi ville bli lettere å implementere fordi i arkitekturen MVC (som applikasjonen bruker) er det viktig at disse delene er uavhengige. Arkitekturen vil bli dypere forklart i avsnitt 4.2. Selv om vi jobbet parallelt med disse to delene kunne vi fortsatt hjelpe til og gi tilbakemeldinger på tvers, og etterhvert som frontend delen ble ferdig implementert jobbet alle på backend. I dette prosjektet gikk frontend delen ut på det grafiske brukergrensesnittet i form av flere grensesnitt i MVC og backend delen gikk ut på henting av data, automatisk rapportskriving og uttrekkstesting i form av modeller i MVC.

Alle gruppemedlemmene har rollen som utvikler i tillegg til en annen rolle som går mer på organisasjon av prosjektet.

- Magnus Sustad var utvikler og *scrum master*. Som *scrum master* hadde han ansvar for at utviklingen holdt et godt tempo, og for å organisere og lede scrum møter med gruppen. Han hadde også hovedansvar for frontend delen.

- Oskar Keogh var utvikler, referent i møter og hadde ansvar for backend delen.
- Esben Bjarnason var utvikler, teknologiansvarlig og hadde ansvar for backend delen.
- Tobias Ellefsen var utvikler, testansvarlig og hadde ansvar for backend delen.

Deepti Mishra var veileder for prosjektet. Hennes rolle var å veilede oss gjennom arbeidet gjennom hele prosjektet og hadde ukentlige møter med gruppen.

Per Arne Stenshagen var produkteier og kontaktpersonen vår hos fylkesarkivet. Han forklarte fylkesarkivets ønsker og ga tilbakemeldinger på arbeidet vårt. Vi hadde møte med produkteieren annenhver fredag, etter hver sprint.

Pål Mjørlund er systemansvarlig ved elektroniske arkiv hos fylkesarkivet og var til stede på alle møter. Han ga viktig innsikt i hvordan arkivene fungerer og hvordan vi kunne bruke de satte tredjepartsverktøyene. Både Pål og Per Arne reserverte tid for å bli med på brukertestene som ble utført.

1.9 Selve rapporten

Rapporten er delt inn i åtte kapitler, som hver tar opp en viktig del av systemutviklingsprosessen vår:

1. **Innledning:** Gir en fullverdig oversikt over oppgaven, rapporten, organiseringen og formålet til prosjektet.
2. **Kravspesifikasjon og analyse:** Beskriver hvordan gruppen og produkteieren kom fram til kravene, og hvordan disse ble stilt. Etter kravene analyserer gruppen hvordan de skal implementere disse kravene og gir en oversikt over problemstillingen på høyt nivå.
3. **Utviklingsprosess:** Beskriver hvordan gruppen planlagte utviklingen av systemet, hvordan de involverte oppdragsgiver og metodikk valg, samt hvordan de forskjellige sprintene gikk og hva de innebar.
4. **Teknisk design:** Beskriver system-flyten med bruk av sekvensdiagram og drøftingen for valg av arkitektur, samt hvordan systemets klasser passer inn. Deretter beskrives grunnlaget for valg av utviklingsmiljø, hvilke bibliotek gruppen bestemte seg for å inkludere og nødvendige tredjepartsverktøy.
5. **Grafisk design:** Gjennomføring av frontend delen til systemet. Kapitlet går ut på hvordan gruppen skisserte det grafiske brukergrensesnittet og hva slags avgjørelser som ble gjort. Deretter beskrives det hvordan strukturen og *layout-en* på de forskjellige sidene ser ut på det ferdig produktet.
6. **Implementasjon:** Beskriver alle de viktigste funksjonalitetene systemet har og hvordan de ble implementert. Deler av systemet som blir beskrevet er brukergrensesnittet, uttrekkstestene, den automatiske rapportskrivingen og

filstrukturen til applikasjonen.

7. **Kvalitetssikring:** Hvordan verktøy, enhetstesting, brukertesting og annen kvalitetssikring ble tatt i bruk for å sikre høy kodekvalitet.
8. **Diskusjon og konklusjon:** Refleksjon og diskusjon rundt valg av avgjørelser, hva som kunne blitt gjort annerledes og hva gruppen har lært gjennom utviklingen av systemet. Det er også skrevet om hva slags mål vi nådde og hva vi ikke nådde.

I rapporten vil det dukke opp både engelske terminologier og navn på komponenter brukt i systemet. De engelske terminologiene vil bli markert med kursiv, og de mest uvanlige av dem som er spesifikt rettet mot dette prosjektet vil ha definisjon liggende i ordlisten. Det er også en akronymliste i rapporten som skriver opp forlengelsen til akronymene. Disse begrepene vil være klikkbare slik at du har mulighet til å kjapt sjekke forklaringene/navnene. Egendefinerte uttrykk som blir tatt i bruk vil være markert med anførselstegn, som for eksempel funksjonsnavn. Alle kapitler og underkapitler er til stede i innholdsfortegnelsen, samt en liste over figurer og kodesnutter. Enkelte avsnitt vil inneholde referanser til andre seksjoner av rapporten. Om dette dukker opp vil det være tydelig hvilken del det refereres til og teksten vil være klikkbar for kjapp navigering.

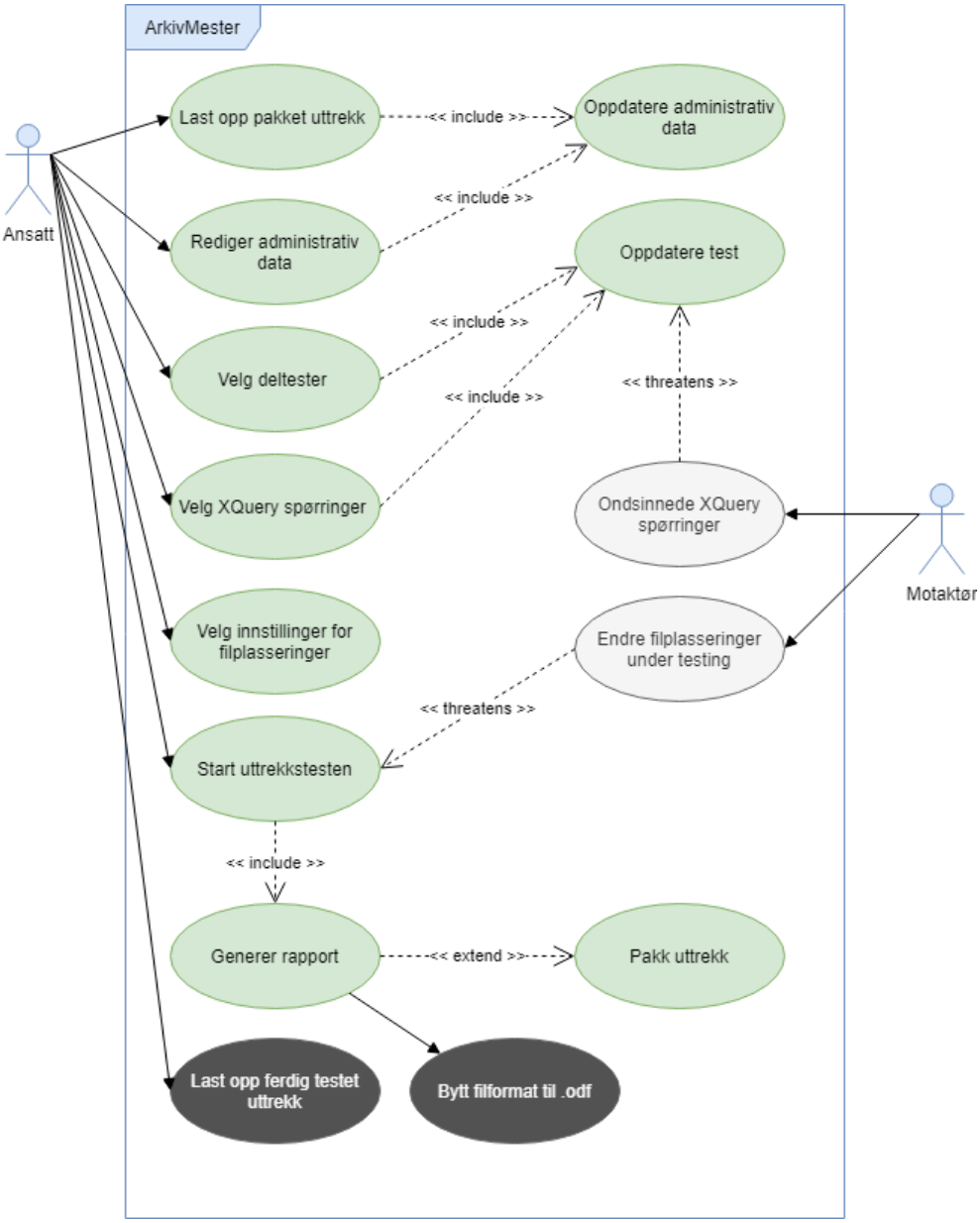
Kapittel 2

Kravspesifikasjon og analyse

Måten vi kom fram til kravene omtalt i dette kapittelet var i samarbeid med de ansatte ved fylkesarkivet. De kom først med en liste over krav som måtte være med. Denne listen inneholdt blant annet Windows kompatibilitet og at vi skulle bruke de samme verktøyene som i den manuelle prosessen. Deretter kom vi frem til flere krav som for eksempel at vi skulle implementere et grafisk brukergrensesnitt i stedet for en kommandolinje grensesnitt som de ansatte forventet. Vi brukte også rammene i avsnitt 1.7 for å definere kravene. Etter hvert som vi kom lengre i utviklingen og på grunn av naturen til metodikken (se avsnitt 3.1) vi tok i bruk kom det flere krav, noen ble fjernet og noen ble endret.

2.1 Funksjonelle krav

For å illustrere de funksjonelle kravene til programmet definerte vi flere *use cases* og lagde et *use case*-diagram, se figur 2.1 for dette. Vi definerte også to *misuse cases* som vi følte var relevante for programmets sikkerhet og slo disse sammen med resten av *use casene* i diagrammet. *Misuse caser* har gråfarge, mens de vanlige *use casene* er blå. Det er også to *use cases* som ikke ble implementert, men likevel ble inkludert siden de kan bli implementert under eventuell videreutvikling av programmet. Disse er markert som svarte i diagrammet. Beskrivelsen på *use casene*, samt utvidet beskrivelse av de to vi følte var viktigst står i avsnitt 2.3.



Figur 2.1: *Use case*-diagram

2.2 Ikke-funksjonelle krav

De ikke-funksjonelle kravene blir implementert og sikret ved at applikasjonen vil blant annet gi relevante feilmeldinger hvis en feil skulle oppstå. Dette ved å fange feil som blir kastet i «try/catch»-løkker. Brukervennligheten blir sikret ved at brukergrensesnittet er svært minimalt og inneholder bare det nødvendige, samtidig er det animasjoner som spilles av når en tung handling skjer. Definisjonen på

en tung handling er en handling som tar mer enn ett sekund å utføre, slik som uttrekkstestene og generering av rapporten. Responstid på lette handlinger har blitt testet og konfirmert via brukertester, dette gjelder også vedlikeholdsevnen, skalerbarheten og de tidligere nevnte tilfellene.

Teknisk

- **Portabilitet:** Programmet skal kun brukes uten tilgang til internett, og skal kunne distribueres til flere maskiner uten internett tilgang.
- **Tilgjengelighet:** Programmet skal være tilgjengelig 24/7 og dermed ikke krasje eller fryse. Selv om en feil dukker opp skal dette kun føre til en feilmelding, og programmet skal ikke måtte startes på nytt.
- **Pålitelighet:** Responstiden på brukergrensesnittet skal ikke være på mer enn ett sekund. Dette vil si alt som ikke krever kjøring av eksterne programmer for å teste eller hente data.
- **Vedlikeholdsevne:** Det skal være mulig for brukeren å forandre på rapportmalen og de forskjellige tilfellene som oppstår via Microsoft Word eller andre programmer som kan lese DOCX filer.
- **Skalerbarhet:** Det skal være mulig for brukeren å forandre på de forskjellige spørringene brukt av programmet for innhenting av resultatsdata for rapporten via et hvilket som helst tekstprogram.

Brukervennlighet

- Brukergrensesnittet skal bare inneholde minstekravet for å kunne bruke programmet på en vanlig måte uten distraksjoner.
- Programmet skal komme med spesifikke feilmeldinger slik at brukeren kan vite hva som er galt og hvordan det kan rettes opp i.
- Programmet skal holde oversikt over statusen for de forskjellige deltestene, altså hvilke som kjøres, er ferdige, eller venter.
- Programmet skal vise brukeren at programmet ikke har krasjet eller fryst under tunge handlinger som uttrekkstesting og generering av rapport. Dette skal vises i form av en spinner animasjon i brukergrensesnittet.

Sikkerhet

I og med at programmet skal være uavhengig av internett for å kunne bruke det, og at den eneste målgruppen av programmet er de ansatte på fylkesarkivet var ikke sikkerhet et stort fokus i denne oppgaven. Alle filer er plassert lokalt på brukeren sin datamaskin så dersom en fil har ondsinnet innhold vil dette kun påvirke én person. Derfor er dette også hva *misuse casene* våre handler om; ondsinnede XQuery spørringer og endring eller flytting av filer, hvor begge to kun kan påvirke en maskin, men ikke eksterne maskiner eller brukere. Disse *misuse casene* er beskrevet i avsnitt 2.3.1. XPath/XQuery *injeksjon* [6] er heller ikke relevant, i og

med at brukere av programmet uansett har tilgang på all data i XML filene, og at det ikke lagres sensitiv informasjon som passord i disse.

2.3 High-level use case beskrivelser

Use case navn: Last opp pakket uttrekk
Aktører: Ansatt på fylkesarkivet.
Mål: Velge hvilket uttrekk som skal testes.
Beskrivelse: Brukeren klikker på «last opp» knappen og velger riktig mappe som inneholder TAR og XML filen via sin filutforsker. XML filen vil bli brukt til å hente administrative data.

Use case navn: Rediger administrativ data.
Aktører: Ansatt på fylkesarkivet.
Mål: Fylle ut manglende administrative data som ikke er med fra den tilhørende XML filen.
Beskrivelse: Brukeren skriver inn i tekstfeltene på brukergrensesnittet der det mangler data.

Use case navn: Velg deltester.
Aktører: Ansatt på fylkesarkivet.
Mål: Styre hvilke deltester som skal bli kjørt i testen.
Beskrivelse: I brukergrensesnittet ligger det en liste over alle deltester som brukeren kan huke av eller på før de starter testen av uttrekket.

Use case navn: Velg XQuery spørringer.
Aktører: Ansatt på fylkesarkivet.
Mål: Styre hvilke egendefinerte spørringer som skal bli kjørt i testen.
Beskrivelse: I brukergrensesnittet ligger det en liste over alle spørringer som brukeren har lagt inn i XQuery mappen sin og kan huke av eller på før de starter testen av uttrekket.

Use case navn: Velg innstillinger for filplasseringer.
Aktører: Ansatt på fylkesarkivet.
Mål: Definere filplasseringene til nødvendige verktøy og kataloger for applikasjonen.
Beskrivelse: I brukergrensesnittets navigasjonsmeny ligger en innstillinger knapp som fører brukeren til en liste over filplasseringene som applikasjonen bruker. Hvis brukeren har lagret et verktøy på et annet sted enn standard filplasseringen så kan brukeren definere den her.

Use case navn: Start uttrekkstest.
Aktører: Ansatt på fylkesarkivet.
Mål: Kjøre test av uttrekket.
Beskrivelse: Brukeren klikker på «start test» knappen. Mens testen kjører vil det være en status i brukergrensesnittet om hvilken deltest som kjører og hvilke som har blitt utført.

Use case navn: Generer rapport.
Aktører: Ansatt på fylkesarkivet.
Mål: Få sluttrapporten i eget dokument så den kan sendes via arkivsystemet.
Beskrivelse: «Generer rapport» knappen vil bli aktivert når uttrekkstesten er fullført og når brukeren klikker på den vil rapporten bli generert og lagret i brukermappen. Den spesifikke fillokasjonen vil bli vist i brukergrensesnittet.

Use case navn: Pakk uttrekk.
Aktører: Ansatt på fylkesarkivet.
Mål: Pakke inn uttrekket til en AIP.
Beskrivelse: «Pakk til AIP» knappen vil bli aktivert når rapporten er ferdig generert og når brukeren klikker på den vil uttrekket sammen med resultatene og rapport bli pakket til AIP og lagret i brukermappen. Den spesifikke fillokasjonen vil bli vist i brukergrensesnittet.

Use case navn: Last opp ferdig testet uttrekk.
Aktører: Ansatt på fylkesarkivet.
Mål: Velge uttrekk med ferdiggjorte deltester for å generere rapporten.
Beskrivelse: Brukeren klikker på «last opp ferdig testet uttrekk» knappen og velger riktig mappe som inneholder TAR fil, XML fil og resultatene fra deltestene i XML format via sin filutforsker. Deretter klikker brukeren på «lag rapport» knappen som vil fungere likt som «Generer rapport» *use casen*.

Use case navn: Bytt rapportens filformat til ODF.
Aktører: Ansatt på fylkesarkivet.
Mål: Bytte filformat til sluttrapporten til ODF i stedet for DOCX.
Beskrivelse: Etter uttrekkstesten er fullført og før man genererer rapporten kan brukeren endre rullgardinen ved siden av lag «rapport» knappen til ODF i stedet for DOCX.

2.3.1 High-level misuse case beskrivelser

Misuse case navn: Ondsinnede XQuery spørringer.
Aktører: Motaktør.
Mål: Forhindre utførelsen av testingen gjennom gjentagende XQuery spørringer og XQuery feilmeldinger.
Betingelser: Uttrekk er lastet opp og klar for konfigurering av tester.
Beskrivelse: En motaktør skriver ondsinnede XQuery spørringer. Dette kan føre til sabotasje av testingen ved at for eksempel en test foregår for alltid.

Misuse case navn: Endre filplasseringer under uttrekkstesting.
Aktører: Motaktør.
Mål: Svekke system-integriteten.
Betingelser: Testing av uttrekket foregår.
Beskrivelse: En motaktør flytter på uttrekket eller andre tredjeparts verktøy sine filplasseringer under testing som fører til svekket system-integritet.

2.3.2 Utvidede use case beskrivelser

Use case navn: Laste opp pakket uttrekksmappe med følgende metadata fil.
Aktører: Ansatt på fylkesarkivet.
Mål: Velge hvilket uttrekk som skal testes
Beskrivelse: Brukeren klikker på «Last opp pakket uttrekk» knappen og velger riktig mappe som må inneholde en TAR og en XML fil via sin filutforsker. Metadata filen vil bli brukt til å hente administrative data.
Type: Essensiell.
Pre betingelser: Brukeren har en mappe som inneholder et uttrekk og en XML fil.
Post betingelser: Riktig mappe er lastet opp.
Spesielle krav: Brukeren kan laste opp pakket uttrekksmappe.
Detaljert hendelsesforløp:

1. Brukeren klikker på «Last opp pakket uttrekk» knappen.
2. En filutforsker kommer frem der brukeren kan navigere til uttrekket.
3. Marker mappen og klikk OK.
4. Uttrekket og metadata filen er nå inne i programmet.
5. Konfigurering av deltester og «Start testing» knappen blir tilgjengelig.

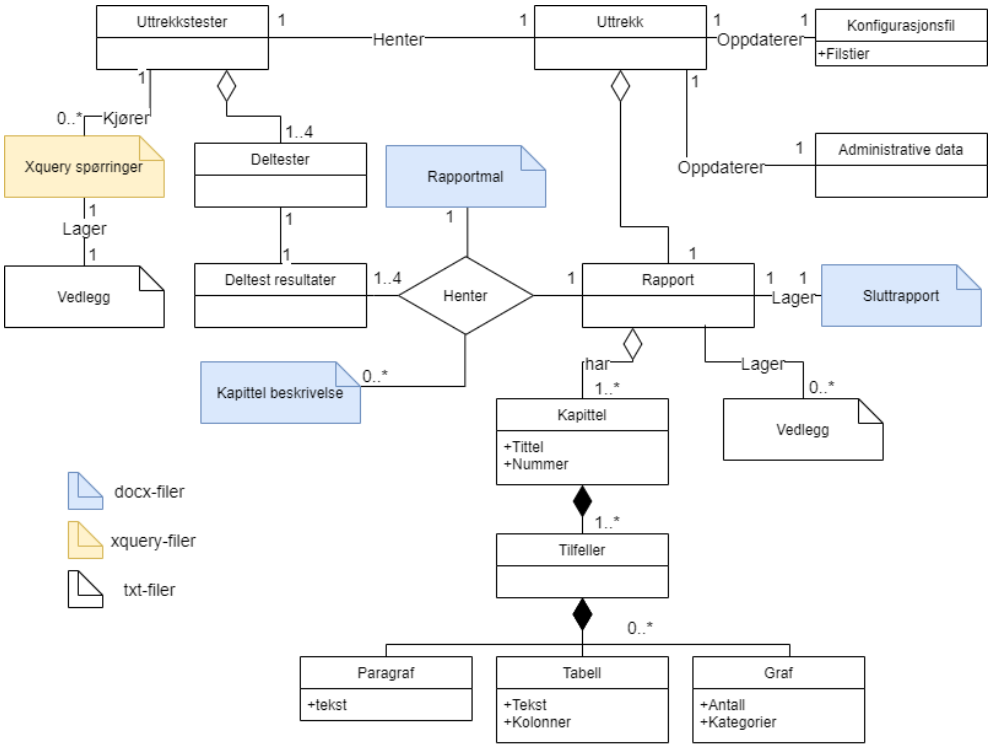
Use case navn: Velge deltester og egendefinerte XQuery spørringer som skal være med.
Aktører: Ansatt på fylkesarkivet.
Mål: Styre hvilke deltester som skal bli kjørt i testen.
Beskrivelse: I brukergrensesnittet ligger det en liste over alle deltester og en liste over alle spørringer som brukeren kan huke av eller på før de starter testen av uttrekket. Hvis en deltest er huket av kommer den til å bli kjørt i testen. For at egendefinerte spørringer skal komme opp i listen må brukeren legge disse inn i XQuery mappen sin, dens fillokasjon er definert i innstillingene i brukergrensesnittet.
Type: Sekundær
Pre betingelser: Det er lastet opp et uttrekk og følgende XML fil.
Post betingelser: Ingen.
Spesielle krav: Brukeren skal kunne konfigurere deltestene i brukergrensesnittet.
Detaljert hendelsesforløp:

1. Brukeren klikker på «Velg tester» knappen.
2. Brukeren haker av de deltestene de vil ha med og fjerner haken på de som ikke skal være med.
3. Brukeren har skrevet tilpasset XQuery spørring, så personen legger den inn i sin XQuery mappe. Brukeren har allerede sjekket innstillingene om at denne mappen er definert.
4. Den egendefinerte spørringen dukker opp i listen over XQuery deltester ved siden av listen over deltester.
5. Brukeren haker av spørringen(e) som skal kjøres.
6. Uttrekket er nå klar for uttrekkstesten.

2.4 Domenemodell

Grunnen til at vi valgte å lage domenemodellen er for å gi oss en bedre visjon over hvordan vi synes komponentene passer sammen. Dette er vanlig i objektorienterte programmeringsspråk som skal gi en oversikt over hva som gjør at klasser skiller seg fra hverandre. Dette blir gjort med å lage et diagram fylt med klasser bestående av enten data eller funksjonskall. Dataen skal gi oss et mentalt kart over beliggenhet, funksjonskall, samt gi oss en oversikt over kommunikasjonsflyten mellom flere klasser [7]. Dette skal gi oss en refleksjonsprosess som skal hjelpe oss med å rettferdiggjøre våre valg framfor oppdragsgiver. Dette gjør domenemodell til et nødvendig diagram å inkludere i prosjektarbeidet. Etter hvert som vi begynte å få en god ide komponentene som skulle inkluderes, så trengte vi å skille mellom de eksterne og interne enhetene til systemet, og vi tok av den grunn ikke i bruk domenemodell UML standarden, men har tatt mye inspirasjon fra den gjennom relasjoner og data som er satt inn i komponentene. Vi vil derfor

referere til vår modell som en «Utvidet domenemodell».



Figur 2.2: Utvidet domenemodell

Når vi begynte å kartlegge klassene så tok vi utgangspunkt i å finne de interne kjerne-komponentene i systemet og relasjonene mellom dem før vi begynte å sette inn eksterne enheter. Alle eksterne filer utenfor systemet har egen type klasse rundt seg for å skille disse fra systemkomponentene. De forskjellige fargekodene til filene hører til bestemte filtyper, og oversikt over hvilke filer som tilhører de ulike fargene er vist på figuren. Dette resulterte med den «Utvidete domenemodellen» vist på figur 2.2, der de viktigste komponentene er inkludert:

- Uttrekk: består av filplasseringen til både TAR- og XML filene som uttrekket består av og brukes for testing.
- Uttrekkstestene: alle tester og valgfrie egendefinerte XQuery spørringer som skal bli kjørt på uttrekket.
- Tredjepartsprogrammer: De nødvendige eksterne programmene som systemet vårt skal kommunisere med. Disse har vi kalt for «deltester» i figuren.
- Rapport: Alle kapitler og delkapitler som skal inkluderes i sluttrapporten.

Mer detaljert forklaring av de forskjellige komponentene finnes i kapittel 6.

Kapittel 3

Utviklingsprosess

Under planleggingsfasen i prosjektet diskuterte vi, og ble enige i hva slags utviklingsmetodikk vi skulle bruke gjennom prosjektet. Denne diskusjonen beskriver vi i prosjektplanen (vedlegg A). Her vil vi kort gjennomgå argumentasjonen for utviklingsmetodikken vi valgte, og deretter se på hvordan denne ble gjennomført i selve prosjektet.

3.1 Valg av metodikk

De viktigste karakteristikkene ved prosjektet som avgjorde valget av metodikk var:

- Konkrete mål.
- Relativt frie tøyler rundt de funksjonelle- og ikke-funksjonelle kravene.
- Oppdragsgiver som ønsket å være involvert i prosessen, og nødvendighet fra gruppens side for å ha kontakt med dem.
- Gruppens tidligere erfaring.

Gjennom samtaler med oppdragsgiver kom det frem at så lenge vi oppnådde kravene som var satt, kunne vi velge selv hvordan vi ville implementere disse, samt sette egne krav vi følte var passende. Både gruppen og oppdragsgiver var interessert i å ha god kommunikasjon med hverandre slik at vi endte opp med et produkt som møtte alles forventninger. God kommunikasjon var også viktig, i og med at det lå i oppgavens natur at vi ville trenge mye hjelp rundt de delene av oppgaven som gikk ut på å skrive ut testrapporten, slik som oppdragsgiver ønsket at den skulle se ut. Vi valgte å bruke en smidig utviklingsmetodikk, noe som passer godt overens med karakteristikkene beskrevet i listen over [8]. Utviklingsmodellen vi endte opp med bruke var scrum.

Scrum gjorde at vi kunne være fleksible i hvordan vi delte opp og valgte oppgaver som skulle gjøres. Vi kunne prioritere eller nedprioritere enkelte oppgaver avhengig av hva som var viktigst eller hastet mest. Scrum legger også opp til regelmessige møter med oppdragsgiver kalt *sprint review* møter som gjorde at oppdragsgiveren ble mer involvert i prosessen og at vi regelmessig fikk vist fremgangen vår.

Disse møtene lot også både oppdragsgiver og gruppen diskutere idéer, og komme med forslag til ting som ikke var nevnt i oppgaveteksten underveis i utviklingen. For å sikre bedre samarbeid innad i gruppen valgte vi å låne parprogrammering fra utviklingsmetodikken extreme programming [9]. Dette brukte vi på spesielt store og kompliserte oppgaver i tillegg til store forandringer i koden hvor vi følte det var bedre å la flere av gruppemedlemmene jobbe samtidig. Gruppen ble også delt opp i ulike roller med ansvar for ulike deler av prosjektet. For mer informasjon rundt scrum rollene, se avsnitt 1.8.

3.2 Gjennomføring

Vi bestemte tidlig i planleggingsfasen å ha sprinter som varte to uker hver. Den største grunnen til dette var at dersom vi brukte to uker på å utvikle produktet før vi hadde møte med oppdragsgiveren, ville vi ha mer å vise frem på hvert møte. Vi følte det ville være god nok tid til at vi kunne ha en del fremgang på utviklingen hver sprint, men også kort nok tid til at vi fikk tilbakemelding fra oppdragsgiveren, før vi eventuelt gikk for langt i feil retning. Sprintene varte fra en mandag til og med fredagen to uker senere.

3.2.1 Møter med oppdragsgiver

Sprint review

Vi hadde et *sprint review* møte på Microsoft Teams den siste fredagen i hver sprint, altså en gang annenhver uke. Grunnen for at vi møtte digitalt var på grunn av covid-19 viruset. Produkteier Per Arne Stenshagen og systemansvarlig for digitale arkiv på fylkesarkivet Pål Mjørlund var til stede hver gang. Det var også åpent for andre ansatte å møte opp. I *sprint review* møtene gikk vi gjennom en demo av prosjektet og framgangen vi hadde hatt siden det forrige møtet, og forklarte hva vi hadde gjort og hvordan det fungerte. Vi svarte også på spørsmål de hadde om det vi hadde gjort og fikk generell tilbakemelding og forslag til endringer og forbedringer som kunne gjøres, samt forslag til flere funksjonalitet. Vi stilte også spørsmål om ting som var uklare, spesielt på testkapitlene i rapporten. Vi stilte også spørsmål vi hadde generelt om gjennomføringen av prosjektet og hva oppdragsgiveren var ute etter.

Testgjennomgang

Etter de to brukertestene vi hadde gjennom prosjektet hadde vi også møter med produkteier og Pål Mjørlund hvor vi gikk gjennom tilbakemeldingene deres. På disse møtene kunne vi få bedre og mer grundig tilbakemelding enn i *sprint review* møtene ettersom brukerne hadde fått tid til å teste programmet på egenhånd. Gjennom møtene gikk vi gjennom testskjemaet testerne hadde fylt ut og de tilbakemeldingene de hadde, og hvordan vi kunne forbedre eller gjøre om på punktene de hadde hatt problemer med. For mer om brukertestene, se avsnitt 7.6.

3.2.2 Interne møter

Sprint Planning

Et *sprint planning* møte ble holdt den første mandagen i hver sprint. I løpet av disse møtene satte vi et mål for sprinten og la til nye *issues* i *backlog-en* på Jira. Deretter ble nye *issues* overført til *sprint backlog-en* på *scrum board-et*. Dersom var igjen *issues* fra forrige sprint, overførte vi disse også, hvis vi fortsatt følte at de var nødvendige. Når alle *issues* var valgt og lagt inn i *sprint backlog-en* hadde vi en planning poker seanse, en teknikk for tidsestimering brukt i smidige utviklingsmetoder [10], til å estimere tiden vi trodde vi ville bruke på d*issues-ene*.

Sprint Retrospective

Den siste fredagen i hver sprint, før møtene med oppdragsgiver og veileder hadde vi *sprint retrospective* hvor vi diskuterte internt i gruppen hvordan sprinten hadde gått. Vi diskuterte hva som hadde gått bra og dårlig i løpet av sprinten og kom med forslag til hva vi kunne gjøre annerledes eller bedre til neste sprint. Om vi hadde nådd målet til sprinten var også et tema på disse møtene, og om vi ikke hadde det, diskuterte vi årsakene til dette og hvordan vi kunne unngå at det ville skje igjen. Målet med disse møtene var at vi skulle bli flinkere til å organisere oss selv og å dele opp oppgaver som skulle gjøres, slik at ingen *issues* var for store, og at det ikke var for mange eller for få *issues* i løpet av sprinten. Etter disse møtene fikk vi mer innsikt i hvor stor en sprint burde være og lærte å sette mer realistiske mål for påfølgende sprinter.

Daglig Sprint

Hver arbeidsdag hadde vi et daglig sprint møte hvor vi snakket om hva vi hadde gjort dagen før og hva vi skulle gjøre videre. Dersom en eller flere gruppemedlemmer var ferdige med en *issue* viste vi en demo av arbeidet slik at de andre kunne se over koden og hvordan det virket i programmet. Andre på gruppen kunne komme med tilbakemelding på dette og si fra om noe trengte forbedringer, eller om det var godkjent og *issue-en* kunne bli markert som ferdig. Denne kvalitetssikringsprosessen er grundigere forklart i avsnitt 7.1. Under disse møtene kunne vi også spørre om hjelp med oppgaver fra andre gruppemedlemmer hvis det trengtes, og gikk deretter gjennom oppgaven som en gruppe til alle visste hvordan det kunne gjøres. Lengden på daglig sprint møtene varierte ut fra hvor mye som hadde blitt gjort siden sist, og om det ble noen diskusjon på hvordan en eller flere *issues* skulle implementeres.

Møte med veileder

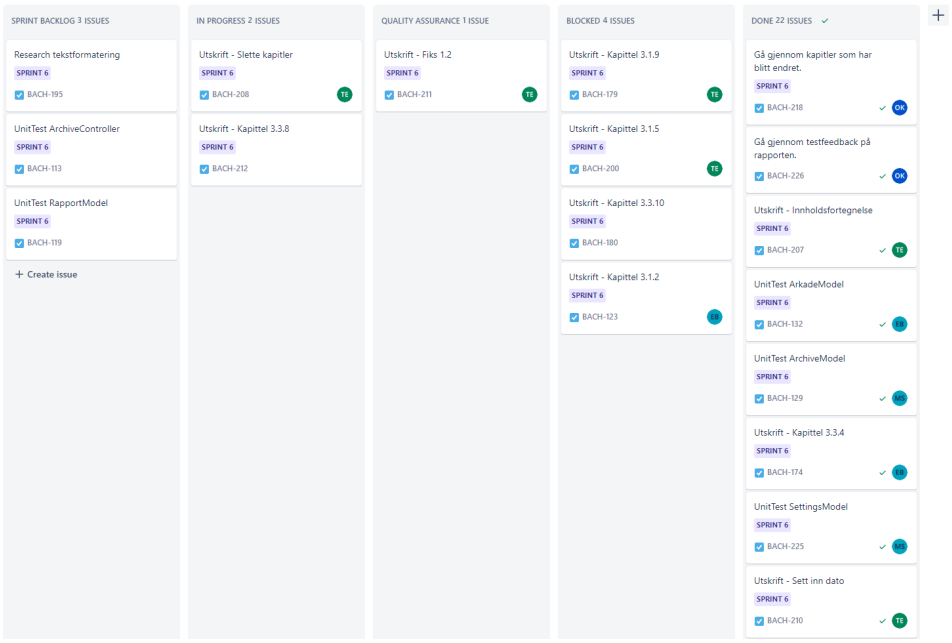
Vi hadde møte med veilederen vår Deepti Mishra en gang i uken på Teams for å diskutere gruppens fremgang, og for å spørre om råd. Til å begynne med holdt vi disse møtene på tirsdager, men etter hvert som vi kom lengre i prosjektet fant vi

ut at vi burde ha dem fredager i stedet. Dette bestemte vi fordi vi ville ha mer og bedre saker å fortelle på møtene, siden det var på slutten av uka. I tillegg kunne disse møtene bli holdt etter *sprint review* møtene annenhver uke, slik at vi kunne fortelle om tilbakemeldingene fra oppdragsgiveren rett etter vi hadde fått den til Deepti. I møtene diskuterte vi framgangen til gruppen og viste ofte en demo av applikasjonen. Deepti hjalp oss også med strukturen av rapporten. Hvis det var noen store hindringer eller ting vi lurte på rundt for eksempel sprint prosessen kunne vi spørre hun om råd for å hjelpe oss med dette.

3.3 Scrum Board

Som *scrum board* brukte vi Jira. Vi la inn alle *issues* som skulle bli implementert i *backlog-en* og flyttet disse til *sprint backlog-en* den sprinten hvor vi skulle fullføre dem. Når en av gruppemedlemmene valgte en *issue* å jobbe på flyttet de den til «In progress» og *issue-en* ble merket med navnet til den som jobbet på den. Deretter kunne oppgaven flyttes mellom de andre kolonnene i *scrum board-et* frem til den ble ansett som ferdig. De ulike kolonnene på *scrum board-et* var:

- Sprint Backlog
  - *Issues* som ikke er påbegynt og skal implementeres i løpet av gjeldende sprint.
- In Progress
  - *Issues* som jobbes på.
- Quality Assurance
  - *Issues* som er ferdige, men må bli sett over og testet av andre gruppemedlemmer før de kan bli markert som ferdig.
- Blocked
  - *Issues* som ikke kan bli gjort ferdig fordi de trenger tilbakemelding fra oppdragsgiver eller er avhengig av at andre *issues* blir ferdig først, før arbeidet kan fortsette.
- Done
  - Fullførte *issues*, market som ferdig.



Figur 3.1: Hvordan *scrum board-et* vårt så ut i Jira.

3.4 Sprint oversikt

Som beskrevet tidligere brukte vi scrum metoden med sprinter på to uker. Vi brukte denne metoden til alle arbeidsoppgaver relatert til koding og design av applikasjonen. Vi lagde et gantt-skjema over hvordan sprintene skulle være lagt opp, og hadde noen milepæler som bestemte når vi skulle ha en prototype klar til testing. Dette skjemaet er videre beskrevet i vedlegg A. Med unntak av den første og siste sprinten som skulle gå ut på henholdsvis planlegging/design og testing, planlagte vi de fleste sprintene i løpet av *sprint retrospective* møtene. *Backlog-en* og *sprint backlog-en* ble fylt opp i *sprint planning* møtet slik at vi kunne tilpasse etter tilbakemeldingen vi fikk i møtet med oppdragsgiveren, og det vi eventuelt hadde blitt enige om i løpet av forrige sprint.

Sprint 1 (01.02.21 - 12.02.21)

Den første sprinten gikk ut på å planlegge og designe applikasjonen, før vi begynte med kodingen. Vi lagde domenemodell, sekvensdiagram, *use cases* og valgte arkitektur for å bedre forstå hva vi skulle gjøre og hvordan vi skulle gjøre det. Deretter lagde vi en *wireframe* skisse av brukergrensesnittet slik at vi kunne begynne å implementere dette. I tillegg lagde vi et kode-skjelett av koden med alle klassene vi visste vi kom til å trenge, slik at vi hadde god oversikt og kunne begynne å jobbe med kodingen. Da disse målene var nådd begynte vi smått på mindre kodeoppgaver og undersøkte hvordan vi kunne implementere den ulike funksjonaliteten programmet skulle ha.

Sprint 2 (15.02.21 - 26.02.21)

Målet med sprint to var å få programmets viktigste funksjonalitet til å fungere fra start til slutt. Selve testingen og genereringen av testrapporten skulle ikke være ferdig, men man skulle kunne gå gjennom brukergrensesnittet for å kjøre tester og skrive rapport slik at vi kunne vise fremgangsmåten for dette til oppdragsgiver. I tillegg undersøkte vi og fant ut av hvordan testrapporten skulle genereres.

Sprint 3 (01.03.21 - 12.03.21)

Etter sprint tre skulle vi ha vår første brukertest, så hovedmålet med sprinten var å gjøre programmet klart for dette. Til å begynne med fokuserte vi på å gjøre god fremgang med genereringen av testrapporten og testing av dette. Mot slutten av sprinten fokuserte vi på å skrive et testskjema som skulle fylles ut av testerne, samt å sørge for at JAR filen kjørte og virket som den skulle.

Sprint 4 (15.03.21 - 26.03.21)

Fokuset med sprint fire var for det meste å komme så langt med testrapporten som mulig, slik at vi kunne fokusere på finpuss neste sprint. I løpet av den første uka testet brukerne programmet, og mange uventede feil dukket opp som måtte fikses. Dette tok lang tid og førte til at vi ikke hadde like god fremgang som vi håpet på. Vi hadde likevel fremgang og implementerte flere ulike funksjoner.

Sprint 5 (29.03.21 - 09.04.21)

Den nest siste sprinten ble brukt til å gjøre ferdig det aller meste av programmets funksjonalitet. Målet var å ha en nesten ferdig prototype som skulle bli testet av brukerne i begynnelsen av neste sprint. Vi fikk inn funksjonalitet for å pakke et ferdig testet uttrekk til AIP og funksjonalitet for å sette data inn i grafer til testrapporten. Vi jobbet også videre på resten av kapitlene til testrapporten. Til slutt refaktorerte vi koden ved å flytte kode som passet bedre inn i modell-klasser bort fra kontrolleren. Mer om refaktoreringsprosessen står beskrevet i avsnitt 7.5

Sprint 6 (12.04.21 - 23.04.21)

Den siste sprinten delte vi opp i to. Den første uken ble vi ferdig med så mye funksjonalitet og forbedringer som mulig, mens vi ventet på testresultatene fra den andre brukertesten. Den andre uka gikk dermed med på å se på tilbakemeldingen fra brukertesten og implementere de forbedringene vi kunne. Vi brukte også tid på å teste programmet på egenhånd, og å dokumentere koden.

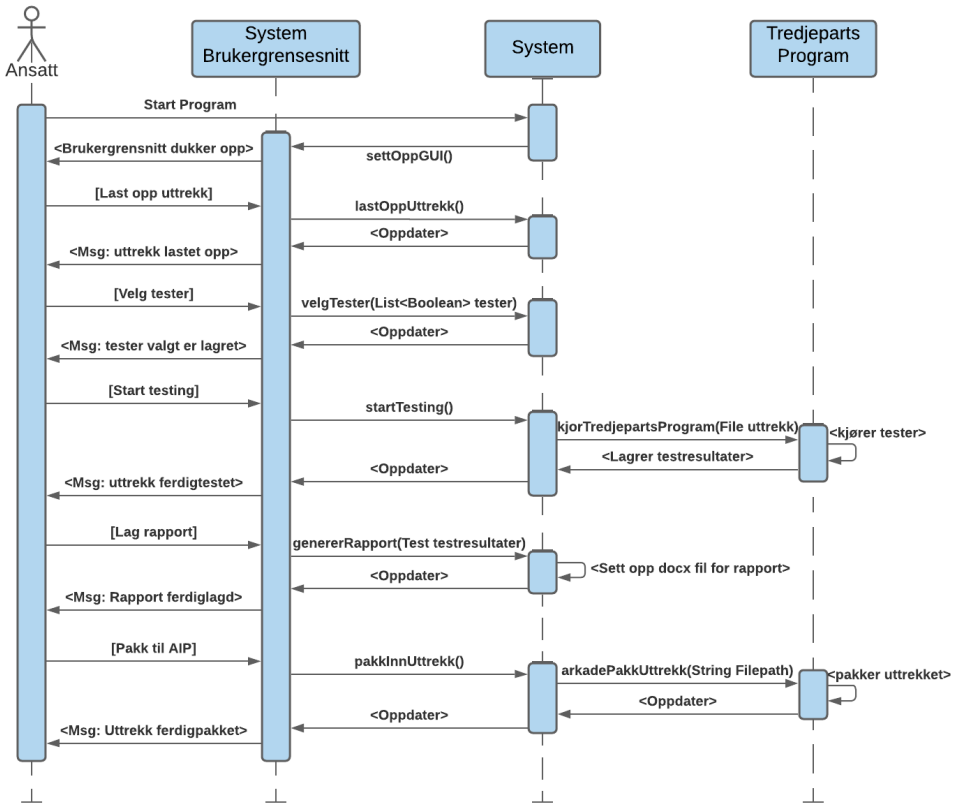
Kapittel 4

Teknisk design

I dette kapittelet vil tematikk rundt oppbyggingen, tankeprosessen og den initielle planen for utførelsen på kode-innholdet bli dekket. Første delen av kapittelet består av planleggingen av systemflyten, klassehierarki og metoder for å utvikle kommunikasjonen mellom klassene på den mest optimale måten. Den andre delen består av bibliotek som ble tatt i bruk, hva de ble brukt til og gunstigheten med å bruke dem i forbindelse med prosjektet vi hadde fått tildelt.

4.1 System sekvensdiagram

For å få en bedre forståelse av et system så er det viktig å sette seg inn i flyten mellom komponentene involvert. Det å sjekke livskretsen til hovedfunksjonaliteten av systemet hjalp oss med å danne et klart bilde av hvordan vi ønsket at komponentene til programmet skulle kommunisere med hverandre.



Figur 4.1: Systemflyten fra brukeren laster opp et arkivuttrekk til det er pakket inn igjen (laget i lucidchart).

Livskretsen til systemet i dette tilfellet begynner når brukeren starter opp systemet, og avslutter når uttrekket er pakket inn til AIP. Det er tre viktige komponenter som kommuniserer med hverandre gjennom systemets livssyklus: «System brukergrensesnittet», som er frontend delen, «System» som lytter til valg gjort og henter riktig data, og «Tredjepartsprogram» hvor testing og pakking av uttrekk blir gjort.

Ved oppstart av programmet starter grensesnittet for brukeren. Brukeren trykker på «Last opp uttrekk» og velger hvilken uttrekksmappe som skal gjøres klar for testing, mappen er da lagret i systemet og klar for testing. Deretter blir «Velg tester» knappen trykket på, som er en alternativ funksjonalitet i systemet. Eneste forutsetningen for å kunne kjøre testene på et uttrekk er at brukeren har lastet opp et uttrekk i systemet. «Start testing» velges deretter, og de valgte tredjepartsprogrammene for testing, i tillegg til egendefinerte XQueries, kjøres. Når testene er fullført velger brukeren «Lag rapport» og alle relevante testresultater blir hentet og lagt direkte inn i et DOCX dokument. Til slutt pakker brukeren uttrekket til AIP sammen med resultatene via AIP funksjonaliteten i Arkade verktøyet.

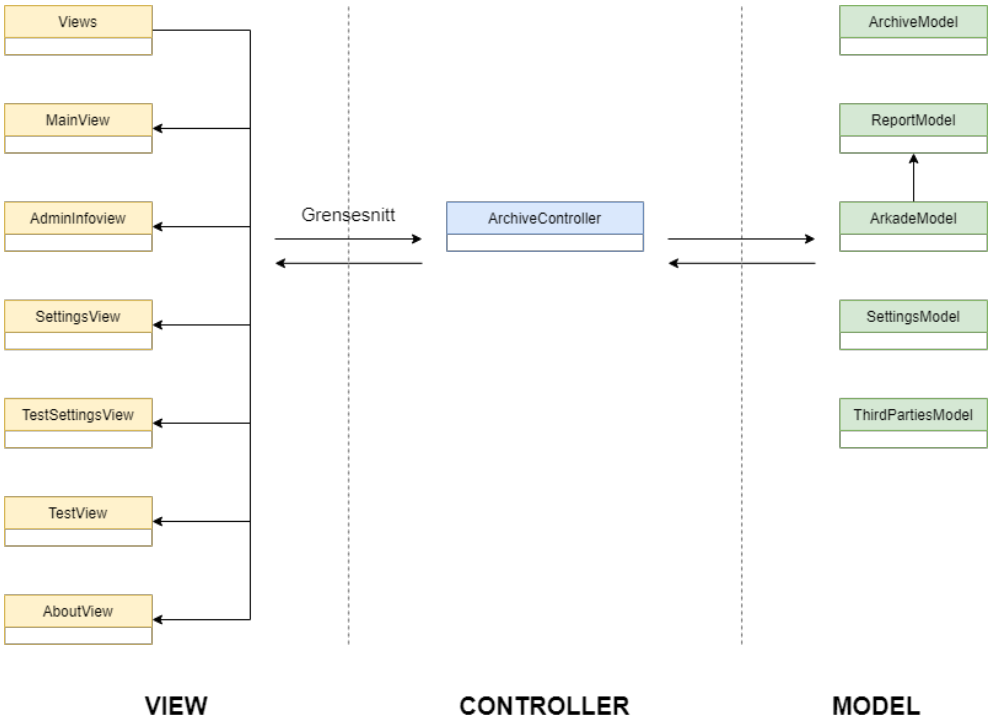
4.2 Arkitektur

Alle systemer som skal bli utviklet over lang tid trenger en solid basis, slik at bestemte retningslinjer for komponenter som blir laget og endret under utviklingen blir fulgt. En systemarkitektur er en form for forhåndslaget plan på hvordan de ulike delene av systemet skal kommunisere med hverandre, og varierer basert på hvilke krav som stilles til systemet. Fordelene med en systemarkitektur er blant annet [11]:

- Gir utviklerne en konkret ide om hvor et komponent hører til.
- Lettere å vedlikeholde systemet over lengre tid.
- Gjør systemet tilpasningsdyktig, siden det er mer oversiktlig å vite hvilke komponenter som kan endres på, og hvilke komponenter som blir påvirket.
- Reduserer dødtid og forvirring.

Det er mange populære og varierende systemarkitekturer som gir fordeler avhengig av hvilke krav og funksjonalitet systemet har, eller skal ha. For eksempel, et system med krav om modularitet vil trenge komponenter som ikke vil påvirke systemet i sin helhet. Et system med fokus på sikkerhet og sensitiv informasjon derimot, trenger et arkitekturmønster bygget på å forhindre lekkasje av denne informasjonen [12]. Avgjørelsen om hvilket arkitekturmønster som var best egnet til komponentene og utviklingsmiljøet ble bestemt basert på de funksjonelle- og ikke-funksjonelle kravene som ble stilt, disse står beskrevet i kapittel 2. Med alt dette tatt i betrakning så endte vi opp med å velge *model-view-controller (MVC)* mønsteret for dette prosjektet.

4.2.1 Model-View-Controller



Figur 4.2: Systemets/programmets klasser delt inn i MVC-mønsteret

For et prosjekt med fokus på modularitet og effektivitet under utviklingen var det viktig for oss at vi tok i bruk et arkitekturmønster som hovedsakelig ga oss disse fordelene. Dette gjorde MVC mønsteret til en god systemarkitektur for dette prosjektet. I MVC er det enkelt å delegere oppgaver til hver enkelt komponent, uten at de påvirker andre komponenter. Siden komponentene ikke påvirker hverandre er det også lettere å modifisere koden til en komponent uten at det må gjøres endringer i en annet [13]. Alt dette blir gjort ved å dele systemet inn i tre deler: modell, grensesnitt og kontroller, som vist i figur 4.2.

Grensesnittet inneholder all frontend koden, og er er alt brukeren ser og kan påvirke gjennom knapper, tekstbokser og liknende grensesnitt-elementer. I vårt program består grensesnittet av flere ulike paneler i et vindu laget med Java Swing, se avsnitt 6.1 for mer informasjon om implementasjonen av brukergrensesnittet. grensesnitt-elementene har hver sin lytter som følger med på om brukeren samhandler med dem. Disse lytterne kommunisererer med kontrolleren, og kontrollen oppdaterer grensesnittet med ny data om nødvendig. Grensesnittet trenger derfor aldri å vite om hvordan ting skjer, bare hvordan det skal vise den nye informasjonen og når det skal si fra til kontrolleren om brukerhandlinger [14].

Modellen inneholder alt av programmets data og logikk. Et system pleier som re-

gel å ha mange modeller, med sine særegne typer data som passer sammen med hverandre [14]. En modell trenger aldri å vite om hvordan informasjonen skal vises eller hvor tilleggsdata kommer fra, bare hvordan logikken utføres. I vårt tilfelle har vi valgt å dele modell-komponentene våre inn i fem deler: «ReportModel», «ArkadeModel», «SettingsModel», «ThirdPartiesModel», og «ArchiveModel». Klassene har som formål å lagre informasjon knyttet til rapporten som skal bli skrevet, innstillinger, integrering av tredjepartsverktøy og å vite hvilket uttrekk som har blitt lastet opp.

Til slutt har vi kontrolleren, som er bindeleddet mellom grensesnittet og modellene. Kontrolleren blir kalt på av grensesnittet gjennom sine lyttere og kaller deretter på en relevant modell som utfører logikken. Deretter oppdaterer den grensesnittet med relevant informasjon fra modellen.

4.3 Backend

Figur 4.3: Noen av teknologiene som ble tatt i bruk.

4.3.1 Utviklingsmiljø

Vi bestemte tidlig at systemet enten skulle bli skrevet i Java eller C++. Gruppen var godt kjent med begge språkene fra tidligere, som ga oss frihet til å bestemme på egenhånd uten at det ene valget ga oss en ulempe framfor det andre når det gjaldt kompetanse. Se mer om gruppens bakgrunn i avsnitt 1.6. En fordel med Java er måten den kjører koden på. Kildekoden blir kompilert til *bytecode*, som er Javas egen virtuelle maskin uavhengig av maskinkode [15]. *Bytecode* blir lagret i .class filer som blir kjørt via Java Virtual Machine (JVM). Dette vil si at

applikasjonen vil fungere på både Windows og Linux, noe som var et ønske fra oppdragsgiver, men ikke et krav. For å få samme muligheten i C++ må man bruke et tredjepartsverktøy [16]. Det mest anbefalte verktøyet vi visste om var CMake, et verktøy som vi hadde hørt en del negativt om i studiet. En annen fordel med Java var at noen gruppemedlemmer allerede hadde erfaring i hvordan man utvikler et grafisk brukergrensesnitt i det språket. Hvis vi skulle utvikle det i C++ måtte vi ha satt oss inn i et helt nytt bibliotek.

Noe annet vi diskuterte var hvilket utviklingsmiljø vi var mest komfortable med å bruke. Blant IDE-er var det to som stakk ut fra resten: Microsoft Visual Studio for C++, og Jetbrains IntelliJ IDEA for Java. Disse er IDE-er gruppen har anvendt tidligere som ga oss et godt referansepunkt for å analysere hva de kan tilby for utviklingen av systemet. Både Visual Studio og IntelliJ er gode utviklingsmiljøer som har en rekke med verktøy, mulighet for feilsøking, utvidelser og *git* kompatibilitet, noe som gjør dem til gode utviklingsmiljø å jobbe i. Der IntelliJ skinner mest er med dets allerede forhånds-pakkede innhold for Java bibliotek og JVM integrering som gjør det lett å begynne med arbeidet, i motsetning til Visual Studio som heller er rettet mot å være kompatibel med de fleste programmeringsspråk [17]. Etter diskusjonene ble det naturlig for oss å velge IntelliJ framfor Visual Studio, samt Java som programmeringsspråk for applikasjonen.

4.3.2 Tredjepartsprogrammer

En essensiell del av programmet vårt var at det skulle være mulig å integrere det med andre tredjepartsprogrammer. Fram til nå har disse programmene blitt brukt separat av de ansatte på fylkesarkivet, og blitt kjørt på manuelt for hver test. Vår oppgave har vært å binde disse sammen til ett enkelt system slik at programmet vårt får den nødvendige informasjonen fra hvert tredjepartsverktøy. Dette har blitt gjort gjennom å bruke kommandolinjen for å kalle på deres eksterne funksjonaliteter. Systemene vi brukte var BaseX, VeraPDF, Arkade5, KOST-Val og DROID.

Arkade5 er hovedverktøyet for å sikre validering av uttrekket og analysererer hvert uttrekk for å finne potensielle avvik og feil etter Noark-standarden. De tre andre verktøyene gjør sjekk av filer, som for eksempel gyldige filformat og PDF/A validering. Her kan det oppdages avvik som Arkade ikke fant i sine egne tester. Alle systemene utenom arkade returnerer resultatene på XML format, mens resultatene fra Arkade er i HTML format. For å kunne finne den relevante dataen til XML filene bruker vi et annet tredjepartsverktøy kalt BaseX, som kjører XQuery spørringene våre gjennom kommandolinjen. For mer om hvordan vi hentet data fra XQuery- og HTML filer, se implementasjonen i avsnitt 6.2.2, og avsnitt 6.3.4.

4.3.3 Biblioteker

Som nevnt tidligere er applikasjonen vår avhengig av å kommunisere med andre enheter, men de standard Java bibliotekene hadde ingen eksisterende funksjonali-

tet for redigering av filene vi brukte. Det vi trengte var en måte å håndtere og manipulere DOCX filer for rapporten og delkapitlene, og HTML filer for Arkade rapporten. For å løse disse to problemstillingene integrerte vi to eksterne biblioteker: Apache POI og jsoup. Jsoup håndterer alt som har med håndtering, manipulering og *parse-ing* av HTML filer og Apache POI tok seg av alt med henting, og redigering av DOCX filer. Senere måtte vi inkludere biblioteket Docx4j, siden Apache POI manglet funksjonalitet for å gjenkjenne og oppdatere innholdsfortegnelsen til testrapporten. Docx4j ga en løsning for å oppdatere innholdsfortegnelsen på slutten av den automatiske rapportskrivingsprosessen, etter overskriftene er skrevet gjennom Apache POI. For det grafiske brukergrensesnittet brukte vi Java Swing, som er en del av Javas grunnleggende biblioteker. Implementasjonen av disse bibliotekene blir beskrevet i kapittel 6.

For testing og kvalitetssikring av koden brukte vi JUnit rammeverket for enhetstester skrevet i Java. Enhetstestene ble implementert i form av testklasser isolert fra kildekoden, som blir kompilert og kjørt, og returnerer enten godkjent eller ikkegodkjent. I avsnitt 7.6.1 er det beskrevet mer om hvordan vi brukte enhetstester for kvalitetssikring.

Kapittel 5

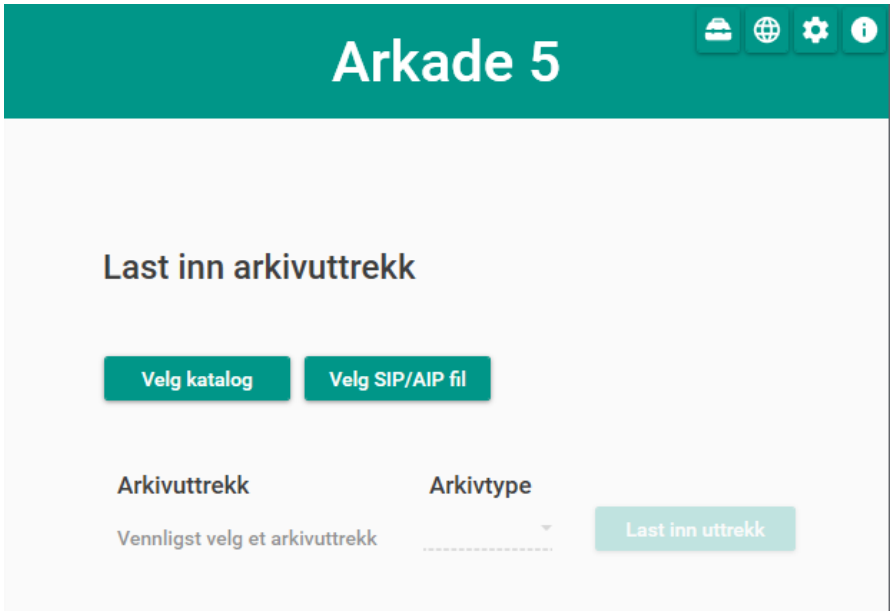
Grafisk design

I dette kapittelet vil prosessen rundt hvordan gruppen jobbet for å utvikle et grafisk brukergrensesnitt, inkludert skissering, layout, struktur og innføringer for å sikre god brukervennlighet bli dekket.

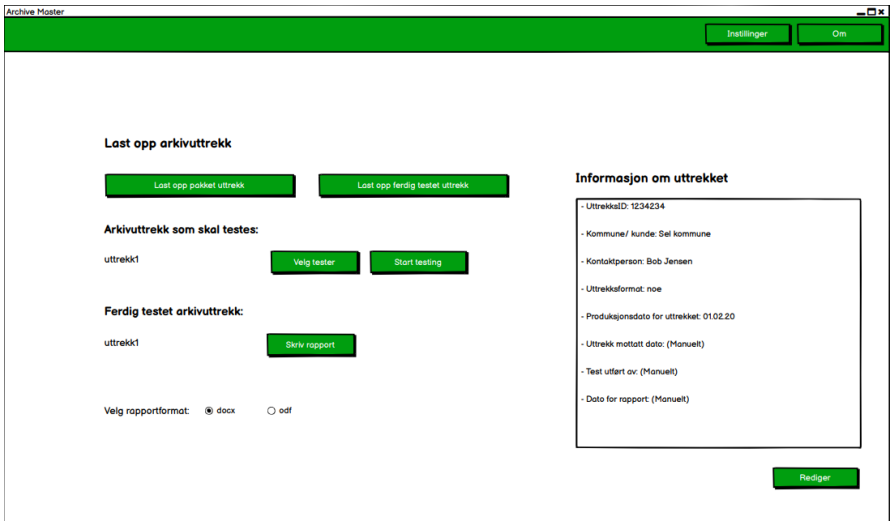
Brukergruppen for applikasjonen er kun de ansatte hos fylkesarkivet og de ønsket at det skulle være enkelt, intuitivt og effektivt å fullføre operasjonene som trengtes. Derfor er hovedprioriteten til brukergrensesnittet nettopp dette. Det ble ikke lagt så mye vekt på hvor avansert utseendet skulle være, men at funksjonaliteten var lett tilgjengelig og at man kan intuitivt komme seg igjennom prosessen av testing og validering av uttrekk. Videre i kapittelet vil det stå om skisseringsprosessen, oppsettet til de forskjellige sidene til applikasjonen og brukervennligheten. For å se mer om hvordan grensesnittet ble implementert, se avsnitt 6.1.

5.1 Wireframe design

Før vi begynte å skrive kode og lage et brukergrensesnitt til programmet lagde vi *wireframe* skisser av de fleste sidene brukergrensesnittet skulle ha. Selv om disse kunne- og ble endret senere, visste vi det ville være lurt å ha noen skisser og forslag til utseende først, slik at vi visste sånn cirka hvordan det kom til å se ut. Da vi designet utseendet og oppsettet til brukergrensesnittet tok vi mye inspirasjon fra Arkade5, fordi brukerne ville være vant med den stilen fra før av. Arkade sitt brukergrensesnitt var også veldig enkelt å bruke, og hadde nesten samme formål som vårt program. En av tingene vi ble enige om rett etter skissene ble laget var å endre fargen på knappene og feltet øverst på siden til mørkeblå. Vi var enige om at dette så bedre ut, samtidig som det skilte vårt program mer fra Arkade. Under i figur 5.1 og figur 5.2 vises forsiden til Arkade sitt brukergrensesnitt og *wireframe* skissen av vår forside. Resten av *wireframe* skissene ligger i vedlegg H.



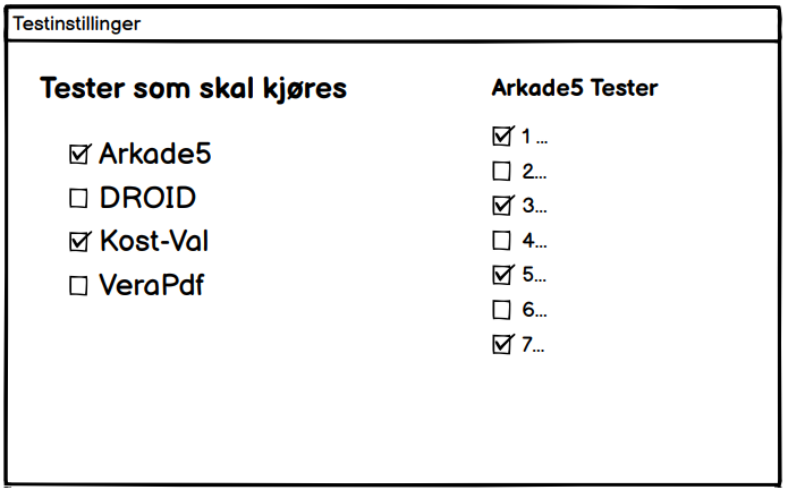
Figur 5.1: Forsiden til Arkade [18]



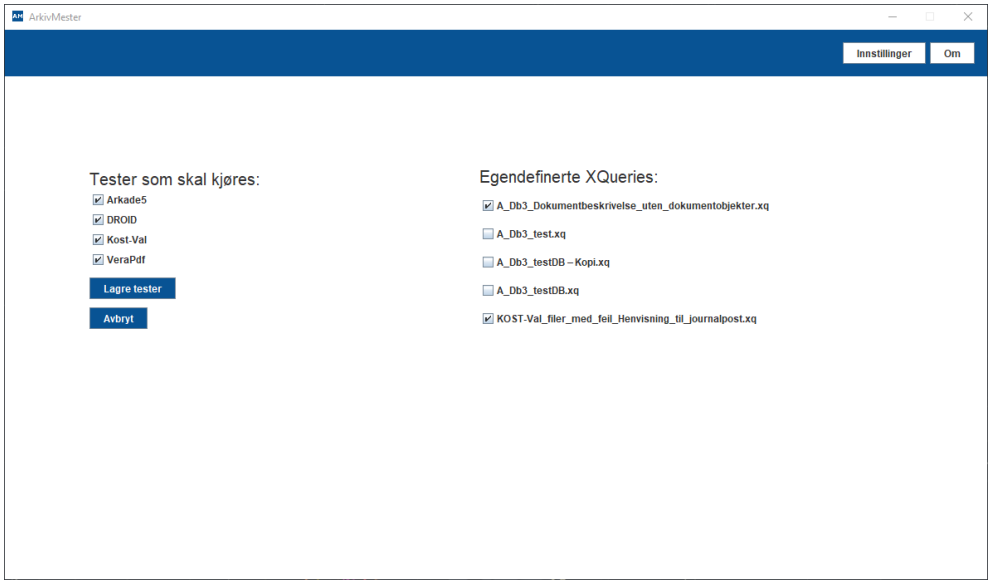
Figur 5.2: *Wireframe* skisse av forsiden til vårt brukergrensesnitt

Selv om noen knapper og funksjonaliteter var ganske likt Arkade så stilte oppgaven vår oss krav om å legge inn annen funksjonalitet som også måtte inn i brukergrensesnittet. Teststatus siden var spesielt viktig, samt oversikt over administrative data som skulle være på forsiden. Under designperioden var vi svært usikre på hvordan testinnstillinger siden, hvor brukeren valgte tester, kom til å se ut og hvordan den skulle fungere. Vi var også usikre på hva slags innstillinger vi

skulle ha i programmet før vi begynte og dermed er det disse sidene hvor det er størst forskjell mellom skissene og det ferdige produktet. figur 5.3 viser hvordan testinnstillinger skissen vår så ut og figur 5.4 viser hvordan den ser ut i det ferdige produktet.

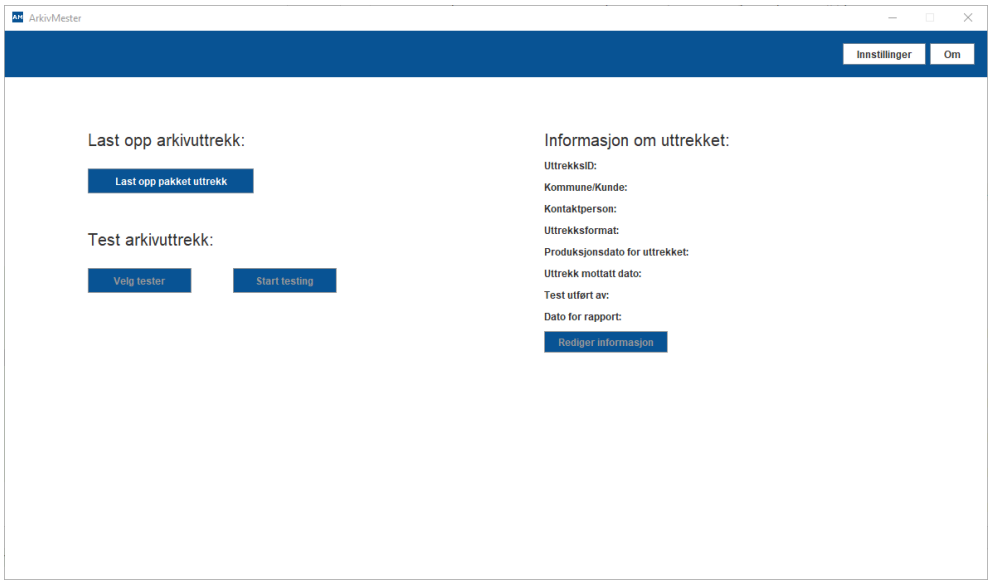


Figur 5.3: *Wireframe* skisse av testinnstillinger siden



Figur 5.4: Testinnstillinger siden til applikasjonen

5.2 Frontend



Figur 5.5: Forsiden til applikasjonen

Vi brukte Java Swing biblioteket til å utvikle det grafiske brukergrensesnittet. Biblioteket er et sett med brukergrensesnitt komponenter skrevet i Java, som gjør det enklere å utvikle skrivebordsapplikasjoner [19]. Disse komponentene er veldig lette og også plattformuavhengige. En fordel med disse komponentene er også hvor minimalistiske de er til standard, noe som gjør at de passer bra til vårt design som legger vekt på funksjonalitet i stedet for estetikk. Komponentene støtter verktøytips når man holder musepekeren over de eller når man velger den via tastaturet, samt at knappene forandrer utseende når de er aktivert eller deaktivert, og når de blir brukt. Dette er veldig bra for brukeropplevelsen fordi man får tilbakemeldinger på handlingene man gjør og det reduserer mulig forvirring. Videre om implementasjonen av brukergrensesnittet er forklart i avsnitt 6.1.

5.3 Layout

Brukergrensesnittet består av tre grunnelementer, toppfelt for navigasjonsbar med mulighet for andre navigasjonsknapper i fremtiden, hovedområdet på venstre side for de essensielle knappene for funksjonaliteten til applikasjonen og en liste over administrativ informasjon over det nåværende valgte uttrekket på høyre side.

Toppfeltet er navigasjonsfeltet øverst på siden og finnes på alle sider i applikasjonen. Per nå inneholder den en «innstillinger» knapp som åpner opp innstillinger siden hvor man kan forandre på filplasseringene til alle nødvendige tredjeparts programmer og kataloger for applikasjonen. Det er også en «om» knapp som åp-

ner opp en side hvor man kan se generell informasjon om applikasjonen.

Hovedområdet inneholder de viktigste elementene, som er knappene som konfigurer statusen til applikasjonen. Det er en knapp som gir brukeren mulighet til å velge hvilket uttrekk som skal testes, en knapp som åpner en side hvor man kan velge hvilke deltester som er aktivert og en knapp for å starte selve testingen av uttrekket.

Informasjonslisten inneholder den administrative informasjonen til uttrekket. I figur 5.6 vises listen etter man har lastet opp et uttrekk. Noen av feltene blir automatisk fylt inn med data fra uttrekket, mens andre er etterlatt for brukeren til å manuelt fylle ut. Denne listen gir altså de ansatte tilgang til å kunne redigere både de fylte feltene og de tomme feltene før de setter i gang hovedtesten og generering av testrapporten. Listen blir også med på testsiden som åpnes når man starter testing av uttrekket. Dette er fordi at disse testene kan vare i flere dager, samt at det kan være flere maskiner som kjører parallelt, noe som kan føre til forvirring om hvilke uttrekk som faktisk blir testet.

Figur 5.6: Listen over administrativ informasjon rett etter et uttrekk er lastet opp.

5.4 Struktur

Under er en liste over de viktigste sidene i applikasjonen vår. For en fullverdig oversikt se vedlegg B

- Forsiden er inngangspunktet for applikasjonen og inneholder konfigureringsknappene for uttrekkene og en liste over den administrative informasjonen. Noen av knappene gir tilgang til de andre sidene i applikasjonen.

Innstillinger gir brukeren muligheten til å endre på eller nullstille filplasseringene til tredjepartsprogrammene brukt av applikasjonen.

- Siden for å velge deltester har en liste over deltester og en liste over egendefinerte XQuery spørringer som brukeren kan aktivere eller deaktivere ved å benytte avkrysningsrutene ved siden av navnet på testene. Rediger administrativ informasjon knappen gir brukeren muligheten til å redigere den administrative informasjonen til uttrekket.
- Kjøring av uttrekkstest-siden inneholder en oversikt over uttrekkets status under testing og validering, se figur 5.7. Den består av fire paneler, inkludert toppfeltet og administrative informasjon som ikke blir vist i figuren til høyre. I det øverste panelet er det en liste over deltestene med status over om den er aktivert, ferdig, kjører, eller venter på å bli kjørt. I det nederste panelet er det en hovedstatus over applikasjonen. Den har en spinner animasjon under lengre operasjoner som testing, generering av rapport og pakking til API for å vite om applikasjonen har fryst eller ikke. Alle tre knappene er aktivert for brukeren når alle testene er fullført og vil gi muligheten til å lage testrapporten og pakke uttrekket til en AIP som inneholder denne rapporten. Man kan velge å teste et nytt uttrekk når som helst.

Figur 5.7: Teststatus siden som viser statusen av deltestene

5.5 Brukervennlighet

Når vi tenker på brukervennlighet for applikasjonen så tenker vi at den skal være lett og intuitiv å ta i bruk, effektiv og gir tilbakemelding på de feilene brukeren kan gjøre. Dette har vi implementert ved å gjøre grensesnittet så minimalistisk som mulig, uten å gjøre det for abstrakt. Det er verktøytips på alle knapper og knappenes utseendet forandrer seg når den blir brukt. Vi har også tenkt på det visuelle hierarkiet som oppstår. Elementene står på linje og er tydelig designet for at brukeren kan lett skille de i fra hverandre. Dette er gjort ved å endre størrelse- og tykkelse på skriften. Fargene som er brukt er applikasjonens blåfarge, hvitt og svart, som går igjen på alle sidene og elementene. Dette gir applikasjonen en tydelig identitet som gjør at man alltid vet når man er inne på applikasjonen og en god kontrast på for eksempel knappene. Hvis brukeren gjør noe feil som for eksempel å laste opp et uttrekk som ikke inneholder metadata filen eller at applikasjonen prøver å kjøre aktiverte XQuery spørringer som ikke finnes vil det komme opp et popup vindu med en relevant melding slik at brukeren lett ser hva som er galt og hva som kan gjøres for å rette opp i det.

Kapittel 6

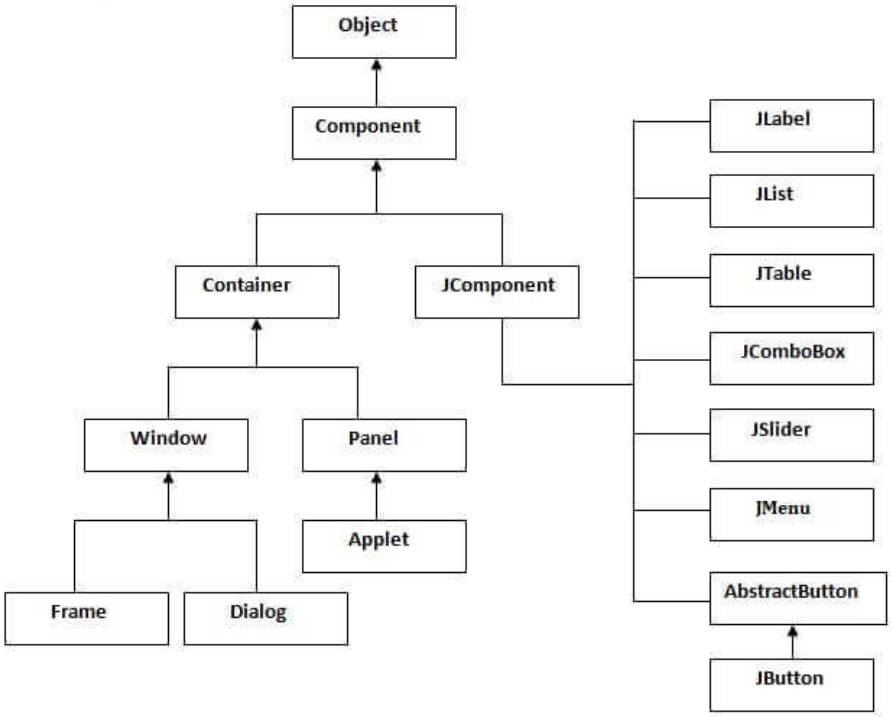
Implementasjon

I dette kapittelet vil implementasjonen av brukergrensesnittet, uttrekkstestene og XQuery spørringer, rapport genereringen og filstrukturen til prosjektet bli dekket.

6.1 Frontend

Applikasjonen har bare ett grensesnitt som er et grafisk brukergrensesnitt for Windows. Grensesnittet er utviklet i Java med hjelp av Java Swing biblioteket og er representasjonen av grensesnitt i Model-View-Controller mønsteret.

Java Swing er et komponentbasert bibliotek og i figur 6.1 kan man se klassehierarkiet for disse komponentene. I roten er det et vanlig Java objekt og jo lenger ned man går jo mer spesialisert blir dette objektet [19]. Applikasjon vår er bygd på ett «frame», som er applikasjonsvinduet. I dette vinduet har hvert grunnelement sitt eget panel, videre fylt med «JComponents». Disse panelene blir lagt inn i en «Container» så det blir lettere å bytte sider, eller «views», ved å fjerne innholdet på siden via konteineren, for å deretter fylle den igjen med nye relevante paneler.



Figur 6.1: Klassehierarkiet til komponenter i Java Swing [19]

Hver side har en «createAndShowGUI()» funksjon som skaper komponentene til siden og legger de til i vinduet. Måten komponentene er skapt er ved å først initialisere dem, så endre eller legge på egenskaper, deretter legge dem til vinduet. Hvert «view» objekt i applikasjonen vår arver fra en superklasse kalt «Views», som inneholder felles egenskaper som blant annet primærfargen og en «actionListener» implementasjon. I de kommende kodesnuttene vil det vises eksempler på hvordan «createAndShowGUI()» funksjonen ser ut.

I kodeliste 6.1 vises skapelsen av toppfelt komponenten som er med i alle sidene i applikasjonen. Konstruktoren tar inn *layout managers*, en klasse som håndterer hvordan innholdet til komponentene skal ligge i forhold til hverandre. I dette tilfellet ønsker vi en navigasjonsmeny bestående av knapper lagt horisontalt i panelet, så *FlowLayout* vil passe best her. Menyen skal starte i høyre side av toppanelet, som er grunnen til at layout klassen har «FlowLayout.RIGHT» konstanten i konstruktoren sin. Videre er det satt en grense på 10 pixler i alle retninger, som gir blankt tomrom rundt menyen, akkurat som *padding* egenskapen i CSS. «setUpTopPanel(topPanel)» funksjonen har ansvar for komponentene til toppanelet.

Kodeliste 6.1: Toppanel komponenten

```
JPanel topPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
```

```
topPanel.setBorder(new EmptyBorder(10, 10, 10, 10));
topPanel.setBackground(primaryColor);
setUpTopPanel(topPanel);
```

I kodeliste 6.2 ser man et eksempel på hvordan innholdet til de ulike panelene blir initialisert. I toppanelet blir det skapt to knapper med verktøytips, som deretter blir lagt til i panelet. Men hvordan setter vi opp koblingen mellom brukerhandlinger i grensesnittet med kontrolleren? Vi bruker observatør mønsteret som vil si at kontrolleren abonnerer på «views» objektene og når disse objektene oppdager en brukerhandling vil den gi beskjed til abonnentene sine, som i dette tilfellet er kontrolleren. Måten «views» objektene fanger opp handlingene er via Java sin *ActionListener*, implementert i superklassen «Views» som vist i kodeliste 6.3. Denne klassen innholder en liste som lagrer alle abonnentene sine. Hver gang en knapp blir trykket på vil dens lyttere kjøres og «actionPerformed()» funksjonen i superklassen vil bli kalt. Den bruker en «switch» kommando på knappens navn for å finne ut hvilken type handling det er og går deretter gjennom alle abonnentene og tilkaller den relevante funksjonen som abonnentene selv implementerer.

Kodeliste 6.2: setUpTopPanel(JPanel topPanel) funksjon for paneloppsett

```
JButton settingsBtn = new JButton("Innstillinger");
settingsBtn.addActionListener(this);
settingsBtn.setBackground(Color.WHITE);
settingsBtn.setToolTipText("Viser innstillinger.");

JButton aboutBtn = new JButton("Om");
aboutBtn.addActionListener(this);
aboutBtn.setBackground(Color.WHITE);
aboutBtn.setToolTipText("Viser informasjon om applikasjonen.");

topPanel.add(settingsBtn);
topPanel.add(aboutBtn);
```

Kodeliste 6.3: Observer listen og begynnelsen av «actionPerformed» funksjonen

```
protected final List<ViewObserver> observers = new ArrayList<>();
public void actionPerformed(ActionEvent e) {
    String buttonName = e.getActionCommand();

    switch (buttonName) {
        case "Last inn pakket uttrekk":
            for (ViewObserver obs : observers)
                obs.uploadArchive();
            break;
        case "Test nytt uttrekk":
            for (ViewObserver obs : observers)
                obs.newTest();
            break;
        case "Rediger informasjon":
            for (ViewObserver obs : observers)
                obs.editAdminInfo();
            break;
```

I kodeliste 6.4 blir det vist en del av grensesnittet kontrolleren implementerer. Grensesnittet inneholder funksjonssignaturene som «views» kaller på og spesifiserer hvilken handling det er snakk om når det har skjedd en brukerhandling. Disse funksjonene er implementert i kontrolleren som en helt vanlig kontrollerfunksjon. På denne måten vil «views» og kontrolleren være skilt så mye som mulig, der «views» ikke bryr seg om hvordan handlingene er håndtert. De gir bare beskjed om at det har skjedd. Kontrolleren kan deretter implementere og utføre alle handlingene uavhengig av hvordan «view» objektene er utviklet. Det eneste negative med denne implementasjonen er at hver gang et «view» objekt blir skapt i kontrolleren, så må den eksplisitt abonnere på den via et funksjonskall. «View» objektet må også arve fra superklassen «Views», noe som gjør at den ikke kan arve fra noen andre klasser siden Java bare støtter arving fra én klasse. I vårt tilfellet så var dette ikke et problem.

Kodeliste 6.4: Begynnelsen av ViewObserver grensesnittet

```
public interface ViewObserver {
void uploadArchive();
void newTest();
void editAdminInfo();
}
```

I kodeliste 6.5 vises det hvordan hoved konteineren er skapt og hvordan den legger til grunnelementene. «f.getContentPane()» returnerer innholds konteineren til vinduet, videre settes *Layout manageren* og bakgrunnsfargen. Når grunnelementene blir lagt til, blir det også spesifisert med hjelp av *BorderLayout* klassen hvor de skal ligge i vinduet. Toppanelet skal øverst, spesifisert av «Borderlayout.NORTH» konstanten, infopanelet skal på høyresiden, spesifisert av «Borderlayout.EAST» konstanten og hovedpanelet vil fylle ut så mye den kan på venstresiden og tvinge infopanelet til å være like bredt som sin minimumsbredde.

Kodeliste 6.5: Legge grunnelementer til hovedkonteineren

```
        Container container = f.getContentPane();

        container.setLayout(new BorderLayout(15, 10));
        container.setBackground(Color.WHITE);

        container.add(mainPanel);
        container.add(infoPanel, BorderLayout.EAST);
        container.add(topPanel, BorderLayout.NORTH);
```

6.2 Tester

6.2.1 Kjøring av testene

Alle testene kjøres gjennom kommandolinjen via Java-koden i programmet vårt. Klassen «ThirdPartiesModel» ble laget for å kjøre og håndtere alle tredjeparts-testverktøy vi var avhengige av å brukei dette prosjektet. I denne klassen er det separate funksjoner for hver enkelt test som skal kjøres, altså Arkade, VeraPDF, KOST-Val, og DROID. Disse funksjonene får medsendt en «path» variabel fra kontroller-klassen som inneholder lokasjonen for filene som skal testes. Etter hvert som programmet ble større og det ble flere og flere filstier som programmet trengte for å kjøre-og få output fra testene bestemte vi å lage en konfigurasjonsfil som inneholdt alle filstiene programmet trengte for å blant annet kjøre testene. Dette gjorde det mye lettere og mer dynamisk å kjøre dem, se avsnitt 6.4 for mer om filstrukturen. Funksjonene får derfor medsendt ett *properties* objekt som brukes til å hente filstiene til de ulike testverktøyene, samt lokasjonen hvor outputen fra disse blir lagret.

Resten av arbeidet som må til for å kjøre testene er å sette sammen en kommando som vil bli kjørt i kommandolinjen. Denne blir kjørt ved hjelp av en *ProcessBuilder* som kjører *cmd.exe*. Til å begynne med ble dette gjort i de respektive testfunksjonene, men etter hvert som vi fant ut at noen av testene trengte flere ulike kommandoer for å kjøres korrekt ble det for mye duplisert og unødvendig kode i disse funksjonene. Derfor ble «runCMD()» funksjonen laget. Denne funksjonen lager en CMD prosess og kjører denne, i tillegg til å skrive ut outputen fra kommandolinjen til konsolen. På de fleste tester fører utskriften til at brukeren lettere kan se hvor langt det er igjen av testen, eller hvis en eller flere av testene ikke virker som de skal.

Kodesnuttene i kodeliste 6.6 og kodeliste 6.7 viser hvordan testene blir kjørt via «runArkadeTest()» og «runCMD()» funksjonene. I «runArkadeTest()» blir først alle filstier samlet, og satt sammen til en kommando som skal kjøres i «runCMD()» funksjonen. Alle testfunksjonene fungerer på samme måte som arkade funksjonen, filstiene vil komme til å være annerledes, og noen av de andre testene krever flere ulike kommandoer, men ellers fungerer de på samme måte.

Kodeliste 6.6: Funksjon som kjører Arkade testen

```
public void runArkadeTest(File path, Properties prop) throws IOException {

    //String with path to arkadeCli
    String cd = cdString + prop.getProperty("arkadePath") + "\"";
    //Path to output folder where test report gets saved.
    String outputPath = "\"" + tempFolder + archiveName + "\\Arkade\\Report\"";
    //Path to temp folder where temporary data about the tests gets stored.
    String tempPath = "\"" + tempFolder + archiveName + "\\Arkade\"";


    //Run ArkadeCli through command line.
```

```
    runCMD(cd + " && arkade test -a " + path + " -o " + outputPath + " -p " +
         tempPath + " -t noark5");

}
```

Kodeliste 6.7: Funksjon som kjører CMD med medfølgende kommando

```
    private void runCMD(String command) throws IOException {
        //Creates a process to run a command in cmd.
        ProcessBuilder cmdBuilder = new ProcessBuilder(
                cmd, "/c", command);
        Process p = cmdBuilder.start();
        //Gets output from cmd and prints it.
        BufferedReader r = new BufferedReader(new InputStreamReader(p.
             getInputStream()));
        String line = r.readLine();
        while(line != null) {
            System.out.println(line);
            line = r.readLine();

        }
        r.close();
    }
```

Testene blir kjørt gjennom kontrolleren når brukeren trykker på «Start testing» knappen i brukergrensesnittet. I og med at brukeren kan ekskludere eller inkludere tester som skal kjøres sjekker programmet først hvilke tester som skal kjøres, brukergrensesnittet blir også oppdatert ut ifra dette.

For å kunne kjøre KOST-Val, VeraPDF, og DROID testene var vi avhengige av å pakke ut TAR mappen arkivuttrekkene kommer i. Arkade gjorde dette på egenhånd, men for de andre testene var vi avhengige av å ha tilgang på filer inne i uttrekket og måtte dermed få programmet til å pakke ut TAR mappen selv. Vi valgte å bruke 7-Zip for å pakke ut arkivet i og med at de ansatte på fylkesarkivet brukte dette programmet fra før av. Arkivuttrekket blir pakket ut via 7-Zip sin kommandolinje funksjonalitet på samme måte som vi kjørte testene. Funksjonen som pakker ut uttrekket blir automatisk kjørt når brukeren starter testene, men fordi 7-Zip ikke kan pakke ut samme uttrekk flere ganger uten at det blir slettet først lagde vi også funksjonalitet i funksjonen som pakker ut uttrekket som sjekker om filene eksisterer og sletter dem dersom de gjør det. Selv om testene til vanlig bruk bare trenger å bli kjørt en gang ble det mye lettere for oss da vi slapp å slette det utpakkede uttrekket manuelt hver gang vi testet programmet. Vi følte likevel det ville være nyttig i tilfelle noe gikk galt under testingen og testene måtte bli kjørt på nytt. Algoritmen i «unzipArchive()» funksjonen bruker *Files.walkFileTree* fra *java.nio.file* biblioteket for å først gå gjennom og slette alle filer i arkivet som er pakket ut, så alle mapper disse filene lå i. I kodeliste 6.8 blir det vist hvordan koden først sjekker om det utpakkede uttrekket finnes, hvordan dette blir slettet, og hvordan «runCMD()» blir brukt for å kjøre 7-Zip via kommandolinjen.

Kodeliste 6.8: Funksjon som kjører 7-Zip

```
public void unzipArchive(File path, Properties prop) throws IOException {
        File unzipped = new File(tempFolder + archiveName + archiveName);
        //If unzipped folder exists, deletes it.
        if(unzipped.exists()){
            Path directory = unzipped.toPath();
            Files.walkFileTree(directory, new SimpleFileVisitor<>() {
                @Override
                public FileVisitResult visitFile(Path file, BasicFileAttributes
                     attributes) throws IOException {
                    Files.delete(file);
                    return FileVisitResult.CONTINUE;
                }

                @Override
                public FileVisitResult postVisitDirectory(Path dir, IOException exc
                     ) throws IOException {
                    Files.delete(dir);
                    return FileVisitResult.CONTINUE;
                }
            });
        }

        //String with path to 7-Zip location.
        String cd = cdString + prop.getProperty("7zipPath") + "\"";
        //Run VeraPDF from command line
        runCMD(cd + " && 7z x " + path + " -o\"" + tempFolder + archiveName + "\" -
             r");
    }
```

6.2.2 XQuery tester

Programmet var avhengig av å kjøre XQuery spørringer på de fleste av XML filene i uttrekkene for å hente data til testrapporten, i tillegg til å hente administrativ data som blir vist i brukergrensesnittet, og egendefinerte spørringer om brukeren ønsket dette. Alle spørringer, utenom de egendefinerte ble kjørt gjennom «runBaseX()» funksjonen i «ThirdPartiesModel», denne funksjonen blir vist i kodeliste 6.9. Funksjonen får medsendt filstien til XML filen som skal bli spurt, navnet på selve XQuery spørringen som skal bli kjørt, og et *properties* objekt som blir brukt for å hente filstien til mappen hvor alle XQueries er lagret lokalt på brukerens datamaskin. BaseX blir kjørt via kommandolinjen via en *ProcessBuilder* på samme måte som testfunksjonene nevnt i forrige delkapittel. «runBaseX()» funksjonen skriver også ut resultatene fra XQuery spørringene til en tekstfil.

Kodeliste 6.9: Funksjon som kjører BaseX via kommandolinjen

```
    public List<String> runBaseX(String xml, String xqName, Properties prop) throws
         IOException {
        String xq = "\"" + prop.getProperty("xqueryExtFolder") + "\\" + xqName + "
            \"";
        String temp = prop.getProperty(tempFolderKey) + "\\xqueryResult.txt";

        String pwd = cdString + prop.getProperty(basexPathKey) + "\"";
```

```
        List<String> result = new ArrayList<>();

        ProcessBuilder baseXBuilder = new ProcessBuilder(cmd, "/c", pwd + " &&
             basex -o \"" + temp + "\" -i \"" + xml + "\" " + xq);

        try {
            Process p = baseXBuilder.start();
            p.waitFor();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }

        File xqueryResult = new File(temp);
        try (InputStream bytes = new FileInputStream(xqueryResult)) {

            Reader chars = new InputStreamReader(bytes, StandardCharsets.UTF_8);

            try (BufferedReader r = new BufferedReader(chars)) {
                String line = r.readLine();
                while (line != null) {
                    result.add(line);
                    line = r.readLine();
                }
            }
        }

        return result;
    }
```

De fleste XQuery spørringene fikk vi ferdiglaget av oppdragsgiver, men vi ble nødt til å endre på noen av dem, og også lage noen helt nye for å finne den informasjonen vi trengte. Med unntak av spørringene som hentet den administrative dataen, ble alle de ulike spørringene samlet i en liste, vist i kodeliste 6.10. XQuery spørringene vi brukte i programmet ble oppkalt etter kapittelet i testrapporten hvor dataen de hentet skulle bli brukt, derfor har alle utenom én navn som for eksempel «3.1.13». Dette gjorde det lett å finne ut hvilken spørring data skulle hentes fra da vi skrev koden for de ulike kapitlene.

Kodeliste 6.10: Liste med spørringer

```
Map<String, List<String>> xqueryResults = new HashMap<>();
 List<String> headerNumbers = Arrays.asList(
"3.1.3", "3.1.5_1", "3.1.5_2", "3.1.7_1", "3.1.7_1b", "3.1.7_2", "3.1.9_1",
"3.1.11b", "3.1.13_1", "3.1.13_2", "3.1.14_1", "3.1.14_2", "3.1.20", "3.1.21",
"3.1.23_1", "3.1.23_2", "3.1.23_3", "3.1.26_1", "3.1.26_2","3.1.27_1","3.1.27_2",
"3.2.1_1", "3.2.1_2", "3.2.1_3", "3.3.1", "3.3.2_1", "3.3.2_2", "3.3.2_3",
"3.3.3_1", "3.3.3_2", "3.3.4", "3.3.5_1","3.3.5_2", "3.3.5_3", "3.3.6",
"3.3.7", "dokumentmedium"
);
```

Kodeliste 6.11: Kjøring av spørringer

```
for(String s :headerNumbers) {
   xqueryResults.put(s, getEmptyOrContent(testArkivstruktur, s));
```

```
}
```

Kodesnutten i kodeliste 6.11 viser hvordan alle XQuery spørringer fra listen i kodeliste 6.10 blir kjørt gjennom «getEmptyOrContent» funksjonen. Resultatene fra alle spørringene blir deretter samlet i en *map*. *Map-en* «xqueryResults» blir brukt i «ReportModel» klassen ved å hente resultatet fra en og en spørring som vist i kodeliste 6.12.

Kodeliste 6.12: Hente data fra en spesifikk spørring

```
List<String> parts = xqueriesMap.get("3.1.3");
```

Funksjonen «getEmptyOrContent()» som blir vist i kodeliste 6.13 er en hjelpefunksjon i kontrolleren som kjører «runBaseX()» funksjonen fra «ThirdPartiesModel» og returnerer resultatet fra en og en spørring dersom de fikk noen resultater. Funksjonen sjekker også om den aktuelle XQuery filen eksisterer på datamaskinen og kommer med en melding om den ikke gjør det.

Kodeliste 6.13: Funksjon som henter data fra XQuery spørringer

```
    private List<String> getEmptyOrContent(String xml, String header) {
        String empty = "empty";
        File xquery = new File(settingsModel.prop.getProperty("xqueryExtFolder") +
              "\\" + header + ".xq");
        if(xquery.exists()) {
            try {
                List<String> para = thirdPartiesModel.runBaseX(
                        xml,
                        header + ".xq",
                        settingsModel.prop);

                if(para.isEmpty()) {
                    return Collections.singletonList(empty);
                }

                return para;
            } catch (IOException e) {
                mainView.exceptionPopup("BaseX kunne ikke kjøre " + header + " .xq
                     filen. Sjekk om filen eksisterer");
                return Collections.singletonList(empty);
            }
        }else {
            mainView.exceptionPopup("BaseX kunne ikke kjøre " + header + " .xq
                 filen. Sjekk om filen eksisterer");
            return Collections.singletonList(empty);
        }

    }
```

Egendefinerte spørringer

En av funksjonalitetene som var viktigst for oppdragsgiver var å kunne kjøre egendefinerte XQuery spørringer på uttrekk. Det vil si spørringer som ikke er en del av den vanlige testrapporten, men som likevel kan være nyttige for testingen av uttrekket. Egendefinerte XQueries blir valgt på samme måte som brukeren velger hvilke tester som skal kjøres, via «Velg tester» siden. Resultatet fra disse blir skrevet til .txt filer som blir plassert sammen med testrapporten for uttrekket. Egendefinerte spørringer må være plassert i en egen mappe av brukeren, og filstien til denne mappen må legges til i innstillinger menyen. De blir deretter hentet av programmet og en og en spørring kommer i «velg tester» menyen hvor de kan velges av brukeren.

I den første iterasjonen av denne funksjonaliteten bestemte vi oss for å la brukeren velge en og en XQuery spørring som skulle kjøres ved å hake dem av i brukergrensesnittet, for å deretter skrive navnet på den aktuelle XML filen som skulle testes i en tekstboks ved siden av. I samtaler med oppdragsgiver fikk vi forklart at de ønsket å kunne kjøre en spørring på flere ulike XML filer samtidig for å blant annet sammenligne disse, dette var ikke mulig med vår gjeldende kode så vi ble nødt til å komme frem til en annen løsning.

Løsningen vi kom fram til var å bruke databasefunksjonaliten til BaseX til å lage en database som inneholdt alle XML filene som kunne bli testet. Dette medførte at brukeren ikke lenger trengte å skrive inn i brukergrensesnittet hvilken XML fil som skulle bli testet, og åpnet muligheten for å kjøre samme spørring på flere XML filer. Brukeren trengte bare å hake av XQuery spørringen som skulle kjøres i brukergrensesnittet. For at dette skulle fungere måtte spørringen som skulle kjøres bli oppdatert for å kjøres på XML filene i databasen. Et eksempel på dette er vist under i kodeliste 6.14 hvor databasen «arkivmester» åpnes og XML filen «arkivstruktur» blir hentet derfra. Deretter kjøres spørringen som vanlig på denne filen.

Kodeliste 6.14: Linje som bestemmer hvilken XML fil spørringen skal kjøres på

```
db:open("arkivmester", "arkivstruktur.xml")
```

Databasen blir satt opp på via kommandolinjen med «runCMD()» funksjonen. Programmet får også tak i filstien til alle XML filene som skal legges til i databasen og legger dem til, noe som også gjøres via «runCMD()» funksjonen. Programmet henter de spørringene brukerne har valgt via «getCustomXqueries()» og kjører dem via «runCustomBaseX()» som kjører spørringene via BaseX sin kommandolinjefunksjonalitet på samme måte som «runBaseX()» funksjonen beskrevet i avsnitt 6.2.2. Dette blir vist i kodeliste 6.15 etter at uttrekket er ferdig testet og rapporten er generert blir databasen slettet slik at den ikke bruker unødvendig

minne og gjør programmet tregere.

Kodeliste 6.15: Sette opp og legge til i databasen

```
runCMD(pwd + " && basex.bat -c \"CREATE DB " + dbName + "\"");

String cmdOpen = " && basex.bat -c \"OPEN ";
String cmdAdd = "; ADD \"";

runCMD(pwd + cmdOpen + dbName + cmdAdd + xml1 + "\"\"");
runCMD(pwd + cmdOpen + dbName + cmdAdd + xml2 + "\"\"");
runCMD(pwd + cmdOpen + dbName + cmdAdd + xml3 + "\"\"");
```

6.3 Rapport-automatisering

Som hovedoppgaven til prosjektet tilsier, så går mye av oppgaven ut på å automatisere rapportskrivingsjobben for å effektivisere arbeidet til fylkesarkivet. Dette blir gjort ved å hente all den nødvendige dataen fra HTML- og XML filer som blir funnet av tredjepartsprogrammene vi bruker, for så å skrive resultatene fra disse inn i en sluttrapport gjennom vårt system. Det var ikke satt noe krav om å gjøre ferdig hele automatiseringen i sin helhet. For det første er noen av uttrekkene utdaterte, som gjør at måten strukturen er bygd opp på i de ulike uttrekkene varierer. For det andre krevde visse resultater fra noen deler av kapitlene videre sjekking av XML filer for å finne ut hvordan disse resultatene oppstod, og for det tredje var det usikkerhet fra oppdragsgiver sin side om hvilke data som skulle bli hentet i noen innviklede deler av noen kapitler. Med det i bakhodet så la vi fokus på å gjøre det enkelt å redigere tekst som skulle inn i rapporten gjennom funksjonskall slik at det senere kunne bli laget nye kapitler, eller at kapitlene som allerede fantes i rapporten kunne redigeres.

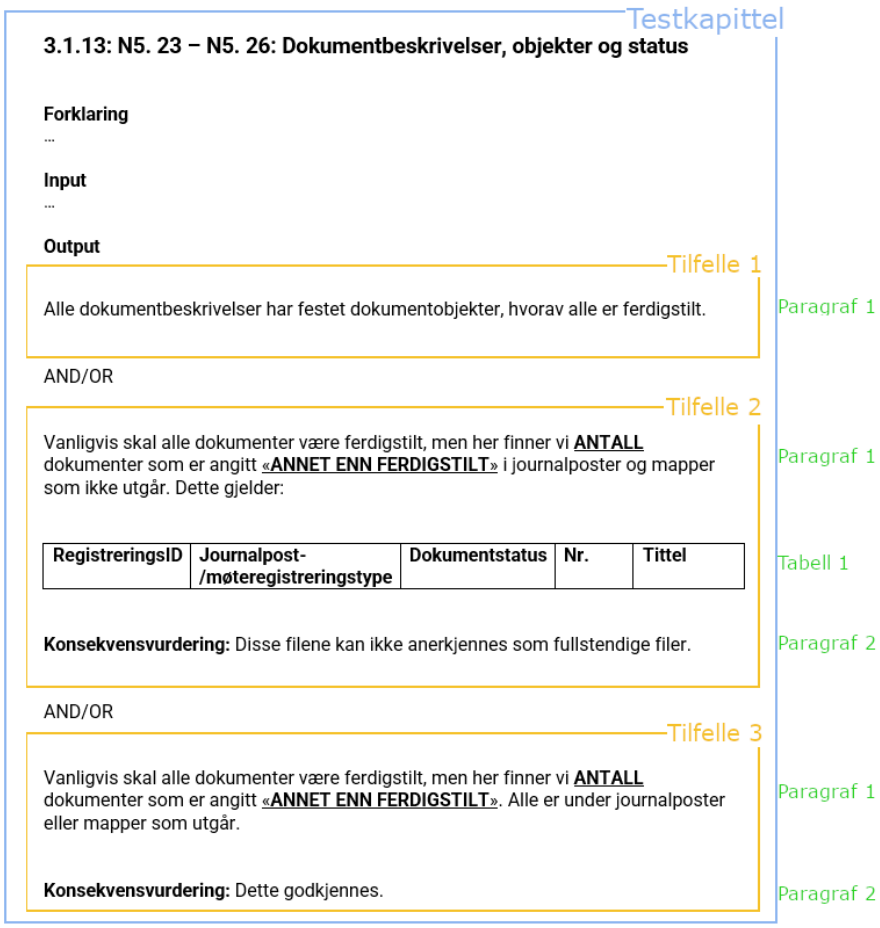
Til å begynne med hadde vi en rapportmal, en fasit på hvilke resultater som var forventet på ett av uttrekkene hadde fått tildelt, og en Word fil for hvert kapittel som inneholdt tekst og midlertidige ord for tekst som vi selv skulle legge inn etter hvert. Rapportmalen består av kapitlene som skal fylles inn, samt noe tekst til hvert kapittel som skulle gi oss en ide om hvordan sluttrapporten skulle bli seende ut. Kapittelfilene hadde beskrivelse av hva som trengtes i kapittelet, hvor vi kunne hente dataen fra, og hva som skulle skrives til sluttrapporten. Til slutt hadde vi fasiten, som var en tidligere versjon av sluttrapporten. Den ble brukt til å sjekke om vår sluttrapport hadde innhold som samsvarte med de riktige resultatene.

For å håndtere denne problemstillingen ble det laget to primære modell-klasser som tok seg av å hente riktig innhold, og å plassere dette på rett plass i sluttrapporten: «Reportmodel» og «Arkademodel». «Arkademodel» tar seg av å hente data fra arkaderapporten, og er en utvidelse av «Reportmodel». Vi går videre inn

på den i avsnitt 6.3.4. «Reportmodel» og dens oppgave om å håndtere DOCX filer er det vi kommer til å se nærmere på i dette delkapittelet.

6.3.1 Struktur

Vi gikk gjennom en del iterasjoner gjennom utviklingsfasen på hvordan dataen skulle bli håndtert på best mulig måte. En av metodene vi startet med å prøve ut var å ha alt rapportinnholdet lagret i en XML fil under ressursene i prosjektet vårt. Denne metoden ble rotete å bruke, og under møtene med oppdragsgiver kom det frem at de ikke syntes dette så ut som den beste løsningen, derfor endte vi opp med å forkaste den, og vi begynte å bruke Apache POI biblioteket istedenfor. Ulempene med XML og gunstighetene med Apache POI kommer vi mer inn på i avsnitt 8.2.2 i konklusjonen.



Figur 6.2: Splitting av innholdet til en DOCX fil med delkapittel 3.1.13 som eksempel

Dette er et eksempel på DOCX filen til kapittel 3.1.13 med struktur lik rapport klassen i den «utvidede domenemodellen» i figur 2.2. Når vi henter info fra disse

filene så henter vi som regel alt etter «Output» og bruker «AND/OR» for å skille mellom tilfellene som kan forekomme. For å få delt opp avsnittene på en logisk måte bestemte vi oss for å lage tre klasser inni «Reportmodel» som tok seg av kapitlene. Kapitlene inneholdt en liste med tilfeller, tilfeller som inneholdt en liste med innhold, og til slutt innholdet som var tilegnet enten paragraf, tabell, eller graf avhengig av hvilke av disse som var i det respektive tilfellet. Tanken bak strukturen forble den samme som tidligere, men mer fleksibel. Det å splitte de ulike delene inn i tre klasser gjorde det enklere å håndtere Apache POI funksjonaliteter som tabell og graf, og det ble gunstigere å legge til, eller slette, visse deler av et innhold.

6.3.2 Oppsett

«Reportmodel» starter prosessen sin med å kalle på funksjonen «getDocument()» (vist i kodeliste 6.17) for å hente DOCX fil til rapportmalen og kapittelbeskrivelsene. Funksjonen har filplasseringen som parameter, og dersom den får treff, og filen er av typen DOCX som klassen «XWPFDocument» tar i bruk, så vil den bli lagret i variabelen «Document», som vist i kodeliste 6.16. Vi henter rapportmalen med funksjonen, og videre iterer systemet igjennom denne teksten, og henter kun kapitteloverskriftene.

Kodeliste 6.16: document er en static variabel som holder på fildata.

```
document = getDocumentFile(templateFile);
```

Kodeliste 6.17: Funksjon som henter DOCX filer fra filplassering.

```
public XWPFDocument getDocumentFile(String filepath) {
    try (
        InputStream fis = getClass().getResourceAsStream(filepath)
    ) {
        return new XWPFDocument(fis);
    } catch (IOException | NullPointerException e) {
        return null;
    }
}
```

«HeadersData», som vist i kodeliste 6.18, er en liten klasse som har som oppgave å identifisere hvilket nummer en overskrift tilhører. I rapportmalen finner vi mange underkapitler som for eksempel «3.1.13», som Apache POI (med sine begrensinger) ikke er i stand til å hente nummeret til. «HeadersData» håndterer dette problemet ved å ta i bruk «compareName», som også er vist i kodesnutten. Den tar seg av å sammenligne alle overskriftene som blir sendt inn for å deretter lagre dem i en liste og en *map*. Listen inneholder eksisterende overskriftstyper, og sammenligner dem med med parameteren til funksjonen, for å sjekke om disse er ulike hverandre. *Map-en* oppdaterer tallrekken hver gang funksjonen blir kalt på.

Kodeliste 6.18: Funksjon som husker nummer for hvert kapittel.

```
public static class HeadersData {
    private final List<String> name;
    private final Map<String, Integer> headerMap;

    public void compareName(String other) {

        if(headerMap.computeIfPresent(other, (k, v) -> v+1) != null) {
        int temp = name.size()-1;
        String currentName = name.get(temp);
        while(!other.equals(currentName)) {
            headerMap.put(currentName, 0);
            currentName = name.get(--temp);
        }
        } else {
            headerMap.put(other, 1);
            name.add(other);
        }

        while(name.size() > headerMap.size()) {
            name.remove(name.size()-1);
        }
    }
```

Dersom vi for eksempel bruker «compareName()» for første gang, og sender inn «overskrift1» som har verdi «1» som parameter vil tallrekken i *map-en* oppdatere seg til å være «1». Om vi gjør dette en gang til med en annen overskrifttype, «overskrift2», som også har verdien «1» som parameter vil tallrekken oppdatere seg til å være «1, 1». Systemet vil da vite at disse to overskriftene har kapittelbeskrivelsene sine liggende i DOCX filene for henholdsvis kapitlene 1 og 1.1. Som vist i figur 6.3 så har filene navn som gjør det enkelt for oss å gjøre tallrekkene om til disse filnavnene.

2.docx	12.03.2021 14.58	Microsoft Word-d...	58 kB
3.1.1.docx	12.03.2021 14.58	Microsoft Word-d...	56 kB
3.1.2.docx	19.04.2021 14.54	Microsoft Word-d...	63 kB
3.1.3.docx	05.05.2021 20.13	Microsoft Word-d...	57 kB
3.1.4.docx	12.03.2021 14.58	Microsoft Word-d...	56 kB
3.1.5.docx	19.04.2021 14.54	Microsoft Word-d...	85 kB
3.1.6.docx	12.03.2021 14.58	Microsoft Word-d...	56 kB
3.1.7.docx	05.05.2021 20.13	Microsoft Word-d...	57 kB
3.1.8.docx	05.05.2021 20.13	Microsoft Word-d...	57 kB
3.1.9.docx	02.05.2021 16.55	Microsoft Word-d...	85 kB
3.1.10.docx	12.03.2021 14.58	Microsoft Word-d...	58 kB
3.1.11.docx	05.05.2021 20.13	Microsoft Word-d...	57 kB
3.1.12.docx	05.05.2021 20.13	Microsoft Word-d...	57 kB
3.1.13.docx	05.05.2021 20.13	Microsoft Word-d...	58 kB
3.1.14.docx	05.05.2021 20.13	Microsoft Word-d...	58 kB

Figur 6.3: DOCX filer for kapittel beskrivelsene vi fikk tilsendt fra oppdragsgiver

6.3.3 Velge riktig tilfelle, og sette inn innhold

Alle tilfeller et kapittel-objekt inneholder er inaktive, og vil ikke bli skrevet til sluttrapporten. Dette lar oss velge ut på egenhånd hvilke tilfeller som skal inkluderes gjennom funksjonskall. I de fleste scenarioer sjekkes det om det er noe resultat fra XQuery spørringene, og basert på om dette resultatet er tomt eller ikke så velges en av to tilfeller som blir satt i rapporten. Dette blir illustrert i kodeliste 6.19 med «if-else» utsagnet. Her hentes det fra XQuery resultatet for delkapittel «3.1.21», og om dette ikke er tomt så kaller vi på «setNewInput()», som redigerer innholdet av typen paragraf. «setNewInput()» har tre parametere: kapittelnummer, innhold og hvilket tilfelle som skal med i rapporten. Paragrafer vil ha ord med store bokstaver, fet skrift og understrek for å signalisere at de er midlertidige. Systemet vil lete etter disse midlertidige dataene og bytte de ut med innholdsparameteren.

Kodeliste 6.19: Eksempel på to forskjellige tilfeller som kan bli hentet basert på om resultatet er tomt eller ikke.

```
para = xqueriesMap.get("3.1.21");
if(para.get(0).equals(EMPTY)) {
    setNewInput(Arrays.asList(3, 1, 21), Collections.emptyList(), 0);
}
else {
    setNewInput(Arrays.asList(3, 1, 21), Collections.emptyList(),1);
}
```

Funksjonen som blir brukt dersom delkapittelet inneholder en tabell som skal fylles med data er «insertTable()», som vises i eksempelet i kodeliste 6.20. Denne funksjonen setter innholdet inn i de riktige cellene i tabellen. For at programmet skal kunne sette data inn i tabellen så må et aktivt tilfelle være tilstede i det eksakte kapittelnummeret tabellen er i. Av den grunn så brukes «setNewInput()» også i dette tilfellet, men med tomt innhold, før «insertTable()» blir kalt på. «insertTable()» har to parametere; kapittelnummer og en liste med innholdet som skal inn i tabellen. Disse har samme formål som i «setNewInput()», og mengden med kolonner til tabellen er forhåndsbestemt allerede i DOCX-filen der selve tabellen blir hentet fra.

Kodeliste 6.20: Eksempel på et tilfelle hvor det legges data inn i en tabell.

```
List<String> dokumentstatus = arkadeModel.getTableDataFromHtml("N5.15", 4);

setNewInput(Arrays.asList(3, 1, 8), Collections.emptyList(), 0);
insertTable(Arrays.asList(3, 1, 8), dokumentstatus);
```

Den siste essensielle funksjonen som brukes til genereringen av sluttrapporten er «insertGraph()», vist i kodeliste 6.21, som legger data inn i en eksisterende graf. Funksjonen tar imot fire parametre, der de to første og den siste har de samme oppgavene som parameterene i «setNewInput()» i kodeliste 6.19. «getCols()» har

et annet formål, som er å returnere antallet kategorier som grafen skal ta i bruk. På den måten kan funksjonen vite i hvilket rad i grafen dataene skal legges inn i.

Kodeliste 6.21: Eksempel på et tilfelle hvor data skal legges inn i en graf.

```
para = xqueriesMap.get("3.1.5_1");
if(!para.get(0).equals(EMPTY)) {
    insertGraph(Arrays.asList(3, 1, 5), splitIntoTable(para), getCols(para), 0);
}
```

6.3.4 Arkaderapport

Det viktigste tredjepartsverktøyet som brukes til å validere ett arkivuttrekk er Arkade5. Arkade produserer en rapport basert på innholdet i uttrekket i form av en HTML fil. Rapporten beskriver datastrukturen til uttrekket, for eksempel antall dokumenter, manglene referanser, og detaljerte beskrivelser av avvikene fra Noark-standarden [4] uttrekket kan ha. Rapporten blir delt opp i flere kapitler, som alle tester ulike ting, og inneholder tabeller med de ulike avvikene og annen nyttig informasjon om uttrekket. Videre i denne seksjonen vises det hvordan vi bruker denne rapporten i applikasjonen vår, og hvordan vi henter ut relevant data. I figur 6.4 ser man et eksempel på hvordan to kapitler av arkaderapporten ser ut.

N5.51 - Klassereferanser

Type: Innholdskontroll

Resultater

Ingen avvik funnet.

N5.59 - Antall journalposter

Type: Innholdskontroll

Resultater

Lokasjon	Melding
	Periodeskille er skarpt og antallet journalposter i arkivstrukturen er ikke likt det i offentlig og løpende journal
	Antall journalposter funnet i arkivstrukturen: 610
offentligJournal.xml	Antall journalposter dokumentert i offentlig journal: 710
loependeJournal.xml	Antall journalposter dokumentert i løpende journal: 710

Figur 6.4: Utdrag fra Arkaderapporten

«ThirdPartiesModel» klassen kjører Arkade på uttrekket gjennom kommandolinjen som beskrevet i avsnitt 6.2.1. Arkaderapporten blir lagret i brukermappen til applikasjonen, se avsnitt 6.4.1 for mer om denne mappen, og så lest inn av «ArkadeModel» igjennom «getFileToString()» funksjonen, se kodeliste 6.22. Metoden

vi bruker for å hente data fra arkaderapporten er å først lese den inn via *FileReader* funksjonen til et «StringBuilder» objekt, for å deretter søke gjennom dette objektet for informasjonen vi trenger. Dette beholder HTML strukturen, samtidig som vi får muligheten til å lese innholdet som tekst. Dette blir brukt senere til å *parse* teksten til *Document* format, som andre funksjoner nevnt senere i kapittelet (kodeliste 6.23) trenger for å kunne søke etter HTML elementer. Parameteret til «getFileToString()» er et properties-objekt som lar oss få tak i filplasseringen til arkaderapporten. Filen blir altså lest linje for linje og lagt inn i «ArkadeModel» sin *StringBuilder*, som holder på teksten fra rapporten for senere bruk.

Kodeliste 6.22: Hente HTML som tekst

```
1  StringBuilder htmlRawText = new StringBuilder();
2
3  public boolean getFileToString(Properties prop){
4          htmlRawText = new StringBuilder();
5          // Folder path: Arkade/output
6          filePath = prop.getProperty("tempFolder") + "\\" + prop.getProperty("
               currentArchive") + "\\Arkade\\Report"; //#NOSONAR
7          try {
8              // Dir: "arkadeOutput" folder
9              File dir = new File(filePath);
10             // Get first file in dir
11             filePath = filePath + '\\' + Objects.requireNonNull(dir.list())[0];
12         } catch (Exception ex) {
13             System.out.println("Get first file in Arkade/output. Error: " + ex.
                   getMessage()); //NOSONAR
14             return false;
15         }
16         try (FileReader fr = new FileReader(filePath);
17              BufferedReader br = new BufferedReader(fr)) {
18
19             String val;
20             while ((val = br.readLine()) != null) {
21                 htmlRawText.append(val);
22             }
23         } catch (Exception ex) {
24             System.out.println(ex.getMessage());
25             return false;
26         }
27         return true;
28     }
```

Når «ReportModel» klassen trenger å hente data fra arkaderapporten bruker den «ArkadeModel» sin funksjonalitet. Grunnen til at logikken for arkaderapporten er isolert i sin egen modell-klasse er fordi det gjør implementeringen av sluttrapporten og arkaderapporten uavhengige av hverandre. Dette er for å sikre «ReportModel» klassen hvis Arkade blir oppdatert i fremtiden, i tillegg til at koden blir mer lesbar og oversiktlig.

Implementasjonen for å hente data fra *StringBuilder* objektet starter med «getDataFromHtml()» funksjonen vist i kodeliste 6.23, som brukes til å filtrere ut data-

en vi trenger. Den benytter jsoup biblioteket som *parse-r StringBuilder* objektet fra «ArkadeModel», til et *Document* objekt i og med at jsoup kun kan behandle slike objekter [20]. Vi søker så etter «index» parameteren i *Document* objektet, som spesifiserer det øverste elementet til avvikstabellen. Tabellen til det øverste elementet blir så hentet, og delt opp i rader før den blir sendt videre til «getCellsInTable()» funksjonen, se kodeliste 6.24. Radene blir splittet opp til kun cellene er igjen og lagt til i «htmlTable» listen. Hvert andre element i «htmlTable» er XML filen som har avvik, og tilhørende melding om disse avvikene fra arkaderapporten.

Kodeliste 6.23: Finner kapittel i rapporten og kaller på «getCellsInTable()»

```
public List<String> getDataFromHtml(String index){

    List<String> htmlTable = new ArrayList<>();
    Document doc = Jsoup.parse(htmlRawText.toString());

    if(doc.getElementById(index) == null) {
        System.out.println("No index: " + index);
        return htmlTable;
    }
    Element element = doc.getElementById(index).parent();
    org.jsoup.select.Elements rows = element.select("tr");
    getCellsInTable(htmlTable, rows);
    return htmlTable;
}
```

Kodeliste 6.24: Henter cellene til avviktabellen fra «getDataFromHtml()»

```
public void getCellsInTable(List<String> htmlTable, Elements rows){

    for(org.jsoup.nodes.Element row :rows){
        org.jsoup.select.Elements columns = row.select("td");
        boolean firstColumn = true;

        for (org.jsoup.nodes.Element column:columns) {
            if (firstColumn){
                firstColumn = false;
                htmlTable.add("" + column.text());
            }
            else {
                htmlTable.add(column.text());
            }
        }
    }
}
```

Originalt skulle «Arkademodel» bare lese fra HTML filen og hente resultatene fra denne. Resten skulle bli håndtert i «ReportModel», men etter hvert som flere tester ble lagt til ble det for mye og komplisert kode. Teksten i arkaderapporten kan også variere fra uttrekk til uttrekk, som krevde at vi måtte feilsøke og oppdaterte koden flere ganger. Mye av koden i «ReportModel» gikk ut på å bruke data fra arkaderapporten, og av denne grunnen er det mange hjelpefunksjoner i «ArkadeModel»

som hjelper programmet med håndtere innholdet i rapporten. Funksjonene gir en detaljert tilbakemelding hvis noe går galt, og bruker hverandre så mye som mulig for å kutte ned på kodekompleksiteten. Det er viktig at funksjonene er enkle å bruke og kun henter spesifikk tekst fra tabellen, slik at ansatte hos fylkesarkivet kan gjenbruke koden for nye oppgaver. De mest brukte funksjonene som henter data fra «getDataFromHtml()» er «getSpecificValue()», som vises i kodeliste 6.26, og «getTotal()», som vises i kodeliste 6.27.

Funksjonen «getSpecificValue()» tar imot en «index»-variabel som blir brukt av «getDataFromHtml()» for henting av kapittel, og «containsValue»-variabelen som er teksten cellen det søkes i må inneholde for å bli returnert. Koden går gjennom avvikstabellen og sjekker om noen av cellene inneholder teksten fra «containsValue»-variabelen med bruk av den innebygde Java-funksjonen *contains()*. Alle elementer som inneholder denne teksten blir returnert som en liste. I for eksempel kodeliste 6.25 henter vi kun avvik som går på datoer uttrekket inneholder.

Kodeliste 6.25: Eksempel hvor det hentes dato avvik fra HTML tabellen

```
List<String> invalidDates = arkadeModel.getSpecificValue("N5.03", "Date value.");
```

Kodeliste 6.26: Henter avvik meldinger som innholder spesifikk tekst fra rapport.

```
public List<String> getSpecificValue(String index, String containsValue){
    List<String> htmlTable = new ArrayList<>();
    for(String i : getDataFromHtml(index)){
        if(i.contains(containsValue)){
            htmlTable.add(i);
        }
    }
    if (htmlTable.isEmpty()) {
        System.out.println(index + " Can't find deviation
        with: " + containsValue);
    }
    return  htmlTable;
}
```

«Total» er et tall som blir nevnt i nesten alle avvikstabeller i arkaderapporten, og beskriver totalt antall avvik av en type. Denne verdien blir brukt til å finne data til flere ulike kapitler i sluttrapporten. For å hente tall som «Antall dokumentfiler» eller «Total» blir funksjonen «getTotal()» brukt, denne funksjonen er vist i kodeliste 6.27, og et eksempel på bruksmåten blir vist i kodeliste 6.28. Funksjonen «getSpecificValue()» finner riktig verdi, og dersom denne verdien finnes, blir den send videre til en annen funksjon, «getNumberInTextAsString()», som tar tallet ut av teksten og sender det videre til «getTotal()». «getTotal()» skal bare hente en unik verdi fra tabellen, så den godtar kun et element fra «getNumberInTextAsString()», og gjør inneholdet om til en «Integer» slik at det kan brukes av de funksjonene som trenger dette istedenfor tekst. Hvis teksten «Total» blir sendt med som parameter i «getTotal()», godtar den første element som blir funnet av «get-

NumberInTextAsString()» i og med at «Total» er en unik verdi som alltid kommer først i tabellene hvor den finnes. Hvis ingen, eller mer enn et elemet blir funnet, returnerer «getTotal()» «-1», som blir brukt i programmet for å sjekke om noe gikk galt under hentingen av en verdi.

Kodeliste 6.27: Funksjonen «getTotal».

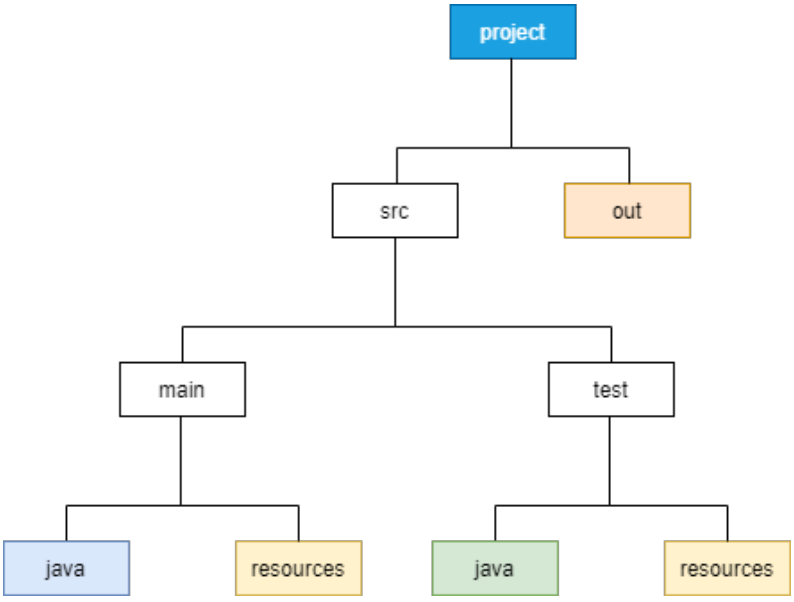
```
public Integer getTotal(String index, String containsValue){

    List<String> tmp = getNumberInTextAsString(index,containsValue, ":");
    if(tmp.size() == 1 || (!tmp.isEmpty() && containsValue.equals(TOTALT))){
        return Integer.parseInt(tmp.get(0));
    }
    else if (tmp.isEmpty()){
        System.out.println("   " + index + " Has  0 elements") ;
    }
    else{
        System.out.println("   " + index + " Has " + tmp.size() +
        " elements. Only TOTALT will get first element if several elements") ;
    }
    return -1;
}
```

Kodeliste 6.28: Henter antall Journalstatus med status Arkivert.

```
int arkivert = arkadeModel.getTotal("N5.22", "Journalstatus: Arkivert - Antall:");
```

6.4 Filstruktur



Figur 6.5: Filstrukturen til applikasjonen

Filstrukturen til applikasjonen er som vist i figur figur 6.5. Den er definert etter Maven-standarden som gjør at vi letter kan skille kildekode fra kompilert kode, altså .java- og .class filer. Standarden gjør det også lett å skille kildekode relatert til applikasjonen og kildekoden relatert til testing fra hverandre.

Her er en forklaring for hver katalog i filstrukturen.

- **src** er rotkatalogen for all kildekode og testkode.
- **out** er rotkatalogen for all kompilert kode og ressursfiler for den kjørbare applikasjonen.
- **main** er rotkatalogen for applikasjonens kildekode og nødvendige ressursfiler som rapportmalen og konfigurasjonsfilen.
- **test** er rotkatalogen for all testkode, som JUnit tester og deres nødvendige ressursfiler.
- Videre i både **main** og **test** katalogene er kildekode og ressursfilene adskilt ved å plassere de i forskjellige underkataloger. **java** er for kildekoden og **resources** er for ressursfilene.

For å bygge den kjørbare applikasjonen og kompilere kildekoden har vi brukt IntelliJ IDEA sin innebygde kompilator og byggverktøy. Grunnen til at vi ikke bruker Gradle eller Maven byggverktøyene er fordi at applikasjonen er ren Java og trenger derfor ikke slik støtte [21]. Vi bruker også allerede IntelliJ IDEA som vårt utviklingsmiljø og støtten den gir er perfekt for både inkrementell bygging og bygging fra bunnen av.

6.4.1 Brukermappe

Et problem vi hadde i starten av prosjektet var når vi skulle implementere brukerinnstillinger i applikasjonen. Vi tenkte først at vi kunne legge konfigurasjonsfilen, som inneholdt disse innstillingene, i ressurskatalogen og la programmet lese og skrive fra den så brukeren kan endre og lagre innstillingene sine. Men vi fant fort ut at når man kompilerer og bygger koden til JAR filen, som er den kjørbare applikasjonen, så klarer den ikke å skrive til ressursfilene som ble bygget inn i filen. Derfor bestemte vi oss for å flytte denne konfigurasjonsfilen sammen med andre kjøretids-kataloger og filer til Windows brukermappen. Dette er et populært sted for kjørbare applikasjoner for å lagre sine konfigirasjonsfiler og midlertidige filer for lesing og skriving. For å lære mer om filstrukturen til applikasjonens katalog i brukermappen, se vedlegg B.

Kapittel 7

Kvalitetssikring

For å sikre at koden og programmet vårt var av høy kvalitet, og nådde ulike standarder i bransjen brukte vi flere metoder og verktøy. I tillegg utførte vi grundig testing av programmet, både på egenhånd og via brukertester.

7.1 Code review

Etter hvert som utviklingsoppgaver ble ferdiggjorte ble de alltid satt i QA-kolonnen i *scrum board-et*. Oppgavene måtte deretter bli sett over og godkjent av de andre gruppemedlemmene i et møte for å eventuelt bli markert som ferdig i *scrum board-et* og *push-et* til *developer branch-en*. *Code review* møtene ble slått sammen med daglig scrum møtene våre hvis noen hadde oppgaver som var plassert i QA-kolonnen. Dersom utviklere var usikre på om oppgaven var løst på en god måte eller bare ville ha tilbakemelding på hvordan det var gjort eller så ut, var dette en mulighet for alle til å ha en diskusjon rundt dette.

Under møtene hvor vi hadde *code review*, gikk ett og ett gruppemedlem gjennom *issues* de hadde plassert i QA-kolonnen. Utvikleren delte da skjermen sin og viste hva som hadde blitt gjort. Utviklerne kom med tilbakemeldinger på hverandres kode, og kunne spørre om forklaringer på kode som var vanskelig å forstå. Vi kunne komme med forslag til hvordan koden kunne blir forbedret, eller om vi så en annen, bedre måte å løse et problem på som ville gjøre den mer effektiv. Dersom gruppen under møtene ble enige om at alt så bra ut ble oppgaven ansett som ferdig.

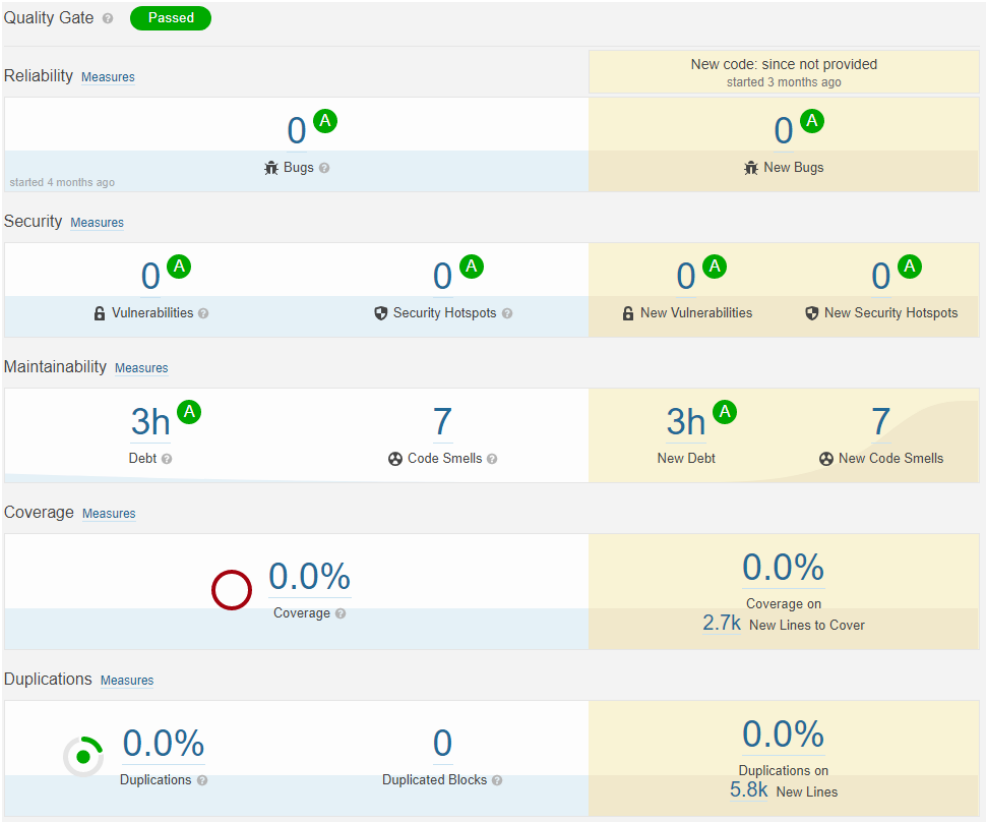
Disse møtene fungerte bra for gruppen. Alle fikk konstruktiv tilbakemelding på arbeidet sitt og lærte mye av dette. Å forklare koden for andre gjorde også at vi ble nødt til å reflektere over egen kode på en annen måte enn hvis vi bare gjorde det på egenhånd. Vi fikk også forståelse for de andre delene av systemet som vi ikke jobbet på, som igjen ga en bedre forståelse av systemet i sin helhet. Dette ville være nyttig dersom noen måtte ta over arbeidet til noen andre ved eventuell sykdom, eller hvis en oppgave trengte flere utviklere.

7.2 SonarCloud og SonarLint

Vi ville sikre at koden vår fulgte så mange kodestandarder som mulig og ikke hadde noen feil eller svakheter som ikke nødvendigvis fikk programmet til å krasje, men likevel kunne føre til at det kjørte dårligere enn det burde, eller kunne lede til større feil senere. I tillegg til IntelliJ sin innebygde *linter*, altså et verktøy som analyserer kode for å finne problemer [22], brukte vi to andre verktøy for å sikre at kodekvaliteten holdt seg høy.

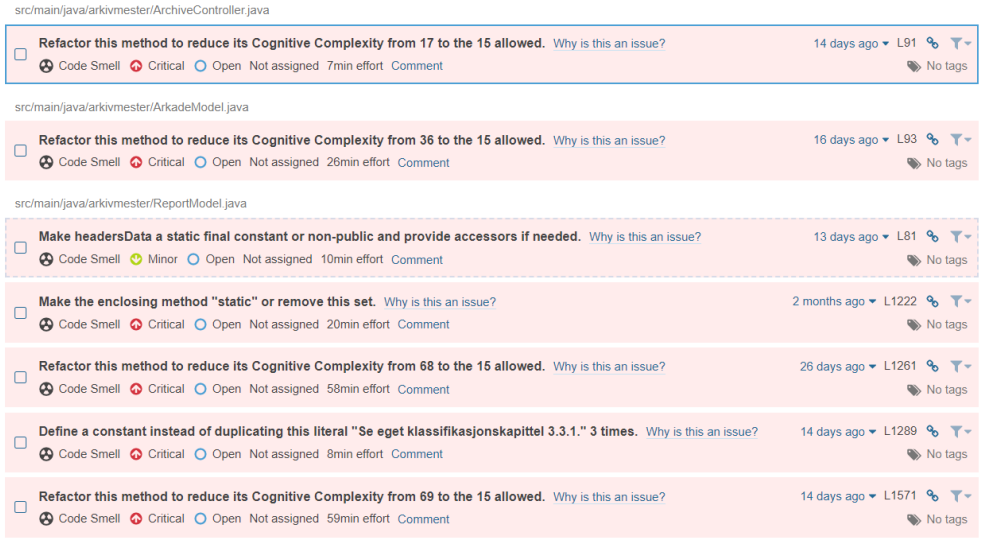
SonarCloud

SonarCloud er en *open source* plattform brukt til inspeksjon av kodekvaliteten ved å analysere koden og finne feil, sårbarheter, duplisering av kode, kompleksitet og ulike kodestandarder, for å nevne noen [23]. Plattformen er en skybasert løsning av SonarQube, med samme bruksområde [24]. Vi brukte SonarCloud for å analysere koden vi jobbet på før den ble slått sammen med *developer branch-en* og deretter brukte vi den på *developer* før denne ble *push-et* til *master*. I figur 7.1 ser man oversikten over en *branch* sin status i SonarCloud.



Figur 7.1: SonarCloud analyse oversikt

Som vist i figur 7.2 hadde ikke programmet vårt mange mangler eller problemer som SonarCloud reagerte på. De sju *code smells-ene* som blir vist er ting som vi valgte å ikke rette opp i, enten fordi vi følte det var unødvendig eller fordi analysen reagerte på kode som ble skrevet av en god grunn, og måtte forbli slik for at programmet skulle fungere. *Cognitive complexity* problemet oppsto på grunn av mange «if»-setninger i funksjoner hvor testrapporten ble generert, og disse var nødvendige fordi de bestemte hvilke tilfeller som skulle i rapporten. Selv om vi delte opp disse funksjonene i flere deler ble de fortsatt regnet som for komplekse, som vil si at koden kan være vanskelig å lese og forstå [25]. Vi følte dette var unødvendig, og at det å dele opp funksjonene i mindre biter heller ville gjøre det vanskeligere å finne fram i koden.



Figur 7.2: Resultat av SonarCloud analyse

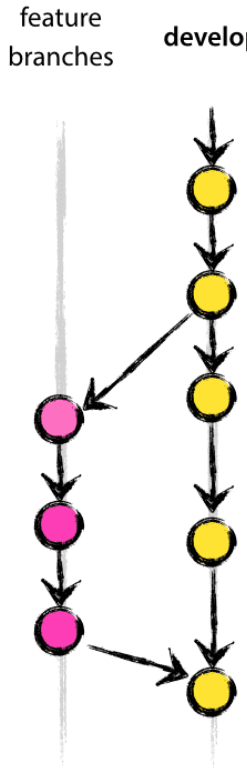
SonarLint

Samtidig med SonarCloud, som var på en ekstern side, hadde vi SonarLint som en IntelliJ utvidelse [26]. SonarLint plukker opp de fleste av de samme problemene som SonarCloud fant, men problemer som for eksempel sikkerhetssvakheter og duplisert kode kommer ikke med her. Det var likevel et utrolig nyttig verktøy som lot oss finne svakheter så fort de oppsto.

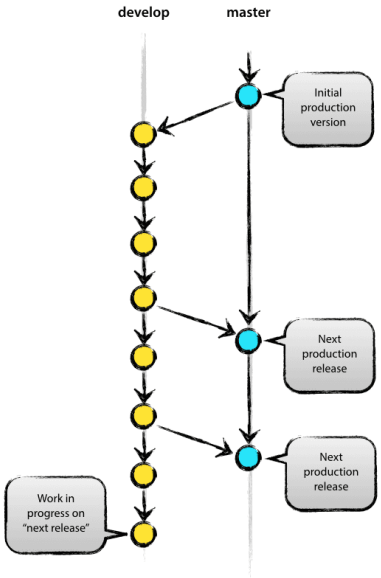
7.3 Git

For å lettere kunne jobbe på og vedlikeholde koden til prosjektet vårt brukte vi en modifisert versjon av *gitflow*. *Gitflow* er en arbeidsmetode for *git* som ble populær i 2010 [27] og som hjelper utviklere å jobbe på et stort prosjekt. Metoden passer bra for prosjekter hvor det er satt frister underveis i utviklingen hvor noe skal være ferdig, som passet bra med vår bruk av utviklingsmetodikken scrum. I denne metoden blir det brukt tre forskjellige typer *branch-er* kalt *master*, *developer*, *feature*, *release*, og *hotfix* [28]. Til vårt prosjekt følte vi det var unødvendig å ha *release* og *hotfix branch-ene*, i og med at vi var en liten gruppe og vi følte det ville bli rotete med flere enn *master*, *developer*, og *feature*, som vi brukte. *Master* og *developer* blir brukt for å holde på koden og de andre filene til hele prosjektet. Forskjellen mellom disse to er at *master* er en ferdig iterasjon av prosjektet, mens *developer* er et slags mellomledd mellom *master* og *feature branch-ene* hvor all ny funksjonalitet blir lagt inn før det går videre til *master* [28].
I vårt tilfelle ble *master*- og *developer branch-ene* slått sammen på slutten av hver sprint når all funksjonalitet som skulle eller kunne bli gjort i løpet av sprinten var

ferdig. En ny *feature branch* ble laget for all ny funksjonalitet som skulle jobbes på etter hvert som gruppemedlemmer begynte å jobbe med denne. De fikk navn etter funksjonaliteten som ble jobbet på, for eksempel *feature-runtests* hvor vi implementerte kjøringen av testene beskrevet i avsnitt 6.2.1. *Feature branch-ene* skal aldri samhandle med *master branch-en*, men blir i stedet *push-et* til *developer* når funksjonalitet er ferdig implementert og har gått gjennom kvalitetssikringsprosessen. Figurene figur 7.3 og figur 7.4 viser flyten fra *feature* til *developer*, og deretter fra *developer* til *master*.



Figur 7.3: Flyten fra *feature* til *developer* branchene [27].



Figur 7.4: Flyten fra *developer* til *master* branchene [27].

7.4 JavaDoc

JavaDoc er et verktøy som følger med JDK, og blir brukt til å generere Java dokumentasjon i HTML format. Så lenge man skriver kommentarer i dokumentasjonsformatet vist i figur 7.5 vil en oversikt over klasser og funksjoner med forklaring av disse stå i HTML filen på en oversiktlig måte. Det finnes også flere ulike tagger

JavaDoc gjenkjenner og bruker for å gi en bedre oversikt [29]. Et eksempel på disse er *@param* som forklarer en funksjonsparameter.

```
/**
 * Runs the selected custom XQueries.
 * @param prop Properties object containing the config.
 * @throws IOException Cannot create/read files.
 */
```

Figur 7.5: Eksempel på kommentar i dokumentasjonsformat

I selve JavaDoc-en kommer det en oversikt over alle klasser i programmet med beskrivelser av disse. Deretter kan man klikke seg inn på en klasse og få en oversikt over dens funksjoner, som også kan klikkes på for å få en detaljert oversikt med blant annet parametre og hva den returnerer. Et utdrag fra JavaDoc-en vår vises i figur 7.6.

runXquery

```
public void runXquery(java.util.Properties prop)
               throws java.io.IOException
```

Runs the selected custom XQueries.

Parameters:

`prop` - Properties object containing the config.

Throws:

`java.io.IOException` - Cannot create/read files.

Figur 7.6: Eksempel på hvordan dokumentasjonen til en funksjon kan se ut i JavaDoc

Formålet med å dokumentere koden på denne måten var for å hjelpe de som skal videreutvikle programmet til å få en bedre oversikt over dets klasser og funksjoner og hva funksjonene deres er. Oppdragsgiver har sagt at programmet vi lager her vil bli videreutviklet på av de som jobber på fylkesarkivet, så vi er sikre på at denne dokumentasjonen blir nyttig. I tillegg hjalp det også utviklerne på gruppen da vi skulle se hva koden noen andre hadde skrevet gjorde. Vi så sjeldent på selve JavaDoc-en, men på grunn av den oversiktlige måten å skrive kommentarer på var dette like nyttig i for eksempel de tilfellene hvor vi skulle se på en enkelt funksjon.

7.5 Refaktorering

Etter hvert som vi kom lengre ut i utviklingen av programmet, og det ble større og større, med mer og mer kode som vi måtte holde orden på, så vi at det var nød-

vendig å refaktorere deler av koden for å holde kodekvaliteten på et høyt nivå. Under disse refaktoreringene satte vi oss ned sammen og jobbet som en gruppe på dette. I og med at vi gjorde store forandringer under disse seansene tenkte vi det var best om alle jobbet på de samme kode delene slik at vi slapp for mange problemer og konflikter i forbindelse med versjonene de ulike utviklerne hadde.

De største forandringene vi gjorde var i kontrolleren, og sortering av koden til testrapport kapitlene i «ReportModel» klassen. I kontroller-klassen ble det etter hvert mer og mer kode som egentlig hørte til i modellene. En stund etter vi la merke til dette bestemte vi en dag hvor vi skulle fikse dette ved å flytte funksjoner og annen kode inn i klassene de egentlig hørte til i. Dette førte til en kontroller-klasse som var mye ryddigere og mer oversiktlig enn tidligere.

Koden til testkapitlene som holder til i «ReportModel» klassen var ofte uoversiktlig i og med at 2-3 personer jobbet på dem samtidig, og testkapitlene var ofte i feil rekkefølge. I tillegg hadde vi tidligere to ulike funksjoner som inneholdt kode for kapitlene. Til å begynne med hadde de to funksjonene ulik funksjonalitet, men etter hvert var de i utgangspunktet helt like. Vi bestemte oss for å dele opp funksjonene som håndterte koden for kapitlene i tre ulike funksjoner, og sørget for at alle kapitlene var i riktig rekkefølge og hadde kommentarer som gjorde at man lett kunne se hva slags kode som tilhørte hvilket kapittel.

Mot slutten av utviklingsperioden hadde «ReportModel» mange krevende, men essensielle funksjonaliteter for genereringen av testrapporten som var implementert, samtidig som for eksempel grafer var under implementasjon. Dette gjorde at denne modellen endte opp med å bli svært rotete, og vanskelig å håndtere. For å løse dette, så bestemte vi oss for å sette alle de individuelle funksjonalitetene som passet sammen inn i egne klasser i «ReportModel» klassen for å separere dem. Etter refaktoreringsprosessen var ferdig gikk vi fra å ha to klasser i modellen til å ha fem klasser. Disse klassene sin jobb er å dele opp rapport-teksten inn i testkapitler, tilfeller og innhold. Det var en løsning som gjorde det lett å identifisere hvilke funksjoner som passet for hvilken klasse.

Alle disse forandringene ble gjort for å ta stilling til utviklerne som skal jobbe videre med systemet. Lesbarhet og modularitet var målet for refaktoreringen, og med det skulle det bli mer oppnåelig å vite hvor i koden nytt innhold til flere kapitler kunne implementeres, og gi en bedre oversikt over de mest essensielle funksjonene.

7.6 Testing

7.6.1 Enhetstester

Vi lagde enhetstester til alle klassene, og mange av funksjonene i disse klassene. Enhetstester tester individuelle enheter og komponenter i programvaren for å validere at disse oppfører seg som de skal [30]. Det er mange fordeler med enhetstester, blant annet blir feil funnet raskere enn de ellers ville ha blitt, dersom enheten blir forandret på og testen blir kjørt. Det fører også til raskere utvikling i og med at man ikke trenger å teste hele programmet ved å kjøre det i sin helhet, legge inn data og se at alt virker, eller sette flere *breakpoints* i koden dersom IDE-en støtter dette. Selv om det kan ta en stund å lage testene vil det spare tid senere i utviklingsprosessen i og med at selve testingen tar kortere tid [30].

I vårt program var det ikke alt som kunne bli testet ved hjelp av enhetstester fordi mye av programmets funksjonalitet var avhengig av andre, eksterne programmer for å fungere. For funksjoner som håndterte eksterne filer, slik som Arkade rapporten og DOCX filer lagde vi en mappe for testressurser som testene kunne hente fra. Dette lot oss teste mye av den viktigste funksjonaliteten, og at funksjonene klarte å hente det de skulle fra eksterne filer. Vi testet også brukergrensesnittet ved å lage enhetstester som testet om de riktige knappene var aktivert/deaktivert, om siden ble oppdatert når disse knappene ble trykket på, og om alle komponentene som skulle være i brukergrensesnittet faktisk var der.

7.6.2 Intern testing

Etter hvert som vi implementerte kode for nye kapitler til testrapporten testet vi alltid mer enn ett arkivuttrekk for å forsikre oss om at koden ville virke på flere ulike typer uttrekk, etter hvert som vi fikk tilsendt disse av oppdragsgiveren. Før hver brukertest, testet vi også programmet på egenhånd og fulgte og fylte inn testskjemaet vi hadde laget slik at vi visste hva slags resultater vi kunne forvente fra brukertestene. Som følge av våre egne tester fikk vi også innsikt i om testerne fikk andre resultater enn oss, som var nyttig i og med at de ville komme til å bruke programmet på mange flere ulike uttrekk enn oss.

7.6.3 Brukertester

I planleggingsfasen av prosjektet bestemte vi at vi skulle ha to brukertester på to prototyper underveis i prosjektet. Den første brukertesten skulle bli holdt cirka halvveis i utviklingsfasen, etter den tredje sprinten, og den andre skulle bli holdt før den siste og sjette sprinten hvor all funksjonalitet skulle være implementert. Etter den siste sprinten ble vi også enige om å ha en tredje og siste brukertest hvor oppdragsgiver kunne gi en endelig tilbakemelding på programmet i sin helhet. Målet med de første to brukertestene var å gi de som skulle bruke programmet

muligheten til å gi mer grundig tilbakemelding enn i møtene vi hadde med dem, i og med at de faktisk fikk prøvd ut programmet på egenhånd og ikke bare så på våre demonstrasjoner av programmet. En annen grunn til å ha brukertestene var at testerne hadde tilgang på flere uttrekk enn det vi hadde. Disse kunne gi andre resultater enn vi hadde fått i våre tester, og føre til problemer vi ikke hadde. Det var også viktig at de testet større uttrekk som kunne ta veldig lang tid på å fullføre for å forsikre oss om at programmet ikke krasjet under testingen.

Før brukertestene lagde vi en kjørbar JAR fil og la alle XQuery spørringer (som er nødvendig for programmet) i en egen mappe som vi sendte til testerne. I tillegg fikk de et testskjema med en tabell som skulle fylles ut underveis. Denne tabellen, som finnes i vedlegg I inneholdt tester for all implementert funksjonalitet i programmet, samt hva som skjedde dersom brukeren lastet opp feil type fil, eller gjorde noe annet som skulle gi dem en feilmelding. Det var også et felt hvor brukerne kunne skrive mer generelle tilbakemeldinger som ikke var dekket av testskjemaet. Dokumentet med testskjemaet inneholdt også en installasjonsguide, en oversikt over funksjonalitet som ikke var implementert enda, hvilke kapitler i testrapporten som var implementert, og en oversikt over nødvendige eksterne programmer som trengtes for å kjøre programmet.

Første brukertest

Den første brukertesten ble sendt til oppdragsgiver i slutten av sprint tre, og testingen foregikk den første uken i sprint fire. Testerne regnet med å bruke cirka en uke på å teste programmet ferdig. Funksjonaliteten som var ferdig til testen var det meste av funksjonaliteten som gjorde at brukeren kunne teste uttrekket og lage testrapporten. Mye annen funksjonalitet som å skrive inn administrative data, eller å oppdatere filplasseringer ved hjelp av innstillinger menyen var også inne. Den største delen av programmet som var uferdig var innenfor genereringen av testrapporten, som etter sprint tre bare hadde noen få kapitler som var implementert. Derfor var fokuset av denne testen mer på hvordan programmet så ut og hvordan det føltes å bruke.

Den første brukertesten gikk ikke helt som planlagt, men vi fikk likevel masse nyttig informasjon ut av den. Vi fikk tilbakemelding tidlig i testperioden om at programmet krasjet når rapporten skulle genereres, sammen med feilmeldingen testerne fikk. I tillegg virket ikke funksjonaliteten som hentet administrativ data fra uttrekkets XML fil og plasserte denne i brukergrensesnittet. Begge disse feilene hadde samme grunnlag, at uttrekkene som ble testet var annerledes enn det vi hadde fått utdelt. I og med at vi kun hadde ett uttrekk som vi testet programmet på hadde vi ikke grunn til å tro at formatet, særlig på arkaderapporten og XML filene kunne være annerledes i ulike uttrekk. Disse ulikhetene i uttrekkene førte til at kode som fungerte på uttrekket vi hadde testet fra før av fikk programmet til

å krasje da brukerne testet sine uttrekk. Den administrative dataen var også i et annet format enn det vi hadde brukt tidligere, som førte til at dataen ikke kunne bli hentet.

Samtidig med de første resultatene fra brukertesten fikk vi også medsendt fire nye uttrekk som ble brukt under testingen. Da vi testet disse kunne vi reprodusere feilene testerne fikk, og endre koden slik at disse ble fikset. Vi ønsket at brukerne skulle gå gjennom hele den normale bruksmåten av programmet uten at det krasjet slik at vi kunne få bedre tilbakemelding på dette. Derfor bestemte vi å sende en ny versjon av programmet hvor de største feilene var fikset.
Utenom disse store problemene resulterte den første brukertesten i at vi fikk mye god og konstruktiv tilbakemelding fra oppdragsgiver. Det var andre mindre feil på noe av funksjonalitetene, som vi tok relativt kort tid å fikse etter vi ble gjort oppmerksomme på det. Ellers var tilbakemeldingen vi fikk bra, og oppdragsgiver likte det vi hadde gjort til det tidspunktet.

Andre brukertest

Brukertest nummer to ble sendt til de ansatte på fylkesarkivet i slutten av sprint fem, og foregikk i første halvdel av sprint seks, og varte i like lang tid som første brukertest. Målet til gruppen før brukertesten var å være ferdige med det meste av kodingen slik at vi bare hadde igjen små forbedringer og finpussing. Dette målet ble ikke møtt og vi manglet implementasjonen av noen av delkapitlene til testrapporten. Resten av funksjonalitetene så vi derimot på som ferdig, dersom oppdragsgiver ikke hadde forslag til forandringer eller forbedringer på disse. Det vi mest ville få ut av denne testen var tilbakemeldinger på testrapporten og ny funksjonalitet vi hadde lagt til i programmet siden sist, spesielt kjøring av egendefinerte XQuery spørringer.

Resultatene fra denne brukertesten var mye bedre enn den forrige brukertesten. Programmet krasjet aldri, og testerne møtte ikke på noen store problemer. Det vi fikk mest tilbakemelding på var kapitler i testrapporten som ikke ga forventet resultat. De fleste av disse feilene var lette å rette opp i og krevde bare at vi så over koden og kapittel filene og gjorde noen mindre endringer på disse.
En viktig endring som skjedde som følge av denne brukertesten var animasjoner, som skulle gjøre programmet mer brukervennlig. Oppdragsgiver ønsket en indikasjon på om kjøringen av testene og skrivingen av rapporten fortsatt kjørte etter en lengre kjøring. På større uttrekk tar disse handlingene lang tid, og det kunne oppstå usikkerhet på om programmet kanskje har fryst. Etter denne tilbakemeldingen la vi derfor til enkle animasjoner som spilte mens dette foregikk.

7.6.4 Regresjonstesting

Etter hver endring og utvidelse av koden hadde vi regresjonstester for å finne ut om dette skapte konflikter eller ødela for andre deler av programmet [31].

Her testet vi funksjonaliteten til programmet på samme måte som det hadde blitt gjort før for å forsikre oss om at alt fungerte som det skulle. Regresjonstestingen gjorde at vi oppdaget problemer raskt etter at en endring hadde blitt gjort. Disse problemene ble fikset så fort de ble oppdaget, ofte ved at flere på gruppen jobbet sammen, i og med at problemet kunne oppstå i koden til en annen utvikler enn den som gjorde endringen.

Kapittel 8

Diskusjon og konklusjon

8.1 Resultater

8.1.1 Resultatmål

Innlandet fylkesarkiv har fått et ferdig utviklet produkt som automatisk tester, validerer og genererer rapporten for deres digitale arkivuttrekk, og har et fungerende grafisk brukergrensesnitt. Dette var hovedmålet til de ansatte, noe som både de, og gruppen selv mener er oppfylt. Applikasjonen utfyller kravene i kapittel 2 og gir brukeren tilgang til å laste opp arkivuttrekk, konfigurere testene, og la programmet kjøre disse testene på det valgte uttrekket, samtidig som man har full oversikt over statusen av testene. Etter fullført testing er det mulighet til å automatisk generere rapporten, noe de ansatte på fylkesarkivet brukte lang tid på å gjøre manuelt, både på grunn av størrelsen på rapporten og tiden det tar å finne riktig data som skal være med i rapporten. Rapporten blir generert ut fra fylkesarkivets mal og er designet av de ansatte der. Deltest-verktøyene som brukes på arkivuttrekket er de samme som blir brukt i den manuelle prosessen.

Koden har høy kvalitet siden vi har fulgt kjente standarder, praksiser, og gjennomført kvalitetssikring på all funksjonalitet som ble *pushet* til *master branch-en*, samt implementering av enhetstester for store deler av applikasjonen. Vi har også implementert høy modularitet i prosjektet, som ønsket av fylkesarkivet. Dette er fordi standardene på digitale arkivuttrekk vil forandre seg hele tiden og hvis applikasjonen ikke kan forandre seg etter standarden, vil den etter hvert bli ubrukelig. De ansatte kan enkelt gå inn i *repository-et* og forandre på rapportmalen som er brukt, i tillegg til å forandre på XQuery spørringene for rapporten. Vi har også delt opp hvert testkapittel i rapporten til hver sin DOCX fil som inneholder de mulige tilfellene som kan oppstå basert på dataene til testresultatetene. Disse testkapitlene er lett tilgjengelige i *repository-et* også, som gjør det svært enkelt å modifisere applikasjonen for å holde den oppdatert. For å hjelpe de ansatte med vedlikeholdet er det også medfølgende produktet en brukermanual, og dokumentasjon på koden etter Google sin standard [32].

8.1.2 Effektmål

Applikasjonen vil hjelpe de ansatte på fylkesarkivet til å effektivisere prosessen ved at den gjør store deler av arbeidet de hadde automatisk. De første målingene av de ansatte etter fullførte brukertester sier at applikasjonen kan gjøre arbeidet opptil 50 prosent raskere enn om de hadde gjort det manuelt. Dette er et veldig stort tall, i forhold til at en manuell prosess som kan ta opptil flere uker å gjennomføre. Samtidig kan en ansatt sette opp flere maskiner parallelt som kjører hver sin kopi av applikasjonen, uten behov for større mannskap. Dette betyr at i stedet for at bare én ansatt kan gjøre én prosess, så kan én ansatt gjøre flere prosesser på samme tid. Applikasjonen sparer altså tid og menneskelige ressurser som da kan bli brukt andre steder.

8.1.3 Læringsmål

Alle på gruppen hadde som mål å jobbe i et profesjonelt miljø, noe som ligner så godt som mulig på en jobb vi kan ha i fremtiden. Vi hadde stor lyst til å anvende scrum metodikken i et ordentlig prosjekt fordi vi hadde lært mye om den tidligere og vi visste at store selskaper i verden brukte denne metodikken. Prosjektet var perfekt for denne metodikken siden kravene ikke var satt i stein og det ville være naturlig at det kunne komme endringer etterhvert. Etter prosjektet var gjennomført hadde vi mye erfaring med bruken av scrum, alt fra naturen av de forskjellige scrum møtene, som for eksempel *sprint planning*, til å jobbe innenfor sprinter. Det vanskeligste med scrum var å estimere hvor lang tid en oppgave trengte for å bli implementert. Dette førte til at de første sprintene hadde for mange *issues* i *backlog-en*, som igjen førte til at de måtte bli overført til den følgende sprinten. Ettersom vi fikk litt erfaring på hvordan vi jobbet sammen, og individuelt, fikk vi mer og mer riktige estimeringer, og sprintene mot slutten av utviklingsprosessen hadde nesten perfekt arbeidsmengde. En stor faktor for denne forbedringen var *sprint retrospective*, hvor vi snakket sammen om hva som gikk bra, hva som gikk dårlig, og hva vi kunne gjøre bedre i neste sprint, samt råd fra veilederen.

Programmeringsspråket Java var enda et læringsmål. Vi ønsket å bli bedre Java-utviklere og etter at noen av oss hadde fullført et semester med Java som hovedspråk, ønsket vi å gå dypere inn i språket. Vi har blitt eksponert for flere Java-biblioteker og hvordan disse er dokumentert og implementert. Spesielt bruken av Java Swing biblioteket som ble brukt for det grafiske brukergrensesnittet. Vi fikk jobbe med et komponentbygget grensesnitt som ga oss muligheten til å ikke bruke mer RAM enn nødvendig, på grunn av gjenbruket av komponenter. Komposisjonen av komponent-objektene var intuitiv og viste oss hvordan fabrikk- og sammensatt mønsterene blir brukt i praksis. Det var litt nytt for oss å behandle alt som objekter og manipulere disse gjennom referansene deres, men vi har hatt god opplæring tidligere i objektorientert programmering og ble raskt kjente med språkets paradigme. Et annet område vi gikk mer i dybden på var hvordan Java koden ble kompilert til *bytecode*, en maskin uavhengig maskinkode lagret i .class

filer, som deretter blir kjørt i en JVM [15]. Etter gjennomføringen av prosjektet føler vi oss kompetente i Java utvikling og har fått en god forståelse på de beste praksisene og bruken av språket.

Vi fikk også god erfaring i hvordan man utvikler kode med høy kvalitet ved hjelp av hverandre og andre verktøy. Vi ble fort vant til å gjennomføre kvalitetssikring av hverandres kode, en prosess som vi lærte mye av, både av oss selv og av de andre på gruppen. Vi fikk innsikt i flere alternativer og meninger om hvordan en funksjonalitet burde bli implementert og vi hadde flere diskusjoner som førte til et forbedret resultat. I tillegg brukte vi en *pipeline* for automatisk kvalitetssikring, som ga oss verdifull innsikt over om koden fulgte anbefalte praksiser og om det var mulige sikkerhetshull og programmeringsfeil. Det lærte oss om hvordan man setter opp en relevant *pipeline*, hvordan den fungerer og hvordan man kan feilsøke den hvis det oppstår feil i byggingen eller testingen i *pipeline* prosessen.

8.2 Alternative valg underveis

8.2.1 Brukergrensesnitt bibliotek

Når det kom til hvilket bibliotek vi skulle ta i bruk for det grafiske brukergrensesnittet hadde vi to valg. Det stod i mellom Java Swing og JavaFX. Begge er komponentbaserte biblioteker for skrivebordsapplikasjoner og begge har en API som utviklerne kan ta i bruk. Tidligere versjoner av JavaFX derimot, brukte *scripting* i stedet for en API for å bygge komponentene [33]. Når vi bestemte oss for hvilket som var best for prosjektet så tenkte vi først på hva slags type brukergrensesnitt som skulle bli utviklet. Siden et brukergrensesnitt ikke var et krav i starten av prosjektet, men heller et ønske fra de ansatte, visste vi at det ikke ville trenge noen store animasjoner eller et avansert utseende. Vi bestemte oss for å lage noe enkelt som oppfylte kravene, samt hadde god brukervennlighet. JavaFX er bedre for å lage grensesnitt med et mer avansert utseende, men siden vi bare trengte noe enkelt så ville Java Swing være mer enn bra nok i dette tilfellet. Ved bruk av JavaFX må man også anvende JavaFX SDK, som tidligere var integrert i JDK, men senere fjernet og gjort om til en frittstående modul [33]. Dette er noe som ingen på gruppen hadde kunnskap om, og vi måtte derfor lære Java utvikling med en annen SDK. Utviklerene hadde også tidligere kompetanse innenfor Java Swing, men ikke JavaFX. Fylkesarkivet var også mer motiverte til å bruke den standarde OpenJDK-en, som er *open source* og bruker GNU lisensen, enn andre løsninger som JavaFX SDK. Etter alle disse betraktningene ble det åpenbart at Java Swing var det beste for prosjektet. Det ville være enklere, sikre kravene på en god måte og utviklerne kunne komme fortere i gang med utviklingen av applikasjonen.

8.2.2 Rapport

En viktig beslutning vi måtte ta under implementasjonen av rapportstrukturen var hvordan vi skulle hente innholdet som skulle settes inn i rapportmalen. Vi testet ut mange innfallsvinkler under den første sprinten for å se hva som fungerte best for oss, og en av disse var å sette inn all tekst som rapporten kom til å inneholde i forskjellige XML filer, som var en del av prosjektets ressurser. Av det vi testet med XML filer, så hadde vi tre noder som skulle kunne separere innholdet: en kapittel-, en tilfelle- og en innholdsnode. Konseptet er nokså likt hvordan sluttresultatet endte opp, hvor kapittelet er på toppen etterfulgt av tilfeller, og til slutt innholdet, som beskrevet i avsnitt 6.3. Ved å gjøre rapport strukturen på denne måten kunne vi ha hatt alt av tekst på et bestemt sted. Dette virket som den best mulige løsningen for å håndtere strukturen på, men etter hvert som vi testet ut dette konseptet fant vi en del problemer som gjorde det kronglete å jobbe med. Mye av innholdet i rapporten er blant annet avsnitt eller punktlister, som vist i figur 6.2. Tekstnodene har ingen bestemt måte å inkorporere inn disse teksttypene på, og det ble dermed vanskelig å for eksempel lage flere avsnitt uten at vi måtte lage flere innholdsnoder. Det ble etter hvert alt for mange småting som hindret redigering av teksten til at det var verdt å satse på. Det gjorde at vi ble nødt til å finne andre løsninger som var bedre egnet for å håndtere DOCX filer.

Det vi senere begynte å eksperimentere med var hvordan vi kunne bruke biblioteket Apache POI til å hente og redigere DOCX filer. Siden vi allerede hadde alt av innhold til hvert kapittel tilgjengelig gjennom DOCX filer tilsendt av oppdragsgiver føltes det riktig, og mer intuitivt for videreutvikling dersom vi kunne hente all data direkte derifra. På det stadiet av utviklingen var vi fremdeles usikre på hvordan biblioteket sin funksjonalitet kunne oppnå dette så vi lette fremdeles etter andre løsninger, men i løpet av en uke så hadde vi det klart at denne løsningen skulle bli brukt og hvordan vi skulle bruke den. Oppdragsgiver var også fornøyde med denne løsningen, og vi så derfor ingen grunn til å eksperimentere med andre metoder etter dette.

8.2.3 Arkitektur

Før det ble bestemt at vi skulle ta i bruk MVC, undersøkte vi mange andre arkitekturmønstre, og målte dem opp mot kravene som var satt til prosjektet. En av de vi vurderte sterkt var lagdelingsmodellen som vi syntes delte mange likheter med hvordan prosjektet var bygget opp. Det å dele systemet opp i flere lag som håndterer sine egne typer arbeidsoppgaver, som for eksempel presentasjonslaget som sto for grensnitt delen, er noe vi følte var en ideell måte å fordele komponentene våre på. Siden all funksjonaliteten var satt i stein, så hadde det ikke vært problematisk når det kom til skalerbarhet siden de laveste lagene av modellen hadde vært de eneste lagene vi hadde måttet gjøre endringer på. Problemet med å bruke en lagdelt modell var at vi ikke kunne utnytte alle lagene til modellen. Det var grunnet at systemet ikke inneholder en database som gjorde at det nederste laget

ikke hadde en grunn til å være inkludert [34]. MVC ble da et bedre valg siden den ga en fullstendig arbeidsfordeling for alle delene sine, var fleksibel for endringer i videreutviklingen, og ga oss mulighet til å jobbe med hver vår komponent uten at det skapte konflikter med andres komponenter.

8.3 Kritikk av oppgaven

Nå som vi er ferdig med vår del av utviklingen er det viktig å se tilbake på utviklingens ufullkommenheter underveis, og visse dårligere avgjørelser vi tok som påvirket prosessen. På den måten kan vi ta lærdom av disse i senere tid.

Kapitlene som blir hentet i hver sin DOCX fil kunne vi ha satt sammen til en DOCX fil. Om vi hadde gjort det på denne måten så hadde ikke programmet hatt behov for å åpne og lukke like mange filer som antall kapitler hver gang en testrapport ble generert, nor som tar lengre tid. Det hadde også vært mer oversiktlig dersom vi hadde alt på et sted. Dette var noe vi hadde planer om å gjøre under planleggingsfasen av prosjektet, men det å bruke de DOCX filene vi allerede hadde fått tilsendt føltes mer naturlig når vi var i gang med utviklingen. Det å sette alt sammen ble glemt bort i prosessen i favør for andre problemer, og det ble av den grunn ikke gjort noe med.

Under den første brukertesten som vi hadde i slutten av sprint 3, kom det opp flere problemer fra oppdragsgivers side, som gjorde at vi fikk en del feilmeldinger og at systemet stoppet opp. Dette gjorde at vi måtte bruke en del ressurser, og tid, på å ordne opp i disse problemene. Mange av feilene var uventede i forhold til utrekkene vi selv hadde tilgjengelig. Selv om brukertestene ble håndtert underveis og vi allerede hadde planer om å ha enda en brukertest i løpet av utviklingsprossen var det synd å egentlig alltid måtte ha denne formen for usikkerhet rundt hvordan programmet vårt kom til å påvirke andre uttrekk vi ikke hadde tilgang til. Det som hadde vært en praktisk løsning for oss hadde vært om vi hadde flere uttrekk å jobbe med, selv om vi forstår formålet med å ikke få tilgang på sensitive dokumenter.

Den siste kritikken som er verdt å nevne i utviklingsprosessen er kommunikasjonsflyten mellom oss og oppdragsgiver mot slutten av utviklingen. Møtene vi hadde satt opp ved slutten av hver sprint hadde et bra tempo for framvisning av produktet, og spørsmål underveis. Problemet kom gradvis da det nærmet seg slutten av prosjektet, og vi kun hadde få kapitler igjen å implementere. De gjenværende kapitlene hadde enten mangler eller vage beskrivelser, og de var derfor utsatt til senere. Fram til slutten hadde vi spart opp spørsmål om oppgaven fram til møtene vi hadde satt opp siden vi alltid hadde mange andre *issues* på agendaen som okkuperte tiden vår, og som vi kunne holde på med fram til møtet. Dette tankesettet ga oss en uvane om å utsette spørsmål med noen få dager for å få dem avklart på møtene i stedet. Om vi hadde kommunisert mer gjennom mail, eller samtaler

på Microsoft Teams, ville vi fått en bedre forståelse av noen av kapitlene i bedre tid. Dette resulterte i at noen av kapitlene ikke ble ferdig implemenert, eller at de manglet noen av tilfellene. Kapitler med vag eller ingen beskrivelse hadde ikke oppdragsgiver vektlagt at vi skulle få fullstendig implementert inn, siden det var usikkerhet rundt de eksakte testresultatene og formuleringene kapitlene skulle ha. Det tok også tid å få disse oppdaterte beskrivelsene.

8.4 Evaluering av gruppens arbeid

Vi har samarbeidet veldig bra sammen gjennom hele prosessen og vi er stolte av arbeidet vi har gjort. Under planleggingen av prosjektet brukte vi god tid på å finne de beste teknologiene for gruppen som resulterte i effektive og gode arbeidsforhold. Det var en liten utfordring å ikke kunne møte opp fysisk grunnet koronaviruset, men vi har vært veldige smidige på hvordan vi kunne møtes digitalt i stedet. Vi brukte kommunikasjonsplattformen Discord til uformelle møter og dens chat funksjon til å stille hverandre spørsmål, eller å videreformidle informasjon til de andre gruppemedlemmene. For formelle møter med oppdragsgiver og veileder brukte vi Microsoft Teams for talemøter og til å dele filer. En fordel vi erfarte med digitale møter er at vi ikke trengte å holde de innenfor vanlige arbeidstimer, men kunne for eksempel møtes på kvelden for sen par-programmering hvis ett av gruppemedlemmene var opptatt tidligere på dagen, eller hvis det var et ønske om dette.

Vi brukte scrum, Jira og planning poker for å dele opp arbeidet, og å styre arbeidsmengden gjennom hele prosessen. Dette ga oss god oversikt over hvem som gjorde hva, hva som manglet og hva som var ferdig til enhver tid. Selv om det var litt dårlige estimeringer i begynnelsen av prosjektet så ble vi mer og mer «kalibrert» til hverandre, som resulterte i bedre estimeringer etter hvert.

Alle i gruppen var med på å kvalitetssikre hverandres kode og å gi konstruktiv kritikk på denne. Dette hjalp mye på både kodekvaliteten og kunnskapsnivået til hvert enkelt medlem av gruppen, siden vi ikke bare lærte av våres egne feil, men av hverandres feil og suksesser i tillegg. Rapporten har vi også brukt god tid på. Vi har alle vært med på å ikke bare skrive eget stoff, men å se over hverandres stoff for rettskriving, relevans og forbedringer. Dette førte til flere versjoner av rapporten, men vi er godt fornøyde med sluttresultatet.

8.5 Videre arbeid

Videre arbeid av prosjektet vil bli håndtert av de ansatte i fylkesarkivet. Applikasjonen er utviklet med dette i bakhodet og det er derfor stor tilgjengelighet for modifikasjoner av testverktøy, rapportmal, kapitteltilfeller, og XQuery spørringene som blir brukt til innhenting av data for rapporten. Høy modifisering av applika-

sjonen var også et krav i og med at Noark-standarden [4] forandres og utvikles stadig, noe som betyr at applikasjonen må kunne bli forandret samtidig med den. Det var noen funksjonaliteter med lav prioritet som ikke ble implementert som de ansatte kan implementere på egen hånd. Grunnen til at de ikke ble implementert var fordi omfanget av prosjektet var for stort, som medførte at vi måtte begrense funksjonalitet med lav prioritet, slik at vi fikk nok tid til å fullføre den automatiske genereringen av mesteparten av kapitlene i sluttrapporten. Det er fortsatt noen mindre feil i kapitlene og kapitler som mangler, men dette vil også bli fikset av de ansatte.

De ansatte ved fylkesarkivet har gitt oss tilbud om sommerjobb ved fylkesarkivet for å fortsette utviklingen av prosjektet og kanskje utvide det. Prosjektet har også fått oppmerksomhet av andre fylkesarkiv i landet som kanskje vil få tilgang til applikasjonen vår. Dette vil kreve at de lager sin egen rapportmal, noe som også vil kreve videre implementering av koden.

8.6 Konklusjon

Innlandet fylkesarkiv og bachelorgruppen har i samarbeid utviklet et produkt som vi alle er godt fornøyde med. Produktet har høy kodekvalitet og fyller kravene som har blitt satt. Med dette produktet kan de ansatte ved fylkesarkivet bli mye mer effektive i både menneskelige ressurser og bruk av tid. Applikasjonen tester og validerer digitale arkivuttrekk automatisk slik som de ansatte så for seg i innledningen og det er god mulighet for modifisering av både applikasjonens kode og ressurser for å oppholde relevans og brukelighet i forhold til den norske standarden for elektronisk dokumentasjonsforvaltning.

Avslutningsvis vil vi takke Innlandet fylkesarkiv for samarbeidet og muligheten de har gitt oss ved å få jobbe med denne oppgaven.

Bibliografi

[1] Arkivverket, *Introduksjon til arkivene*, 2021. [Internett] Tilgjengelig fra: https://www.arkivverket.no/kom-i-gang-med-arkiv/introduksjon-til-arkivene [Hentet: 10.05.2021].

[2] R. Fauskrud, *Fylkesarkivet*. Innlandet fylkesarkiv, 2020. [Internett] Tilgjengelig fra: http://www.farkiv.no [Hentet: 10.05.2021].

[3] Arkivverket, *Arkivloven*, 2020. [Internett] Tilgjengelig fra: https://www.arkivverket.no/forvaltning-og-utvikling/regelverk-og-standarder/lover-og-forskrifter-for-arkiv/arkivloven [Hentet: 10.05.2021].

[4] Arkivverket, *NOARK*, 2017. [Internett] Tilgjengelig fra: https://www.arkivverket.no/forvaltning-og-utvikling/noark-standarden [Hentet: 10.05.2021].

[5] Arkivverket, *ISO 14721 Open archival information system (OAIS)*, 2017. [Internett] Tilgjengelig fra: https://www.arkivverket.no/forvaltning-og-utvikling/regelverk-og-standarder/internasjonale-arkivstandarder/iso-standarder-for-arkivdanning/iso-14721-oais [Hentet: 10.05.2021].

[6] OWASP Foundation, *XPATH Injection*, 2021. [Internett] Tilgjengelig fra: https://owasp.org/www-community/attacks/XPATH_Injection [Hentet: 06.05.2021].

[7] B. Anda, *Use case drevet design med UML*. UIO, 2005. [Internett] Tilgjengelig fra: https://www.uio.no/studier/emner/matnat/ifi/nedlagte-emner/INF3120/h05/undervisningsmateriale/Forelesningsfoiler/20050926_OOAD2.pdf [Hentet: 04.05.2021].

[8] Wikipedia, *Agile software development*, u.å. [Internett] Tilgjengelig fra: https://en.wikipedia.org/w/index.php?title=Agile_software_development&oldid=1023834429 [Hentet: 18.05.2021].

[9] Agile Alliance, *Pair Programming*, 2015. [Internett] Tilgjengelig fra: https://www.agilealliance.org/glossary/pairing/ [Hentet: 05.05.2021].

[10] Mountain Goat Software, *Planning Poker*, 2021. [Internett] Tilgjengelig fra: https://www.mountaingoatsoftware.com/agile/planning-poker [Hentet: 11.02.2021].

[11] E. Novoseltseva, *15 Benefits Of Software Architecture*. apiumhub, 2018. [Internett] Tilgjengelig fra: https://apiumhub.com/tech-blog-barcelona/benefits-of-software-architecture/ [Hentet: 12.05.2021].

[12] V. Mallawaarachchi, *10 Common Software Architectural Patterns in a nutshell*. Towardsdatascience, 2017. [Internett] Tilgjengelig fra: https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013 [Hentet: 12.05.2021].

[13] Brainvire, *Six Benefits Of Using MVC Model For Effective Web Application Development*, u.å. [Internett] Tilgjengelig fra: https://www.brainvire.com/six-benefits-of-using-mvc-model-for-effective-web-application-development/ [Hentet: 07.05.2021].

[14] P. Pedamkar, *What is MVC Design Pattern?*. EDUCBA, 2019. [Internett] Tilgjengelig fra: https://www.educba.com/what-is-mvc-design-pattern/ [Hentet: 07.05.2021].

[15] M. Tyson, *What is the JRE? Introduction to the Java Runtime Environment*. InfoWorld, 2020. [Internett] Tilgjengelig fra: https://www.infoworld.com/article/3304858/what-is-the-jre-introduction-to-the-java-runtime-environment.html [Hentet: 05.02.2021].

[16] T. Rodgers, *Use case drevet design med UML*. Red Hat Developer, 2020. [Internett] Tilgjengelig fra: https://developers.redhat.com/blog/2020/10/08/migrating-c-and-c-applications-from-red-hat-enterprise-linux-version-7-to-version-8# [Hentet: 04.05.2021].

[17] Odalovic, *The clash of the giants - IntelliJ Ultimate vs Visual Studio Code*, 2020. [Internett] Tilgjengelig fra: https://www.odalovic.com/blog/intellij-ultimate-vs-visual-studio-code-battle [Hentet: 07.05.2021].

[18] Arkivverket, *Arkade 5 - testverktøy for arkivuttrekk*, 2017. [Internett] Tilgjengelig fra: http://docs.arkade.arkivverket.no/no/latest/ [Hentet: 19.05.2021].

[19] Javatpoint, *Java Swing Tutorial*, 2011. [Internett] Tilgjengelig fra: https://www.javatpoint.com/java-swing [Hentet: 10.05.2021].

[20] Jsoup, *jsoup: Java HTML parser that makes sense of real-world HTML soup.*, 2021. [Internett] Tilgjengelig fra: https://jsoup.org/apidocs/ [Hentet: 19.05.2021].

[21] Java2s, *Maven Tutorial - Maven Directory Structure*, 2021. [Internett] Tilgjengelig fra: http://www.java2s.com/Tutorials/Java/Maven_Tutorial/1030__Maven_Directory_Structure.htm#:~:text=Maven%20defines%20a%20standard%20directory%20structure.%20-%20src,related%20to%20the%20application%20itself%2C%20not%20test%20code [Hentet: 10.05.2021].

[22] Wikipedia, *lint (software)*, 2021. [Internett] Tilgjengelig fra: https://en.wikipedia.org/w/index.php?title=Lint_(software)&oldid=1017736638 [Hentet: 13.05.2021].

[23] Wikipedia, *SonarQube*, 2021. [Internett] Tilgjengelig fra: https://en.wikipedia.org/w/index.php?title=SonarQube&oldid=1019240322 [Hentet: 13.05.2021].

[24] Wikipedia, *SonarSource*, 2021. [Internett] Tilgjengelig fra: https://en.wikipedia.org/w/index.php?title=SonarSource&oldid=999229277 [Hentet: 13.05.2021].

[25] Code Climate, *Cognitive Complexity*, u.å. [Internett] Tilgjengelig fra: https://docs.codeclimate.com/docs/cognitive-complexity [Hentet: 13.05.2021].

[26] Jetbrains Marketplace, *SonarLint*, u.å. [Internett] Tilgjengelig fra: https://plugins.jetbrains.com/plugin/7973-sonarlint [Hentet: 13.05.2021].

[27] V. Driessen, *A successful Git branching model*, 2010. [Internett] Tilgjengelig fra: https://nvie.com/posts/a-successful-git-branching-model/ [Hentet: 12.05.2021].

[28] Bitbucket, *Gitflow Workflow*, u.å. [Internett] Tilgjengelig fra: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow [Hentet: 12.05.2021].

[29] tutorialspoint, *Java - Documentation Comments*, u.å. [Internett] Tilgjengelig fra: https://www.tutorialspoint.com/java/java_documentation.htm [Hentet: 13.05.2021].

[30] Software Testing Fundamentals, *Unit Testing*, 2020. [Internett] Tilgjengelig fra: https://softwaretestingfundamentals.com/unit-testing/ [Hentet: 13.05.2021].

[31] Software Testing Help, *What Is Regression Tesing? Definition, Tools, Method, And Example*, 2021. [Internett] Tilgjengelig fra: https://www.softwaretestinghelp.com/regression-testing-tools-and-methods/ [Hentet: 13.05.2021].

[32] Google, *Google Java Style Guide*. Github, 2021. [Internett] Tilgjengelig fra: https://google.github.io/styleguide/javaguide.html [Hentet: 10.05.2021].

[33] P. Pedamkar, *Java Swing vs Java FX | Know The 6 Most Awesome Differences*. EDUCBA, 2018. [Internett] Tilgjengelig fra: https://www.educba.com/java-swing-vs-java-fx [Hentet: 10.05.2021].

[34] G. Ziemoński, *Layered Architecture Is Good*. Dzone, 2017. [Internett] Tilgjengelig fra: https://dzone.com/articles/layered-architecture-is-good [Hentet: 17.05.2021].

Vedlegg A

Prosjektplan

21 sider.

Prosjektplan

Innlandet fylkesarkiv

Magnus Sustad
Oskar Keogh
Tobias Ellefsen
Esben Bjarnason

Våren 2021

Innhold

1 Mål og rammer

1.1 Bakgrunn

Fylkesarkivet i Innlandet (tidligere Oppland) ble etablert i 1995 og fikk ansvaret for Hedmark fylkeskommunes arkiver 01.01.2020, og senere de historiske arkivene til Innlandet fylkeskommune (1). Fylkesarkivet i Innlandet holder til på fakkelgården i Lillehammer og er ett av flere fylkesarkiv i Norge. Fylkesarkivet er pålagt å ivareta, sikre og formidle sine arkiver. Arbeidet sikrer kunnskap som kan komme til nytte for privatpersoner, organisasjoner, forskere og andre som søker historiske opplysninger. Arkivering sikrer også at det ulike materialet varer så lenge som mulig ved å unngå slitasjer og ødeleggelser (2).

På grunn av sammenslåingen av fylkeskommunene Oppland og Hedmark til Innlandet fylkeskommune, i tillegg til at digitale arkiver blir større og hyppigere har Fylkesarkivet et økende etterslep på testing av digitale arkiver. Derfor er det et sterkt ønske fra Innlandet Fylkesarkiv å automatisere testingen og valideringen av de nye arkivene så mye som mulig.

1.2 Prosjektmål

Vi har delt prosjektmålene inn i tre kategorier: resultatmål, effektmål og læringsmål. Resultatmålene skal oppnås i løpet av prosjektets livssyklus. Effektmål er hva gjennomføringen av dette prosjektet medfører for oppdragsgiver, og læringsmål er hva vi studenter ønsker å lære gjennom prosjektet.

1.2.1 Resultatmål

- Lage et ferdig program med grafisk brukergrensesnitt som kan bli tatt i bruk av Fylkesarkivet i Innlandet ved prosjektslutt uten større modifikasjoner.

1.2.2 Effektmål

- Automatisere prosessen rundt testing av digitale arkiver som fører til mindre etterslep og raskere testing av uttrekk.
- Forbedre og effektivisere testing av digitale arkiv.

1.2.3 Læringsmål

- Bruke Scrum i et ordentlig prosjekt.
- Java.
- Objektorientert programvaredesign.
- Lage et brukervennlig grensesnitt.

- Lære å bruke flere ulike verktøy.
- Testing og kvalitetskontroll
  - Code review.
  - Bruke en pipeline.
  - Enhetstester.
- Lære å bruke kunnskap fra studiet på en god måte.

1.3 Rammer

- Programmet skal fungere på Windows, Linux kompatibilitet er et ønske, men ikke et krav.
- Programmet skal være enkelt å modifisere og oppdatere.
- Programmet skal kunne installeres, oppdateres og brukes uten tilgang til internett.
- Brukergrensesnitt er ønskelig, men ikke et krav. Hvis ikke brukergrensesnitt blir laget må programmet kunne kjøres fra terminalen med menyer og hjelp funksjoner.
- Rapporten programmet lager skal være i .docx eller .odf format og følge standardsetninger og mal fra fylkesarkivet.

2 Omfang

2.1 Fagområde

Fylkesarkivet jobber med å digitalisere arkivet sitt og bruker i dag tre verktøy som hjelp for å manuelt sikre at dokumentene er gyldige og godkjente etter Noark standarden. Etter sammenslåingen av fylkeskommunene Oppland og Hedmark til Innlandet fylkeskommune ble det bestemt at fylkesarkivet skal deponere for begge fylkeskommunene. Et uttrekk kan ta alt fra en time til en måned for å bli fullført og i 2020 så var det tre ganger så mange, og det uten alle uttrekkene fra Hedmark.

Oppgaven vår bruker følgende teknologier for å oppnå dette:

- Java for en kjørbar løsning.
- Java Swing biblioteket for et grafisk brukergrensesnitt.
- JUnit test framework for unit testing.
- Java DOM Parser for XQuery kommandoer mot .xml filer.

- Arkade5 for å teste arkivuttrekk etter Noark standarden satt av Arkivverket.
- Kost-Val for å teste integriteten til arkivuttrekk.
- VeraPDF for å teste integriteten til PDF/A, finner andre feil enn Kost-Val.

Vi ønsker å bruke Java på grunn av at vi er kjent med språket fra tidligere og fordi det tilbyr en lett måte til å skape et grafisk brukergrensesnitt. Programmet fungerer da også på både Windows og Linux som var et ønske fra produktkteieren. Arkade5, Kost-Val og VeraPDF blir brukt fordi de er et krav fra produktkteieren.

2.2 Avgrensning

Vi som utviklere har bare ansvaret for å utvikle et produkt for automatisk rapportskriving og testing, altså ikke skrive om innholdet i dokumentene. Dette produktet er bare ment for intern bruk hos Innlandet Fylkesarkiv/IKA Opplandene.

2.3 Oppgavebeskrivelse

Oppgaven går ut på at vi som utviklere skal utvikle en programvareløsning for å automatisk teste digitale arkiv mot Noark standarden for Innlandet Fylkesarkiv/ IKA Opplandene. Det skal være et brukergrensesnitt som ansatte kan benytte seg av når de utfører automatisk testing, uten behov for internett tilkobling. Løsningen benytter seg av de allerede brukte verktøyene Arkade5, Kost-Val og VeraPDF, som tester standarden og integriteten. Resultatet for hvert uttrekk vil være en sammenslått rapport av flere mindre rapporter fra ulike områder, som skal bestå av alle mulige feil uttrekket har og liste de opp med lettleste feilmeldinger.

Funksjonelle krav løsningen har er følgende:

- Brukeren kan laste opp pakket uttrekksmappe.
- Brukeren kan laste opp uttrekksmappe og ferdiggjorte tester.
- Brukeren kan starte testing av uttrekk.
- Brukeren kan skrive inn XQuery kommandoer.
- Testene skal kunne modifiseres både i brukergrensesnittet og i koden.
- Programmet skal ha et grafisk brukergrensesnitt.
- Sluttrapporten kan skrives ut i .docx og .odf format.
- Oversikt over hvilke uttrekk som er klare, under testing og ferdig testet.

- Oversikt over administrative data for uttrekket.
- Programmet skal hente fra Arkade5, Kost-Val og VeraPDF.

Ikke-funksjonelle krav løsningen har er følgende:

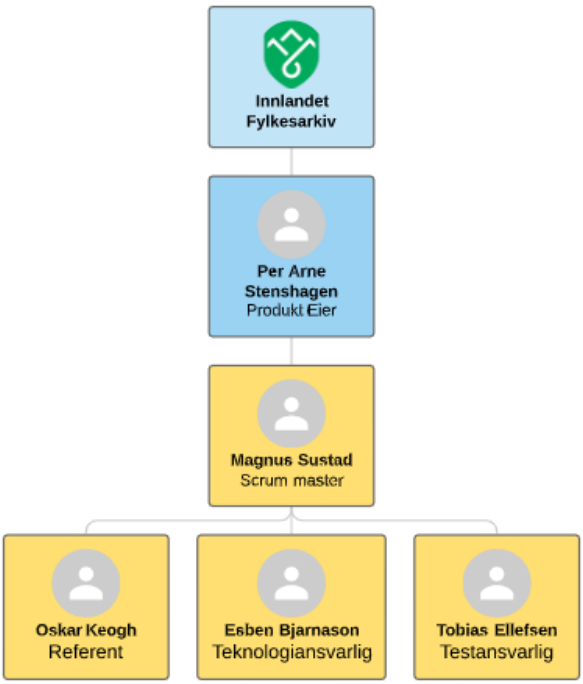
- Programmet skal ikke krasje.
- Programmet skal kunne kjøres over flere dager.
- Programmet skal installeres, oppdateres og kjøre uten internett.
- Programmet skal være kompatibel på Windows og Linux Ubuntu.
- Programkoden skal være modifiserbar for endring av tredjepart verktøy, testrapport input, rapport standarden.
- Programmet skal være brukervennlig.
- Programmet skal være godt dokumentert.

2.3.1 Arkitektur

I begynnelsen av første sprint kommer vi til å lage en modell av programvare-arkitekturen på et høyt nivå. Dette vil hjelpe oss å få en bedre forståelse av hvordan programvaren kommer til å bli satt sammen, og hvilken arkitektur den skal ha. Modellen vi lager her kan endres ganske radikalt gjennom prosjektet etter hvert som vi kommer lengre i utviklingen, men vi mener det er lurt å ha en grov modell fra starten av selv om den kommer til å endres.

3 Prosjektorganisering

3.1 Organisasjonskart



Figur 1: Organisasjonskart

3.2 Ansvarsforhold og roller

Produkteier - Per Arne Stenshagen

- Vår hovedkontaktperson ved Innlandet fylkesarkiv som vil være til stede på alle møter med oppdragsgiver under prosjektet.
- Forklarer og kommer med innspill slik at produktet vi lager oppfyller fylkesarkivets ønsker.

Scrum Master - Magnus Sustad

- Bindeledd mellom Scrum-teamet og produkteier.
- Sørger for at utviklingen holder et godt tempo og at teamet holder alle tidsfrister.

- Innkaller og leder møter med Scrum-teamet.

Referent - Oskar Keogh

- Skriver referat fra hvert møte med gruppen, veileder, og produkteier. Disse blir også lagt inn i loggboken.

Teknologiansvarlig - Esben Bjarnason

- Sørger for at alle bruker pipeline og linter i tillegg til at branches og commits blir gjort i henhold til gruppens standarder.

Testansvarlig - Tobias Ellefsen

- Har ansvar for at testene blir utført og hva som testes.

Utvliklere - Magnus Sustad, Oskar Keogh, Esben Bjarnason og Tobias Ellefsen

- Utvikler programvaren og følger alle kodestandarder gruppen har blitt enige om.

3.3 Rutiner og regler i gruppen

3.3.1 Grupperegler

1. Alle gruppemedlemmene skal lage en ukesrapport der man skriver hva som er gjort i løpet av uka.
2. Alle gruppemedlemmene skal lage en månedsrapport der man skriver hva som er gjort i løpet av måneden.
3. Alle gruppemedlemmene skal bruke loggboka for å logge tid og aktivitet.
4. Gruppen skal prøve å møte minst tre ganger i uka for å snakke om hvordan det går og veien videre. Møtene kan foregå både digitalt og fysisk, med mål å ha minst ett fysisk møte hver uke.
5. Gruppemedlemmer skal si fra i god tid på forhånd om de ikke kan komme på gruppemøter og møter med veileder/produkteier.
6. Alle gruppemedlemmene skal jobbe minst 25 timer hver uke.
7. Gruppen skal følge en felles kode- og dokumentasjonsstandard.

3.3.2 Brudd på gruppereglene

Dersom en eller flere av reglene brytes holdes et møte hvor vi bestemmer hva som skal gjøres og eventuelt avtale et møte med veileder. Gruppemedlemmer kan bli bedt om å forlate gruppen dersom regelbruddene er alvorlige nok og de andre medlemmene mener dette er riktig.

4 Planlegging, oppfølging og rapportering

4.1 Hovedinndeling av prosjektet

Prosjektet er delt inn i fire faser som vi kommer til å gå gjennom i løpet av oppgavens kretsløp: Planlegging, systemutvikling, testing av system og til slutt rapportskriving. Siden tidsperioden er relativ kort så er det å velge riktig metodikk kritisk til at utviklingen støter på så få hindringer som mulig underveis.

4.2 Valg av Metodikk

Dette er et relativt kort prosjekt med konkrete mål der vi allerede får tildelt verktøy som skal brukes og tydelige avgrensninger er satt på plass. Dette er karakteristiske trekk ved for eksempel fossefallsmetoden (3), som kunne vært en løsning. Det som også kom fram i oppgaven var de frie tøylene vi fikk på å velge ut de funksjonelle kravene, samt en stor tilgjengelighet fra produkteieren under hele utviklingen. Gjennom stadig kommunikasjon må vi tilrettelegge for produkteierens anbefalinger når vi legger opp ideer, noe som gir oss mulighet til å gå begge veier når det kommer til metodikk. Siden produkteieren vår er kontinuerlig med prosjektideene og stiller seg til rådighet gjennom prosessen så gir det oss en mulighet til å være fleksible i våre planer. Dette gjør at en smidig metodikk blir det beste alternativet i våre øyne. Gruppen har også best erfaring med smidige metodikker fra tidligere, og har god forståelse av prosessen slik at arbeidsflyten fungerer bedre.

Den smidige metodikken gruppen har valgt er Scrum, som i motsetning til Kanban har en systematisk rekkefølge av oppgaver som må prioriteres før vi kan gå over til de neste oppgavene. Det er heller ikke så mange faglige ansvarsområder vi kunne ha gitt hverandre på grunn av avgrensningene. Dette gjør at vi ikke kunne klart å etablere en god automatisk flyt på arbeidsfordelingene om vi hadde hatt tatt i bruk et Kanban board. Scrum gir oss god oversikt over arbeidet andre jobber på gjennom sprinten siden vi på forhånd vet hvilke oppgaver som må bli gjort på de spesifikke sprintene (4). Det blir også lettere å vise progresjonen av systemet til produkteier i våre sprint review meetings. I tillegg til Scrum kommer det til å benyttes metoder fra andre metodikker som for eksempel parprogrammering fra Extreme programming, som kan bli nyttig i og med vi har en gruppe bestående av folk fra ulike studieretninger med hver sine ulike erfaringer.

4.3 Plan for oppfølgingsmøter

En sprint er lagt opp til å vare 12 dager, der en sprint starter på en mandag og avsluttes på en fredag. Det er for å gi oss tid til å få fram en ferdig prototype av den funksjonaliteten som har blitt valgt for den sprinten. I enden av hver sprint skal scrum masteren sammen med resten av utviklerne ha et sprint review meeting med produkteier for å oppdatere ham i prosessen. Her får vi muligheten

til å få mer input om arbeidet, og råd om annen funksjonalitet som hadde vært greit å fått inkludert i systemet. Dette blir da lagt inn som user stories og issues i Product backloggen som vi velger ut ifra til Sprint backloggen i neste Sprint planning meeting i den kommende sprinten. Hver tirsdag har vi avtalt å ha kort møte med veilederen vår klokken 13:30, der vi tar opp spørsmål vi har til rapport og utvikling. Innad i gruppen er det planlagt å starte dagen med et digitalt daily Scrum møte før vi fortsetter med arbeidet. Vi har også tre dager der vi møtes fysisk eller digitalt og gjennomgår arbeidet, reflekterer og hjelper hverandre videre.

5 Organisering av kvalitetssikring

5.1 Dokumentasjon, standardbruk og kildekode

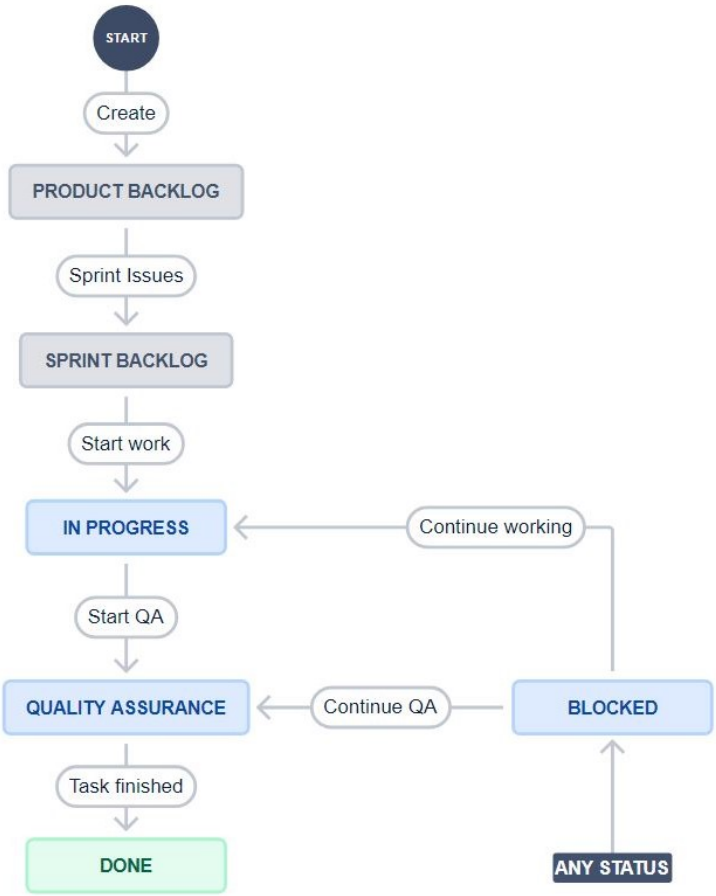
Vi har satt opp felles rutiner, commitregler, programmer/verktøy og kodestandarder som skal følges gjennom hele prosjektet. Det skal være lett for gruppen å finne ut hva som blir jobbet på, hvem som har ansvaret i tillegg til god dokumentasjon på hvordan koden fungerer. Koden blir analysert av Sonarcloud sin integrerte pipeline og deretter gjennomgått av andre utviklere før den blir committed til master branch i Bitbucket.

5.1.1 Jira

Jira er et agile prosjektstyringprogram som vi bruker som scrum board for å holde user stories og issues. På starten av hver sprint velger vi issues som vi kan fullføre i sprinten fra Product Backlog og setter dem i Sprint Backlog. Når en utvikler velger en issue å jobbe med legges den i In Progress. Utvikleren setter seg selv som ansvarlig for issuen slik at alle kan se hvem som har ansvaret og hva som blir jobbet på. Når funksjonaliteten er på plass blir issuen flyttet til Quality Assurance der det blir kjørt JUnit tester og code review på koden. Hvis issuen består alle testene og har fått en godkjenning fra resten av gruppen så flyttes den til Done. Blocked er for issues som ikke kan bli fullført enda på grunn av manglende input fra produktkteier, eller er avhengig av at andre issues blir implementert først.

Status	**Beskrivelse**
Product Backlog	Issues som skal gjøres.
Sprint Backlog	Issues som skal gjøres i nåværende sprint.
In progress	Issues som jobbes på.
Quality assurance	Issues som skal bli sett over og testet av en annen utvikler.
Blocked	Issues som krever at en eller flere andre issues blir fullført først.
Done	Ferdige issues som fyller definisjonen vår av Done.

5.1.2 Jiraprosessen



Figur 2: Arbeidsflyten i Jira

5.1.3 Sonarcloud

Sonarcloud gir oss en integrert pipeline for statisk kodeanalyse i Bitbucket repositoret vårt. Den sjekker for bugs, skrivefeil, sårbarheter og gir tilbakemelding på hvilke kodestandarder som ikke blir fulgt. Dette kan være manglende dokumentasjon på kode, struktur osv. Issue loggen viser detaljert oversikt over hvor koden kan forbedres med eksempler på riktig kodestruktur.

5.1.4 Sourcetree

Sourcetree er et grafisk brukergrensesnitt for repositories. Den gir oversikt over nye branches som blir laget og slått sammen, i tillegg til hvem som jobber på branchen. Den gir også en klar oversikt over commithistorikken for spesifikke branches. Oversikten hjelper oss å ikke bli forvirret over hva som skjer i repositoret.

5.1.5 Testing

Det blir laget automatiske tester for kritiske funksjoner og klasser. Koden skal også bli vurdert av andre utviklere før den blir implementert i developer branchen. Når vi er ferdig med den funksjonelle prototypen vil det bli kjørt brukertester av de ansatte på fylkesarkivet. Den siste sprinten blir dedikert til å teste den ferdige prototypen og det vil bli kjørt nye brukertester.

5.1.6 Kodestandard og branch struktur

Gruppen har blitt enige om å bruke Intellij sin plugin Sonarlint som forsikrer oss om at kodestandarden blir fulgt. Den er intuitiv å bruke, og den retter automatisk opp i kode som ikke følger retningslinjene satt til programmet. I tillegg har vi diskutert, og blitt enige om hvordan retningslinjene for kommentarer, funksjoner, variabler og filer kommer til å være.

Branch strukturen til prosjektrepositoret er en master branch, en developer branch og flere feature branches. Master branchen er bare for funksjonalitet som oppfyller definisjonen av Done og blir som en backup av koden. Developer branchen er også bare for funksjonalitet som oppfyller den samme definisjonen, men vil bli holdt oppdatert underveis i sprinten, når det kommer ferdiggjorte issues. Etter hver sprint vil denne branchen bli merget med master branchen og vil da være klar for neste sprint. Feature branchene er kortlevde branches som blir laget under hver sprint for å jobbe med issues fra sprint backloggen. Utviklere kan enten ha en feature branch for seg selv eller dele med andre utviklere for parprogrammering. Disse branchene kan bare bli merget med developer branchen etter funksjonaliteten oppfyller definisjonen av Done. Det betyr at feature branchene er de eneste branchene som kan ha funksjonalitet som ikke er opp til standarden vår.

5.2 Konfigurasjonsstyring

5.2.1 Arbeidsrutiner

- Issues blir definert og lagt inn i Jira.
- Planning poker brukes under sprint planning for å estimere tidsbruk på issues.
- Sprint planning meeting og sprint review annenhver uke.

- Sprint retrospective som kun holdes av gruppemedlemmene avslutter hver sprint.
- 5-10 minutters daily sprint meeting på begynnelsen av hver dag.
- Branch struktur: feature - developer - master.
- For viktige issues kan vi bruke parprogrammering hvis vi føler det er nødvendig.
- Commitmeldinger må inneholde kommentarer som gir en enkel oversikt på hva som har blitt gjort.
- Før developer branchen blir slått sammen med master branchen må den bli godkjent av minst en annen utvikler.
- Alle klasser og funksjoner skal ha kommentarer som forklarer koden og bruksområde for klassen/funksjonen.
- Kode og kommentarer skal skrives etter anbefalte retningslinjer for det språket.

5.3 Vektøy

Navn	Type	Bruksområde
Jira	Program for issue -og project tracking	Prosjektstyring og scrum board.
Discord	Kommunikasjonsplattform.	Møter og samarbeid med gruppen.
Microsoft teams	Kommunikasjonsplattform.	Møter med veileder og produkteier.
Google Slides	Presentasjonsprogram	Lage lysbilder for presentasjoner.
Google Docs	Skriveprogram.	Loggbok og kladder.
Overleaf	LaTeX editor.	Prosjektrapport og prosjektplan.
Bitbucket	Hosting-tjeneste for kildekode.	Git-repository for koden.
Toggl	Registrerer tidsbruk.	Timeregistrering og loggføring.
Planning poker	Online verktøy for å estimere tidsbruk.	Estimere tidsbruk på ulike issues i en sprint.
BaseX	XML-databasestyringssystem og XQuery-prosessor.	Kjøre XML kommandoer.
Kost-Val og VeraPDF	PDF/A validator.	Validere PDF/A.
Arkade 5	Test og pakkeverktøy for arkivuttrekk.	Teste arkivuttrekk.
IntelliJ IDEA	Java utviklingsmiljø.	Skrive og kjøre java-kode.
Sonarcloud	Ser etter feil og svakheter i koden.	Pipeline for prosjektet.
Teamgantt	Program for å lage gantt skjema.	Lage gantt skjema for prosjektet.
Sourcetree	Git GUI.	Oversikt over branches og commits.
Code With Me	Flere utviklere kan jobbe på samme kode online.	Parprogrammering.
Sonarlint	Linter plugin for IDE.	Sjekker kodekvalitet, skrivefeil og bugs.

5.4 Risikoanalyse

For risikoene er det laget et skjema over ulike problemer gruppen vil kunne støte på under utviklingen av systemet. Dette er problemstillinger som har fått vurdert risiko basert på hvor sannsynlig det er at de forekommer, og hvor høy grad det påvirker resultatet på systemet om de forekommer, dette ligger under konsekvens i tabellen. Begge kriteriene har tre kategorier for å definere hvor høyt på skalaen de ligger:

- Sannsynlighet: Usannsynlig - Sannsynlig - Svært Sannsynlig
- Konsekvens: Uproblematisk - Problematisk - Kritisk

Risiko er delt inn i 5 kategorier for å gjøre det enklere å vite hvilken alvorlighetsgrad kriteriene sammenlagt utgjør:

- Risiko: Ubetydelig - Mindre Alvorlig - Betydelig - Alvorlig - Svært Alvorlig

5.4.1 Risikovurdering

Nummer	Beskrivelse	Sannsynlighet	Konsekvens	Risiko	Tiltak
1	Prosjektet er ikke ferdig innen fristen.	Usannsynlig	Problematisk	Mindre Alvorlig	Ja
2	Sykdom over lengre tid i gruppen.	Usannsynlig	Kritisk	Betydelig	Ja
3	Gruppemedlem slutter eller må forlate gruppen.	Usannsynlig	Kritisk	Betydelig	Ja
4	Koronatiltak blir mer alvorlige.	Sannsynlig	Uproblematisk	Mindre Alvorlig	Nei
5	Verktøy endres, som fører til at vi må gjøre endringer.	Usannsynlig	Problematisk	Mindre Alvorlig	Nei
6	Konflikt i gruppen.	Usannsynlig	Problematisk	Mindre Alvorlig	Ja
7	Tap av rapport og/eller kildekode.	Usannsynlig	Kritisk	Betydelig	Ja
8	Vanskeligheter med integrering av verktøy.	Sannsynlig	Kritisk	Alvorlig	Ja
9	Endringer i kravspesifisering fra oppdragsgiver.	Svært Sannsynlig	Problematisk	Alvorlig	Ja
10	Funksjonalitet som ikke kan bli implementert.	Sannsynlig	Problematisk	Betydelig	Ja

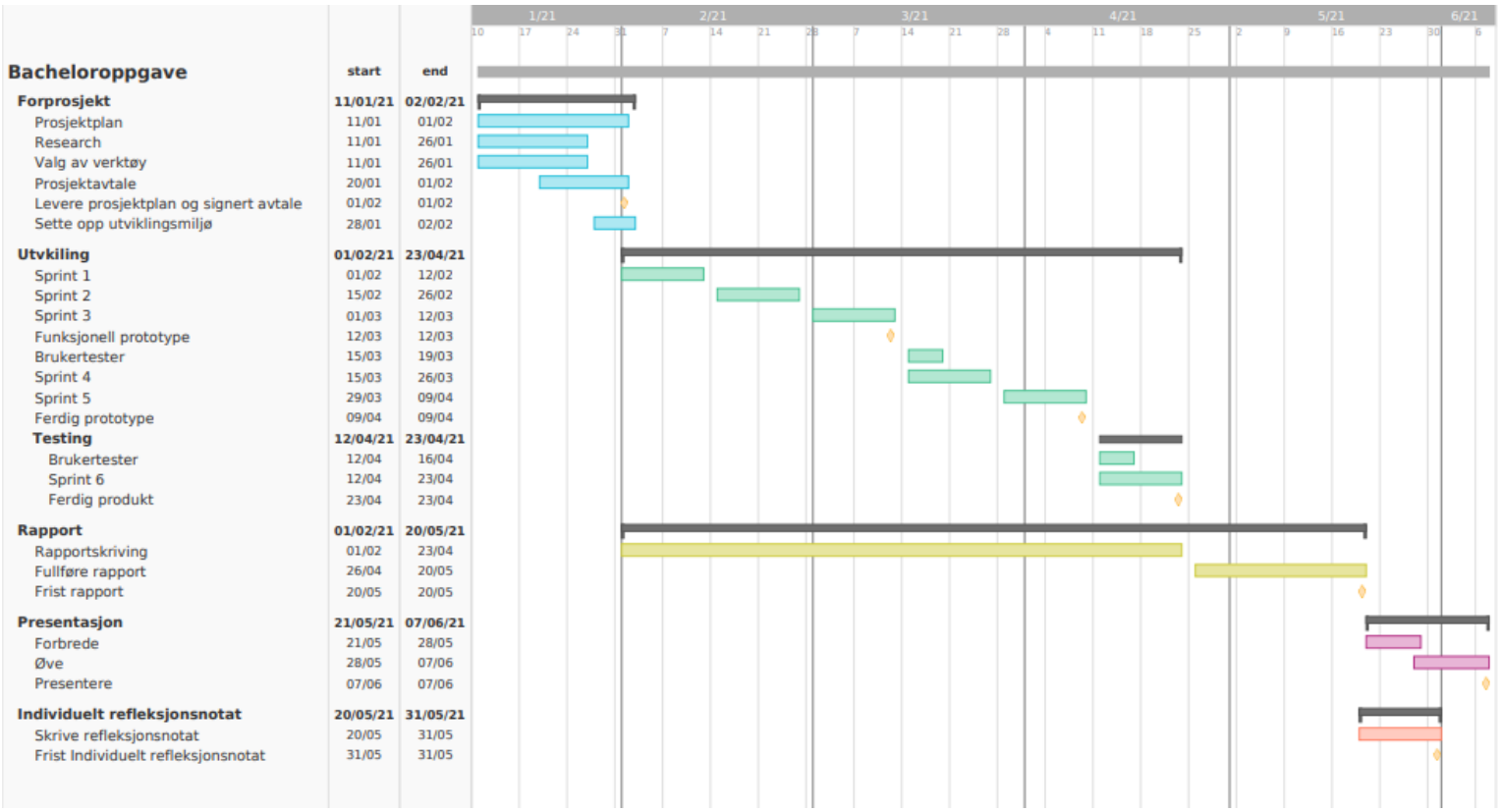
5.4.2 Tiltak

For å forhindre mest mulig av risikoene prosjektet kan møte, så er det satt i gang tiltak der vi minimaliserer risikonivå, ved å redusere enten sannsynlighet eller konsekvensskalaen.

Nummer	Tiltak
1	Hvis vi oppdager at vi ikke kommer til å bli ferdig med alt innen fristen må vi informere produkteieren. Sammen med ham kan vi bli enige om en løsning, kanskje å fjerne noe av den planlagte funksjonaliteten. For å forhindre at vi ikke blir ferdige i tide må vi etter hver sprint forsikre oss om at vi er ajour. Dersom vi ser at noe ikke har kommet så langt som det burde må vi ta tak i dette tidlig.
2,3	For å minimere konsekvensene av langvarig sykdom eller at et medlem må forlate gruppen sørger vi for at alle på gruppen vet hva de andre jobber med og har god dokumentasjon på alt slik andre gruppemedlemmer kan ta over. Dette vil minimere risikoen ved at et gruppemedlem forlater gruppen permanent, eller ved sykdom.
4	Gruppen jobber mest hjemme fra før av, det blir derfor ikke stor forskjell dersom skolen stenger eller det blir vanskelig å møtes.
6	Dersom det oppstår en konflikt i gruppen må vi løse problemet så raskt som mulig gjennom gruppemøter og eventuelt møte med veileder. For å minimere konsekvensene av en slik konflikt er det viktig at vi løser den raskt.
7	For å unngå tap av rapporten eller kildekoden lagrer vi rapporten både i Google docs og overleaf. Kildekoden ligger i et repository på Bitbucket, og lokalt hos alle utviklerne. Gruppemedlemmene må sørge for å lagre alt lokalt med jevne mellomrom, gjerne flere steder.
9	Estimere om vi har tid og kunnskap til å implementere/forandre kravspesifiseringer som produkteier kommer med gjennom utviklingsprosessen. Vi møter produkteier ofte slik det blir lettere for fylkesarkivet å formidle hva de ønsker. Det blir også lettere for oss å finne ut om det er tid til å implementere eventuelle endringer.
10	Hvis vi oppdager at funksjonalitet ikke blir mulig å implementere må vi ta kontakt med produktkteieren tidlig for å forklare grunnen(e) til dette. Sammen med dem kan vi eventuelt finne en alternativ løsning.

6 Plan for gjennomføring

6.1 Gantt-skjema



Figur 3: Ganntt-skjema

6.1.1 Gantt-skjema i tekstform

Nummer	Navn	Startdato	Sluttdato
1	**Bacheloroppgave**	11/1/21	7/6/21
1.1	**Forprosjekt**	11/1/21	2/2/21
1.1.1	Prosjektplan	11/1/21	1/2/21
1.1.2	Research	11/1/21	26/1/21
1.1.3	Valg av verktøy	11/1/21	26/1/21
1.1.4	Prosjektavtale	20/1/21	1/2/21
1.1.5	Levere prosjektplan og avtale	1/2/21	1/2/21
1.1.6	Sette opp utviklingsmiljø	1/28/21	2/2/21
1.2	**Utvikling**	1/2/21	23/4/21
1.2.1	Sprint 1	1/2/21	12/2/21
1.2.2	Sprint 2	15/2/21	26/2/21
1.2.3	Sprint 3	1/3/21	12/3/21
1.2.4	Funksjonell prototype	12/3/21	12/3/21
1.2.5	Brukertester	15/3/21	19/3/21
1.2.6	Sprint 4	15/3/21	26/3/21
1.2.7	Sprint 5	29/3/21	9/4/21
1.2.8	Ferdig prototype	9/4/21	9/4/21
1.2.9	**Testing**	12/4/21	23/4/21
1.2.10	Brukertester	12/4/21	16/4/21
1.2.11	Sprint 6	12/4/21	23/4/21
1.2.12	Ferdig produkt	23/4/21	23/4/21
1.3	**Rapport**	1/2/21	20/5/21
1.3.1	Rapportskriving	1/2/21	23/4/21
1.3.2	Fullføre rapport	26/4/21	20/5/21
1.3.3	Levere ferdig rapport	20/5/21	20/5/21
1.4	**Presentasjon**	21/5/21	7/6/21
1.4.1	Forbrede	21/5/21	28/5/21
1.4.2	Øve	28/5/21	7/6/21
1.4.3	Presentere	7/6/21	7/6/21
1.5	**Individuelt refleksjonsnotat**	20/5/21	31/5/21
1.5.1	Skrive refleksjonsnotat	20/5/21	21/5/21
1.5.2	Levere refleksjonsnotat	31/5/21	31/5/21

6.2 Aktiviteter, milepæler og beslutningspunkter

- Milepæl 1, 1/2/21 : Levere ferdig prosjektplan.
  - Grov plan for prosjektet.
- Milepæl 2, 12/3/21 : Funksjonell prototype.
  - Denne prototypen skal være ferdig etter den 3. sprinten som er ca. halvveis i utviklingsfasen. Prototypen skal ha en god del av den planlagte funksjonaliteten og skal bli testet av produkteier og kanskje andre ansatte på fylkesarkivet.

- Milepæl 3, 9/4/21 : Ferdig prototype.
  - Den ferdige prototypen skal være ferdig etter den 5. sprinten. All funksjonalitet skal være ferdig slik at vi har et produkt som kan gå videre til testing.
- Milepæl 4, 23/4/21 : Ferdig produkt.
  - Etter den 6. sprinten som går ut på å teste programvaren skal vi ha et produkt med all ønsket funksjonalitet som er testet og kvalitetsikret og klar til bruk.
- Milepæl 5, 20/5/21 : Levere ferdig rapport.
  - Rapporten skrives løpende gjennom hele prosjektet, og blir ferdiggjort ca. fire uker etter utviklingen er ferdig.

Referanseliste

1. *Fylkesarkivet* (u.å) Tilgjengelig fra: http://www.farkiv.no/ (Hentet 18.01.21)

2. *Om våre tjenester* (u.å) Tilgjenegelig fra: https://farkiv.no/tjenester/tjenestetilbud (Hentet 18.01.21)

3. Høiseth, Y. (2016) *Smidig vs. fossefall: fordeler og ulemper* Tilgjengelig fra: https://netmaking.no/Blogg/Smidig-vs.-fossefall-fordeler-og-ulemper (Hentet 18.01.21)

4. Rehkopf, M. (u.å) *Kanban vs. scrum: which agile are you?* Tilgjengelig fra: https://www.atlassian.com/agile/kanban/kanban-vs-scrum (Hentet 18.01.21)

Vedlegg B

Brukermanual

8 sider.

Brukermanual
for Fylkesarkivet

ArkivMester

v1.0

Innhold

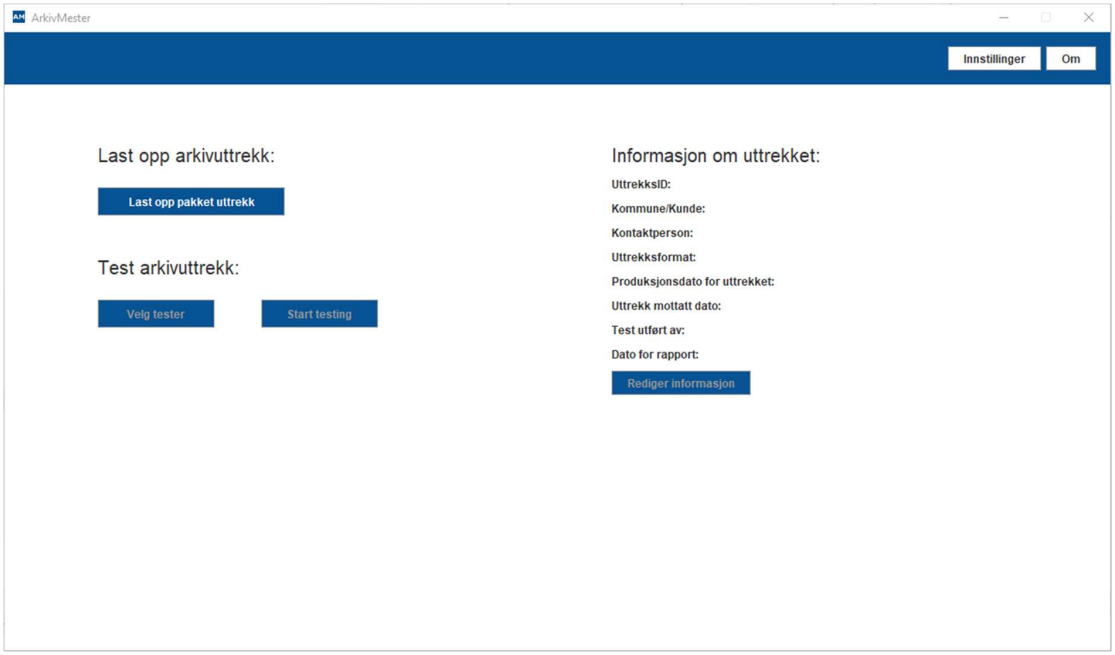
1. Nødvendige tredjeparts programmer

- OpenJDK 15
- Arkade5 CLI 2.2.1
- DROID 6.5
- KOSTVal 2.0.4
- VeraPDF 1.16.1 Greenfield
- BaseX 9.5
- 7-Zip 19.0

2. Installasjonsguide for kjøremiljøet

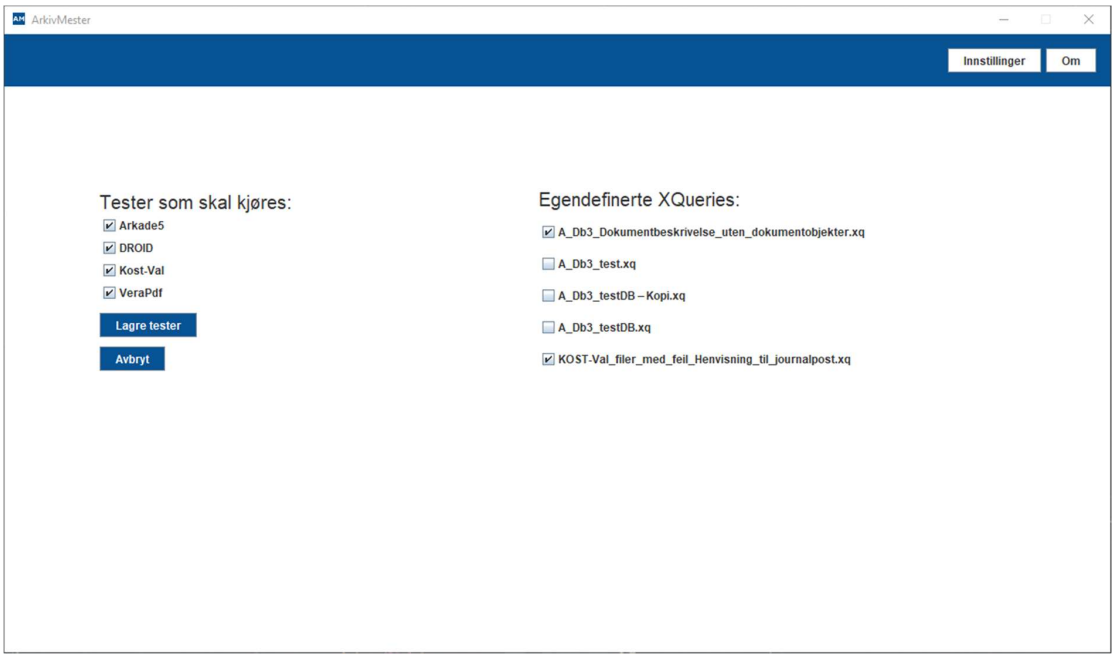
- Sørg for at nødvendige programmer er tilstede på maskinen.
- Kjør ArkivMester.jar
    - **Terminal:** Pass på at terminalen enten står i "..\openjdk-15\bin" eller at "..\openjdk-15\bin" finnes i "Path" miljøvariabelen OG at .jar filen er tilstede der du står.
        - Kommando for å kjøre via terminal: *java -jar ArkivMester.jar*
- Det vil være en konfigurasjonsfil i "C:\Brukere\user\.arkivmester" mappen som holder styr på alle fil-lokasjoner brukt av programmet. Hvis man ønsker kan man forandre de via innstillinger i applikasjonen.
- Sørg for at alle fil lokasjonene stemmer.
- Kopier .xq filene fra "XQueries" mappen til "xqueryExtFolder" mappen definert i innstillinger. Til standard er dette "E:\XQuery-Statements".
- Lag en mappe for egendefinerte Xqueries der den er definert i instillinger. Til standard er dette ""E:\XQuery-Statements/Egendefinerte". Plasser aktuelle egendefinerte XQueries her.
- Nå er ArkivMester ferdig konfigurert og klar til bruk.

3. Forside



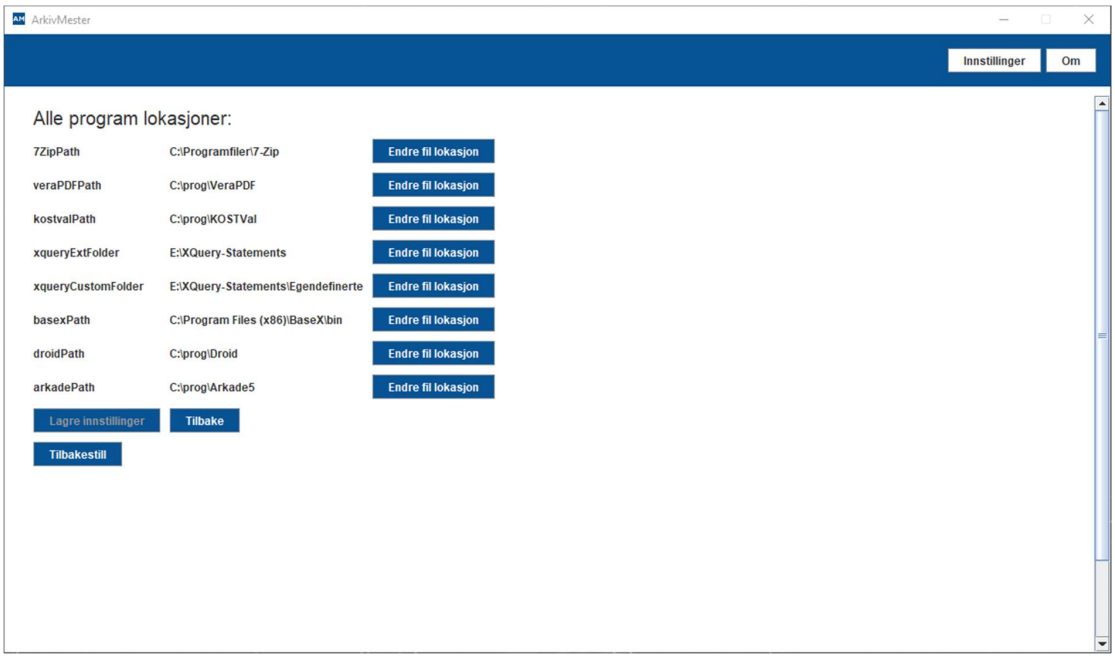
Dette er den første siden man ser når man starter applikasjonen. På venstre side ser du «Last opp arkivuttrekk» knappene. For å laste opp uttrekket ditt må du klikke på «Last opp pakket uttrekk» knappen. En filutforsker åpner seg hvor du kan navigere deg til plasseringen av uttrekket ditt. For at en katalog kan defineres som et gyldig arkivuttrekk må det være en katalog som inneholder en .tar fil, filen som inneholder selve uttrekket og en .xml fil, filen som inneholder metadata over uttrekket. Når du har funnet og valgt et gyldig uttrekk, klikk OK. Da vil «Test arkivuttrekk» knappene bli aktivert og informasjonslisten på høyre side vil automatisk hente inn informasjon fra uttrekket. For å starte testing av uttrekket, klikk på «Start testing» knappen eller du kan først velge hvilke deltester som skal være med ved å klikke «Velg tester» knappen.

4. Velg tester



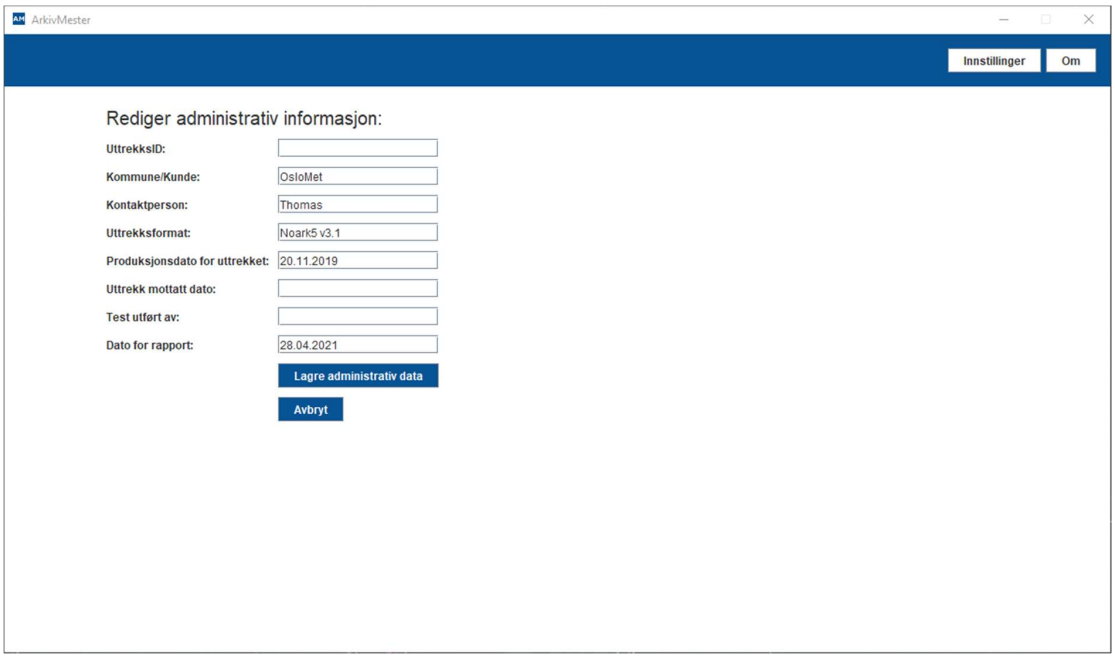
Denne siden kommer når man klikker på «Velg tester» knappen på forsiden. Her kan man velge hvilke deltester man vil inkludere. Listen for egendefinerte XQueries er de filene som ligger i din «Egendefinerte XQueries» katalog. For å spesifisere hvilke xml filer som skal spørres imot kan du spesifisere de ved å skrive *db:open("arkivmester", "dinxmlfil.xml").* Denne spesifiseringen vil gi deg rot noden i xml filen «dinxmlfil.xml». Disse XQuery spørringene vil alltid kjøres til slutt så man kan også bruke de til å kjøre spørringer på resultatene fra deltestene.

5. Innstillinger



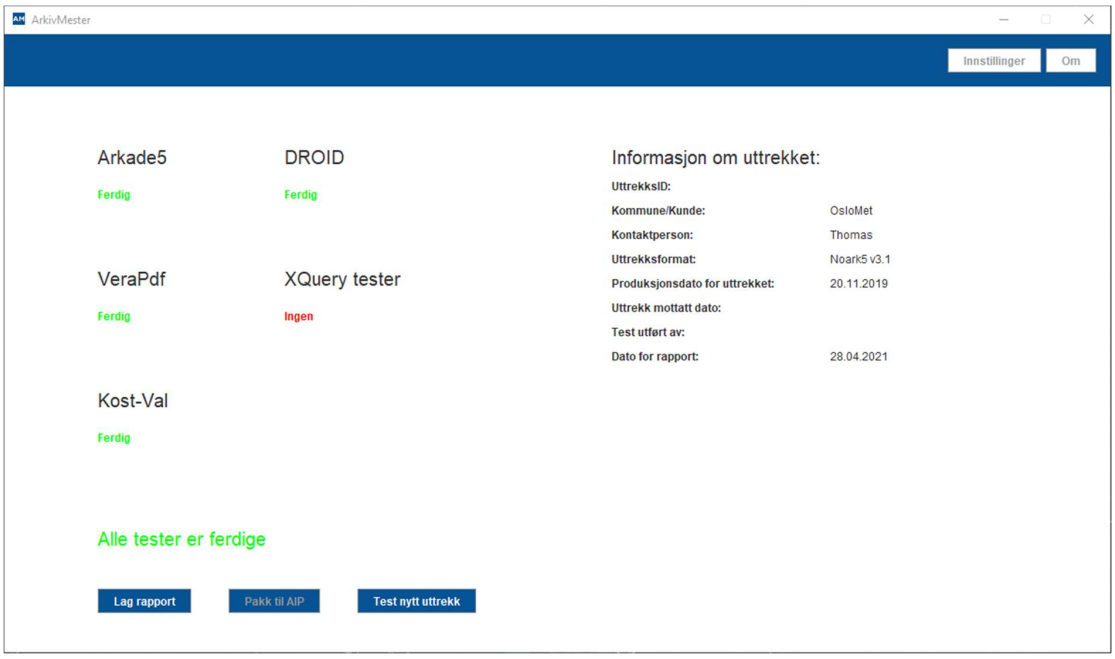
Denne siden kommer når man klikker på «Innstillinger» knappen på forsiden. Her vises det en liste over alle fillokasjonene for nødvendige tredjeparts programmer til applikasjonen. Ved siden av hvert program er det en knapp som man kan bruke til å endre standard lokasjonen. Knappen åpner en lignende filutforsker som når man laster opp uttrekk. Naviger til den der du har lagret tredjeparts programmet og klikk OK. Hvis man vil tilbakestille disse lokasjonene tilbake til standarden for fylkesarkivets Innlandet så kan man klikke på «Tilbakestille»

6. Redigere administrative informasjon



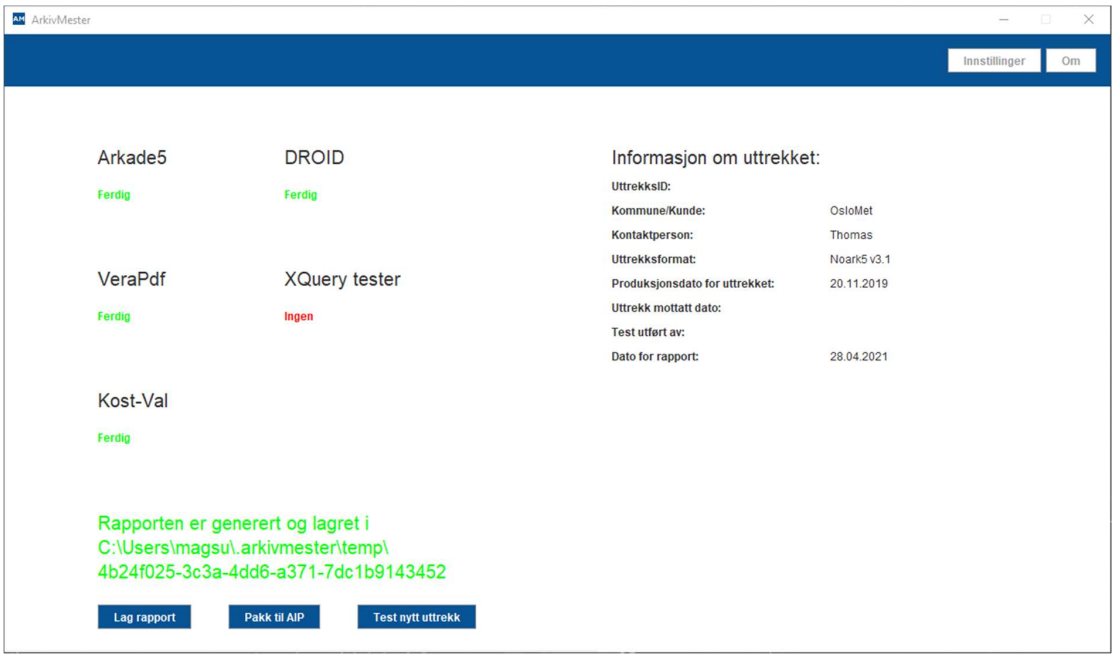
Denne siden kommer når man klikker på «Rediger informasjon» knappen på forsiden. Rediger informasjonen ved å forandre innholdet i tekstboksene, så lagre ved å klikke på «Lagre administrativ data». Det er kontroll på at datoene er skrevet i gyldig format.

7. Start testing



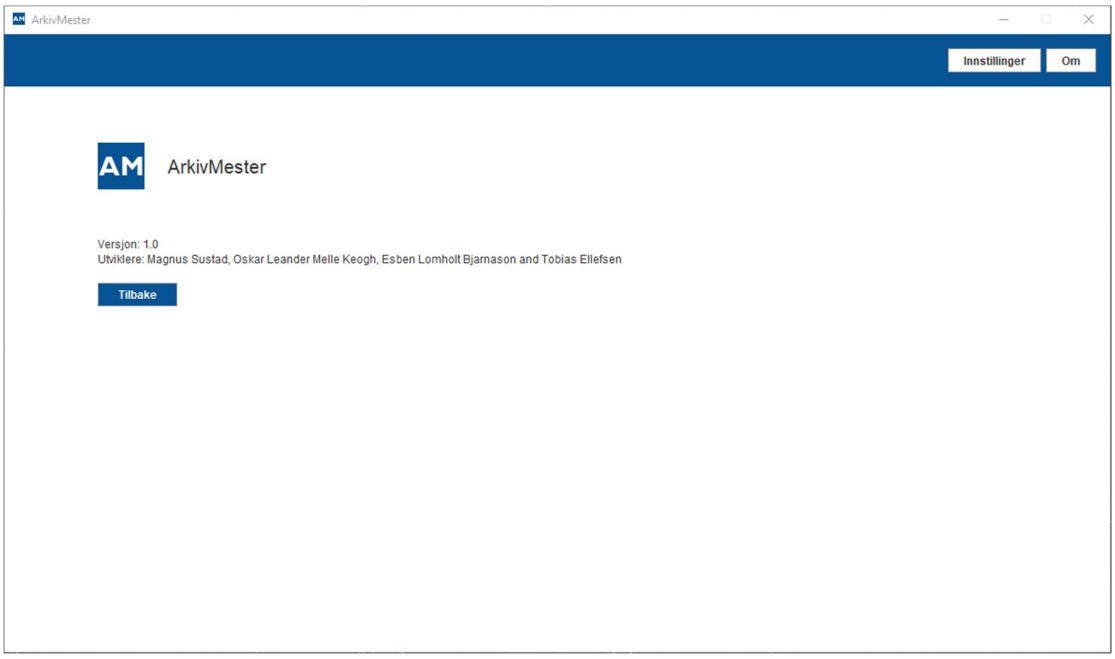
Denne siden kommer når man klikker på «Start testing» knappen på forsiden. Dette er hovedoversikten av uttrekkets status under testing og validering. Øverst til venstre er listen over deltestene med status over om den er aktivert, ferdig, kjører, eller venter på å bli kjørt. Under listen ligger hoved statusen over applikasjonen. Den har en spinner animasjon under lengre operasjoner som testing, generering av rapport og pakking til AIP for å vite om applikasjonen har fryst eller ikke. Alle tre knappene er aktivert for brukeren når alle testene er fullført og vil gi muligheten til å lage rapporten og pakke uttrekket til en AIP som inneholder denne rapporten. Man kan velge å teste et nytt uttrekk når som helst. Informasjonslisten på høyre side er den samme som er på forsiden. Grunnen til at den blir med er fordi at testene kan ta flere dager til å bli fullført og det vil da bli vanskelig å vite hvilke uttrekk som blir testet hvor og når.

8. Generering av rapport og pakking til AIP



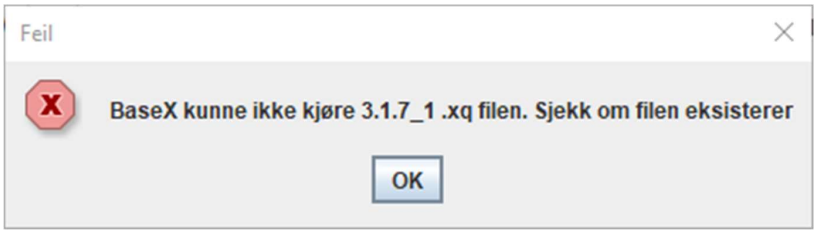
For å lage rapporten etter at testingen er fullført, klikk på «Lag rapport» knappen nederst på siden. Denne operasjonen kan også ta litt lengre tid, men hoved statusen med sin spinner vil slås på igjen for å holde oversikt. Når rapporten er lagd vil filplasseringen stå i hoved statusen og det blir mulig å pakke uttrekket til AIP med denne rapporten og resultatene fra deltestene. Plasseringen til AIPen som blir skapt vil være den samme som de andre resultatene, men vil fortsatt bli vist i hoved statusen. «Test nytt uttrekk» knappen vil være aktiv hele tiden. Den vil bringe deg tilbake til forsiden og resette all informasjon fra det tidligere uttrekket for å gjøre det klart til å teste et nytt uttrekk.

9. Om siden



Denne siden gir bare generell informasjon over applikasjonen og har ingen annen funksjonalitet.

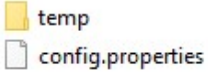
10. Feilmelding vindu



Dette vinduet blir brukt gjennom hele applikasjonen der det kan oppstå en feilmelding. I eksempelet under kan man se en feilmelding som oppstår under generering av rapporten når det mangler en XQuery spørring.

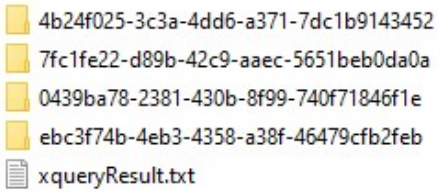
11. Brukermappe strukturen

Når man først starter applikasjonen vil en katalog bli skapt i «C:\Users\brukernavn\» katalogen, kalt «.arkivmester». Denne katalogen vil inneholde konfigurasjons filen, som inneholder



filplasseringer og gjeldende uttrekk og en «temp» katalog som inneholder alle resultater, rapporter og midlertidig plassering av ukomprimerte arkivuttrekk.

Videre inn i «temp» katalogen kan man se en tekstfil som tar imot XQuery spørring resultater og kataloger for lastet opp uttrekk.



Hvis vi for eksempel tar en titt inne i uttrekket som starter på «4b24f025» vil man se kategoriserte resultats kataloger for deltestene, selve ukomprimerte arkivuttrekket og en ferdig pakket AIP.

Vedlegg C

Prosjektavtale

3 sider.

NTNU

Norges teknisk-naturvitenskapelige universitet

Vår dato Vår referanse

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

Marit Hosar (oppdragsgiver), og

Oskar Keogh, Esben Lomholt Bjarnason, Tobias Ellefsen og Magnus Sustad (student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 11.01.2021 til 20.05.2021 .

Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon, reiser og nødvendig overnatting på steder langt fra NTNU i Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle beståtte bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv NTNU Open.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): Deepti Mishra

Oppdragsgivers kontaktperson (navn): Per Arne Stenshagen

Student(er) (signatur): Oskar Keogh dato 20.01.2021

Esben Bjarnason dato 20.01.2021

Magnus Sustad dato 20.01.2021

Tobias Ellefsen dato 20.01.2021

Oppdragsgiver (signatur): Per Arne Hosar dato 25/1-2021

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): _____ dato _____

Vedlegg D

Grupperegler

1 sider.

Rutiner og regler i gruppen

1. Alle gruppemedlemmene skal lage en ukesrapport der man skriver hva som er gjort i løpet av uka.
2. Alle gruppemedlemmene skal lage en månedsrapport der man skriver hva som er gjort i løpet av måneden.
3. Alle gruppemedlemmene skal bruke loggboka for å logge tid og aktivitet.
4. Gruppen skal prøve å møte minst 3 ganger i uka for å snakke om hvordan det går og veien videre. Møtene kan foregå både digitalt og fysisk, med mål å ha minst ett fysisk møte hver uka.
5. Gruppemedlemmer skal si fra i god tid på forhånd om de ikke kan komme på gruppemøter og møter med veileder/oppdragsgiver.
6. Alle gruppemedlemmene skal jobbe ca. 25-30 timer i uka.
7. Gruppen skal følge en felles kode- og dokumentasjonsstandard.

Brudd på gruppereglene

Dersom en eller flere av reglene brytes holdes et møte hvor vi bestemmer hva som skal gjøres og eventuelt avtaler møte med veileder. Gruppemedlemmer kan bli bedt om å forlate gruppen dersom regelbruddene er alvorlige nok og de andre medlemmene mener dette er riktig.

Alle medlemmer i gruppen er enige og vil følge disse reglene gjennom hele prosessen. Veilederen er informert om gruppereglene.

Vedlegg E

Oppgavetekst

4 sider.

(English version below)

Arbeidstittel:
Automatisk rapportskriving for testing av digitale arkiv

Oppdragsgiver
Innlandet Fylkesarkiv / IKA Opplandene

Besøksadresse /henvendelser:
Fakkelgården
Vormstuguv. 40
2624 Lillehammer

Postadresse:
Innlandet fylkeskommune
Postboks 4404 Bedriftssenteret
2325 Hamar

Kontakt:
Per-Arne Waaler Stenshagen
Rådgiver
Tlf: +47 99 27 63 01
E-post: per.arne.stenshagen@innlandetfylke.no

Kontaktperson m.m. vil være tilstede på campus Gjøvik/Teams torsdag 5. november.

Automatisk rapportskriving for testing av digitale arkiv

Bakgrunn:
Grunnet sammenslåingen av fylkeskommunene Oppland og Hedmark til Innlandet fylkeskommune ble det også bestemt at Innlandet Fylkesarkiv/ IKA Opplandene gradvis skulle ta over arkiv deponering for Hedmark fylke og Hedmark kommuner, samt fortsette med eksisterende kommuner og Oppland/ Innlandet.

Da vi også er i en fase hvor den digitale trafikken øker raskt og hvor digitale arkiver også for de eksisterende kommunene og Oppland også blir større og hyppigere, har Fylkesarkivet et økende etterslep på testing av innkommende digitale arkiver.

Det er derfor et sterkt ønske om å automatisere testingen av de nyere arkivene så mye som mulig. Denne automatiseringen vil fungere som en sammenslåing av flere testrapporter skrevet av interne verktøy.

Verktøy og språk:
Rapportene er skrevet av programmene:

- Arkade 5 (Tester selve arkiv uttrekket etter Noark standarden satt av Arkivverket)
- Kost-Val (Tester integriteten til en rekke filer, bl.a. .pdf/a, .jpeg og .tif)
- VeraPDF (Tester integriteten til pdf/a, finner andre problemer enn Kost-Val)
- BaseX (Kjører XPath og XQuery mot .xml filer)

Alle programmene har minst mulighet til å skrive i .xml format. Andre formater avhenger av programmet.

Forklaring på hva disse rapportene sier og hva som skal hentes ut blir gitt.

Automatiseringsverktøyet kan skrives i hvilket som helst språk forutsatt at det fungerer offline, da verktøyet vil bli brukt i sikker sone uten nett tilgang. C#, C++ eller Java er derimot foretrukket.

Brukergrensesnitt er ikke et krav, men hvis dette ikke blir gjort tilgjengelig kreves det at det kan kjøres enkelt i terminalen med menyer og hjelp funksjoner.

Windows kompatibilitet er minimumskrav, Linux (Ubuntu) kompatibilitet er ønskelig.

Verktøyet skal skrive rapporten i enten .docx eller .odf (Open Document Format). Standard setninger og mal rapporten skal følge vil bli gitt.

Verktøyet må også være enkelt å modifisere, slik at den kan holde tritt med oppdateringene de andre programmene får samt å kunne legge til flere standardiserte setninger basert på variabler fra de samme.

Work title:
Automatic report writing for testing of digital archives

Client:
Innlandet County Archives / ICA Opplandene

Visitor's address / inquiries
Fakkelgården
Vormstuguv. 40
2624 Lillehammer

Postal address:
Innlandet fylkeskommune
Postboks 4404 Bedriftssenteret
2325 Hamar

Contact:
Per-Arne Waaler Stenshagen
Advisor
Phone: +47 99 27 63 01
E-mail: per.arne.stenshagen@innlandetfylke.no

The contact and others will be available at campus Gjøvik at Thursday 5. November.

Automatic report writing for testing of digital archives

Background:
Alongside the merger of Oppland and Hedmark counties to Innlandet county earlier this year, the decision to gradually store current and future archives in most municipalities in the new county as well as the archives for the county itself has created an increasing backlog in the testing and depot of digital archives. This har been expatiated by various digitalization processes across the county.

Innlandet County Archives therefore require increased automatization in the form of an automatic and expandable report writing tool that can merge various test reports from internal tools.

Tools and language:
The reports are written by the following programs:

- Arkade 5 (Tests the archive according to the Noark5 standard set by The National Archives of Norway)
- Kost-Val (Tests the integrity of a variety of files including .pdf/a, .jpeg and .tif)
- VeraPDF (Tests the integrity of .pdf/a files and finds other faults than Kost-Val)
- BaseX (Runs XPath and XQuery on .xml files)

All programs have at least the ability to write their reports in .xml formats. Other formats are available.

An explanation for what each report contains and how to read them will be supplied.

The automatization tool can be written in any language on the condition it works offline, as it will be used on a secure server without Internet access. C#, C++ or Java are nonetheless preferred.

A GUI is not required, but any use through the terminal must be comprehensible with help functions attached.

Windows compatibility is required, Linux (Ubuntu) compatibility is a bonus.

The report should be written in either .docx or .odf (Open Document Format). Template and standard sentences will be provided.

Additionally, the tool must be easy to modify for future internal updates, both in terms of potential updates to the other test tools and to add more sentences based on conditional variables from these.

Vedlegg F

Loggbok

5 sider.

Møte med veileder - 02.02.21

Deltakere:
Deepti Mishra (Veileder),
Alle gruppemedlemmer.

Deepti: Framgang siden sist?
Vi hadde et sprint planning meeting i går. Deler skjerm og viser jira board.
Målet er å få et overblikk over prosjektet, så vi begynner med ulike diagrammer og arkitektur osv.

Deepti: Har vi begynt å bestemme en arkitektur enda?
Nei ikke enda, skal begynne med det i løpet av de neste dagene.
Deepti: Når vi lager et diagram får vi mer innsikt i de andre diagrammene.
Klassediagram kan lages senere, men arkitekturen er viktig.

Deepti: Har vi hatt møte med produkteier siden sist? Fått noe mer informasjon om oppgaven?
Ja vi har fått testdata og flere maler og forklaringer på testene og testrapporten. Har ikke gått veldig inn på dataen enda.

Deepti: Så hva er målet med oppgaven i egne ord?
Å automatisere prosessen rundt testing og rapportskriving. Kan ta lang tid for dem nå. Vi bruker ulike programmer for å teste dataen som vi henter og bruker i programmet vårt.

Deepti: Vi har valgt all teknologi?
Ja vi bruker testverktøyene fra utvikler.

Deepti: Har vi estimert tidsbruk på backlogen for denne sprinten?
Ja vi hadde planning poker i går på sprint backlog hvor vi estimerte tidsbruken på hver oppgave.
Deepti:Hvordan brukte vi planning poker til estimering?
Vi valgte først kort uten mye diskusjon, deretter diskuterte vi og valgte kort igjen.

Deepti: Hva er planen for sprinten?
Få et bedre overblikk av oppgaven ved å lage diagrammer og velge arkitektur, og å gjøre litt grunnleggende kode/UI.

Deepti: Har vi noen issues som går på å forstå dataen.

Vi har en som går ut på å forstå arkadetesten.

Deepti: Vi burde holde sprint retrospective før møte med PO, slik at vi kan klargjøre ting eller forklare. Bedre å ha det før review.

Deepti: Burde vi bytte møtetid til fredag?
Får se det an neste gang og bestemme da.

Daglig sprint - 04.02.21

Hva har blitt gjort siden sist?

Tobias : Sett gjennom use casene.
Oskar: Sett over test kapitlene vi har fått tilsendt fra PO.
Magnus: Funnet ut hvordan bachelor-rapporten skal se ut i overleaf.

Hva skal gjøres i dag?

Tobias: Fortsette på sekvensdiagrammet.
Oskar: Se over use caser. Gjøre ferdig UI wireframe.
Esben: Se over use caser.
Magnus: Forbedre use caser. Begynne på basic kode-skjellet av UI.
Alle: Begynne å se på domenemodell.

Sprint planning for sprint 3 - 01.03.21

Mål med sprinten: Gjøre fremgang med testrapporten og gjøre klar for brukertester.

Vi velger oppgavene som skal gjøres i løpet av sprinten. Mange oppgaver går på å gjøre klar for brukertesting, blant annet å lage et testskjema brukerne skal fylle ut og å skrive en readme med brukermanual som kan brukes under testingen. I tillegg begynner vi for fullt med å generere testrapporten.

Vi holder deretter en planning poker på oppgavene for å estimere tidsbruk.

Sprint retrospective for sprint 3 - 12.03.21

Det som gikk bra:

- Fått produksjonsklar kode.
- Godt samarbeid (Parprogrammering)
- Kompatibel med alle windows maskiner.
- Kommet i gang og fått bedre forståelse av hvordan testrapporten skal gjøres.
- Funksjoner som er lette å kalle på og bruke for kjøring av baseX og henting av data fra html. Vil føre til bedre flyt i fremtiden.

Det som gikk dårlig:

- Litt vanskelig å forstå kapitlene. Tok tid.
- Noen issues var dårlige å ga ikke et godt bilde av det vi skulle gjøre.
- Splitte opp utskrift issues bedre.

Til neste sprint:

- Bedre issues. Lage en oversikt over kapitler og hva slags input/output de har, om de er avhengige av andre kapitler, om de skal ha vedlegg, osv slik at det blir lettere for oss å vite hva som skal til for å implementere disse.

Sprint review møte - 26.03.21

Deltakere:
Alle på gruppen.
Per Arne Stenshagen (PO)
Pål Mjørlund.

Vi viser demo av det vi har gjort til de ansatte ved Fylkesarkivet.

De synes XQuery-implementasjonen ser bra ut.
Ønske om mer spesifikke error meldinger.

Vi viser frem rapporten og kapitler som er implementert, deretter forklarer vi koden og fremgangsmåten for hvordan vi legger til kapitler i rapporten slik at de får et inntrykk av hvordan dette gjøres.

Feedback på brukertesting som ikke ble med i testskjemaet.

Bra med funksjonalitet til å håndtere flere uttrekk.
Fremover: Se mer på tolkning av word dokumenter.
3.1.8: Teksten ‘Avsluttet’ kom 2 ganger, det skal den ikke gjøre.
QOL Ønske: Enkel animasjon for testingen som gir en bekreftelse på at .

Spørsmål fra gruppen

Kapitel 3.1.23

- Er det flere paragrafer som kan dukke opp?
    - Nesten uendelig. “Untatt offentlighet” kan håndteres som de andre. Vi kan ramse opp de som finnes.
    - Antalltotalvarighet
        - Opplisting av antall skjerminger med varighet.

Kapitel 3.1.26

- Hva er lite og stort antall konverteringer?
    - Kunne sett hvor stor andel av dokumentene som er konvertert.

Kapitel 3.3.1

- Hva vil ‘nivåene’ si i dette kapitelet?
    - Hvis det er en klasse under en klasse - nivå 2, en mer -nivå 3.

Kapitel 3.3.2

- Skal det være noen rekkefølge på dataen i tabellen?
    - Sortere alfabetisk eller på antall. Begge er greit.

- Møtesakstype og møte er like i våre resultater er dette riktig?
    - Kan være registrert likt. Uttrekket vi testet som er uvanlig.

- Tilfelle 2 har ingen deltagere i hele uttrekket er dette riktig?
    - Ja, det kan være et uttrekk uten regirstrete deltagere.

Kapitel 3.1.27.

- Hva vil ‘Antallspesialarkivdeler’ si?
    - Alle arkivdeler med k-kode i felles- og fagklassene.

Kapitel 3.1.14

- Må man sjekke for alle arkivdeler?
    - Ja, skal sjekke om mappen er innenfor start og sluttdato for sin arkivdel.

Generell tilbakemelding på testrapporten:

Sorter etter det som er logisk i tabellene. Hvis det er flere forskjellige navn sorter etter navn, ellers antall og noen ganger dato hvis det passer.
Varsel om større feil, og melding om ingen avvik som vi la på på egenhånd ser bra ut.
Første uttrekket vi har mest feil på namespace, de andre skal ikke ha det.

Vedlegg G

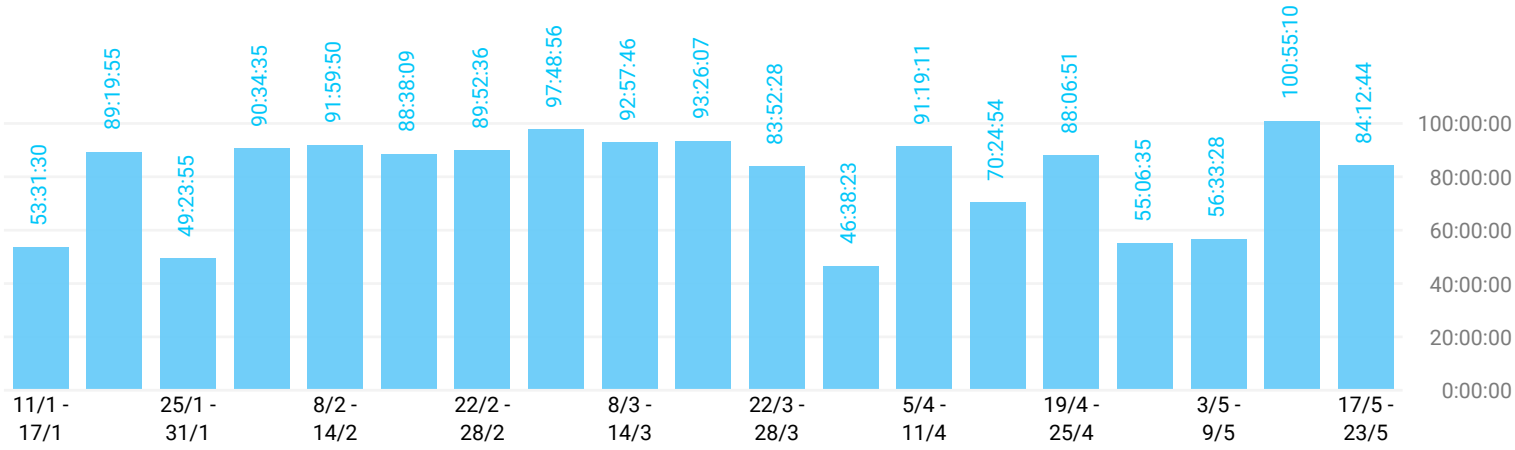
Timelogg

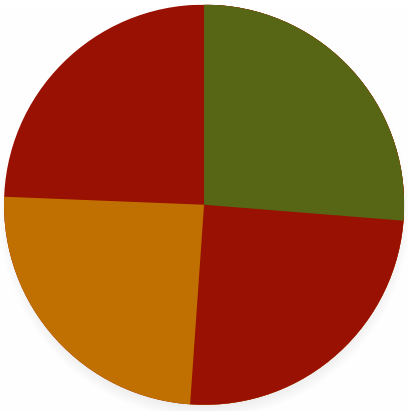
3 sider.

Summary Report

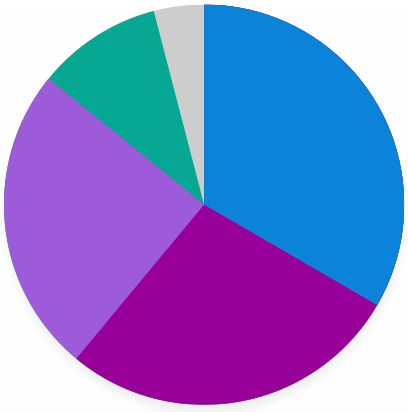
11-01-2021 – 19-05-2021

TOTAL HOURS: 1514:43:03





USER	DURATION
OK Oskar Keogh	397:19:06
EB Esben Bjarnason	376:20:14
TE Tobias Ellefsen	373:06:55
MS Magnus Sustad	367:56:48



TIME ENTRY	DURATION
Koding	506:01:19
Rapport	417:45:20
Møte	376:12:41
Research	153:38:37
Other time entries	61:05:06

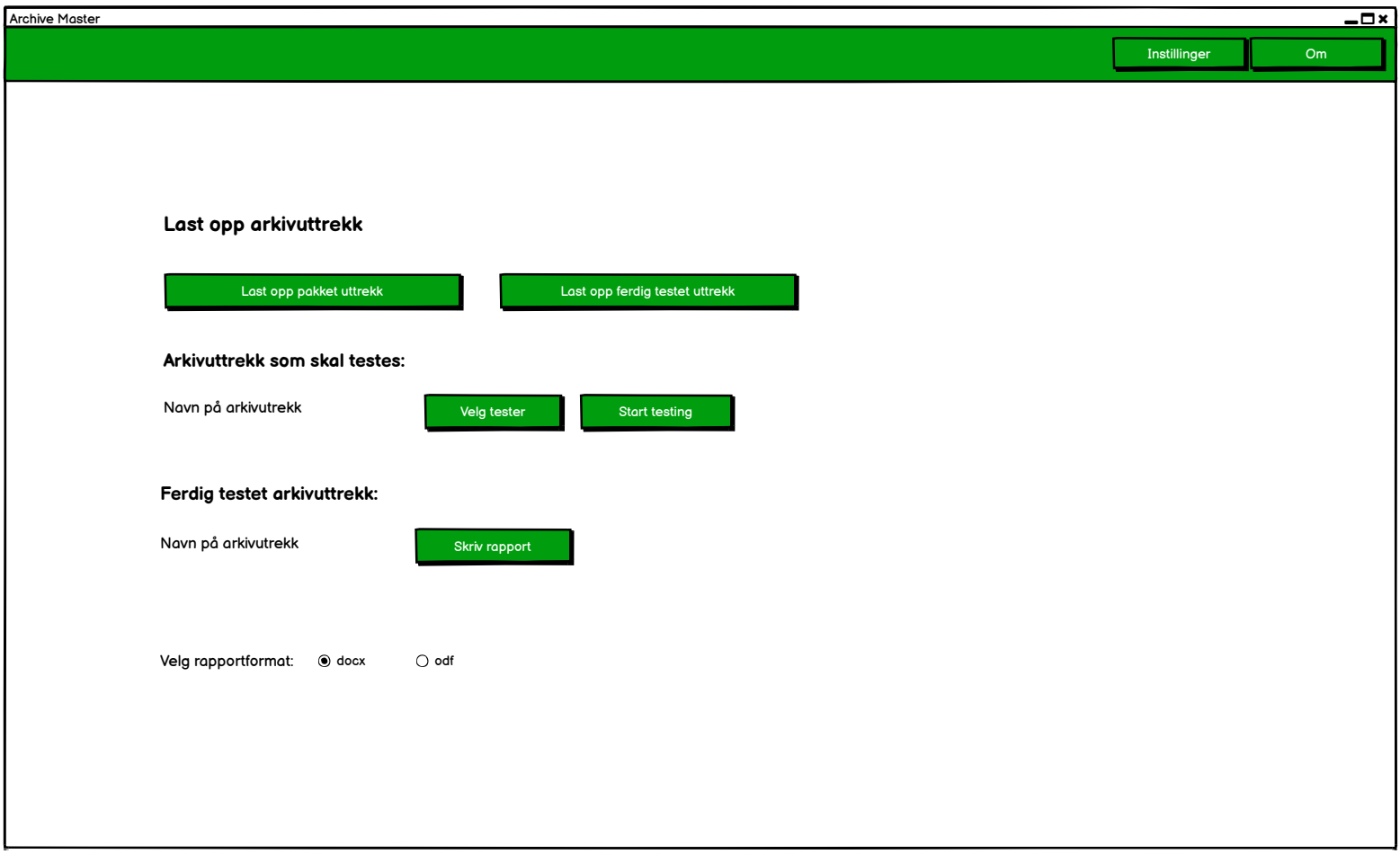
USER - TIME ENTRY	DURATION
EB Esben Bjarnason	376:20:14
Koding	148:00:34
Møte	85:55:14
Professional Programming	12:45:00
Rapport	105:54:58
Research	23:44:28
MS Magnus Sustad	367:56:48
Koding	117:41:32
Møte	90:47:07
Professional Programming	17:21:12
Rapport	102:08:54
Research	39:58:03
OK Oskar Keogh	397:19:06
Design	3:27:03
Koding	124:07:14
Møte	96:13:36
Professional Programming	16:14:00

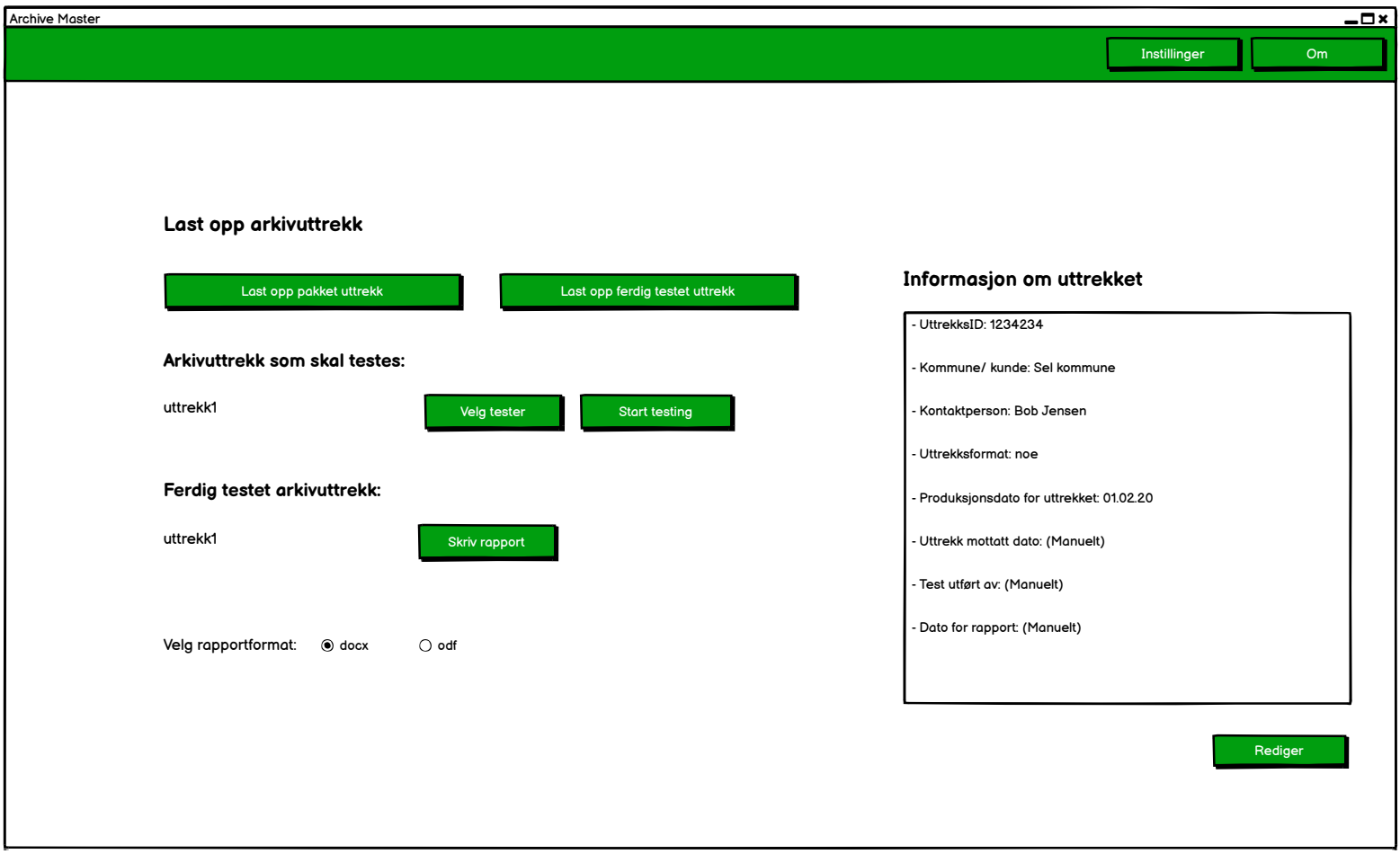
USER - TIME ENTRY	DURATION
Rapport	119:50:50
Research	37:26:23
TE Tobias Ellefsen	373:06:55
Koding	116:11:59
Møte	103:16:44
Professional Programming	11:17:51
Rapport	89:50:38
Research	52:29:43

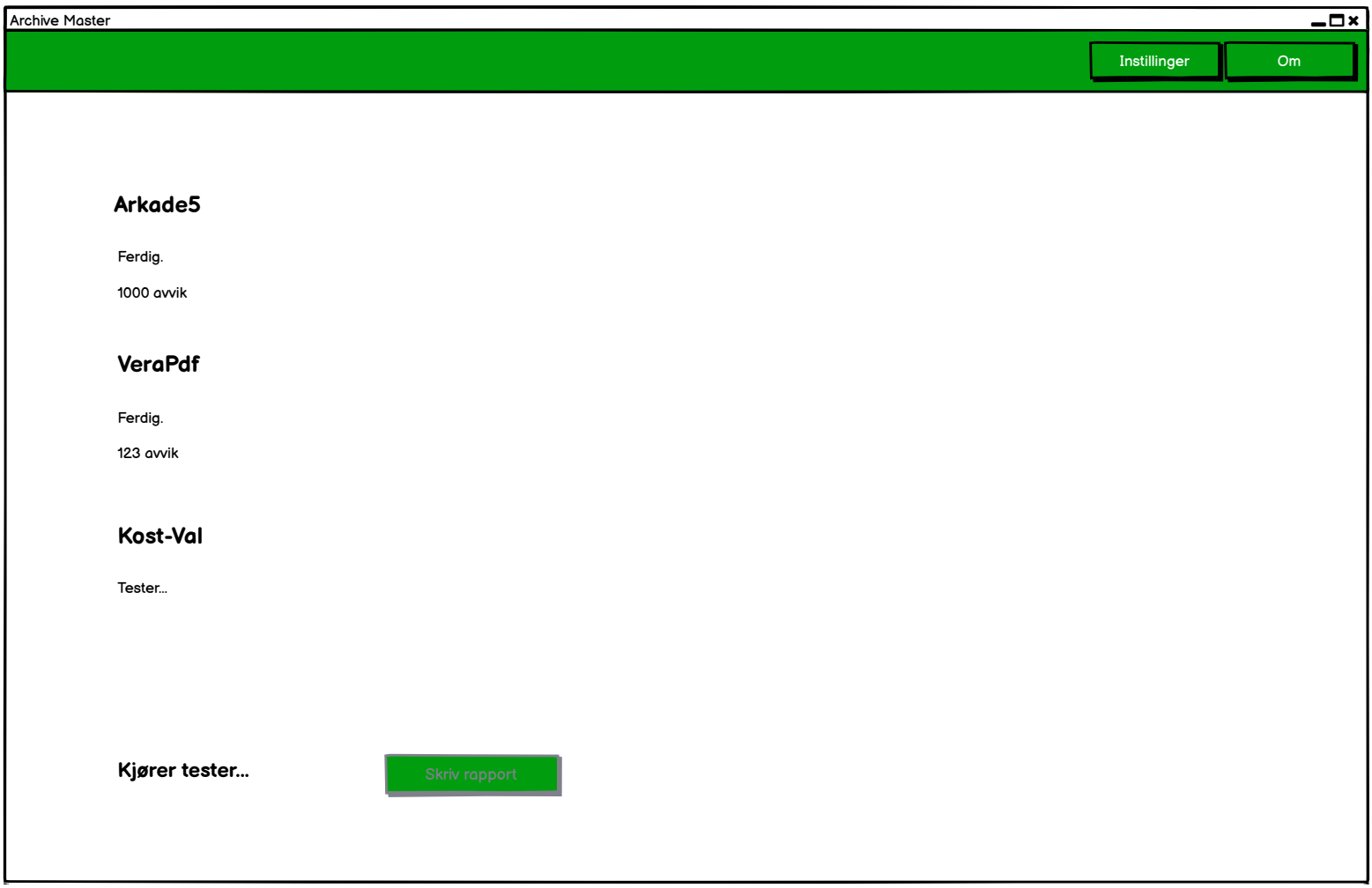
Vedlegg H

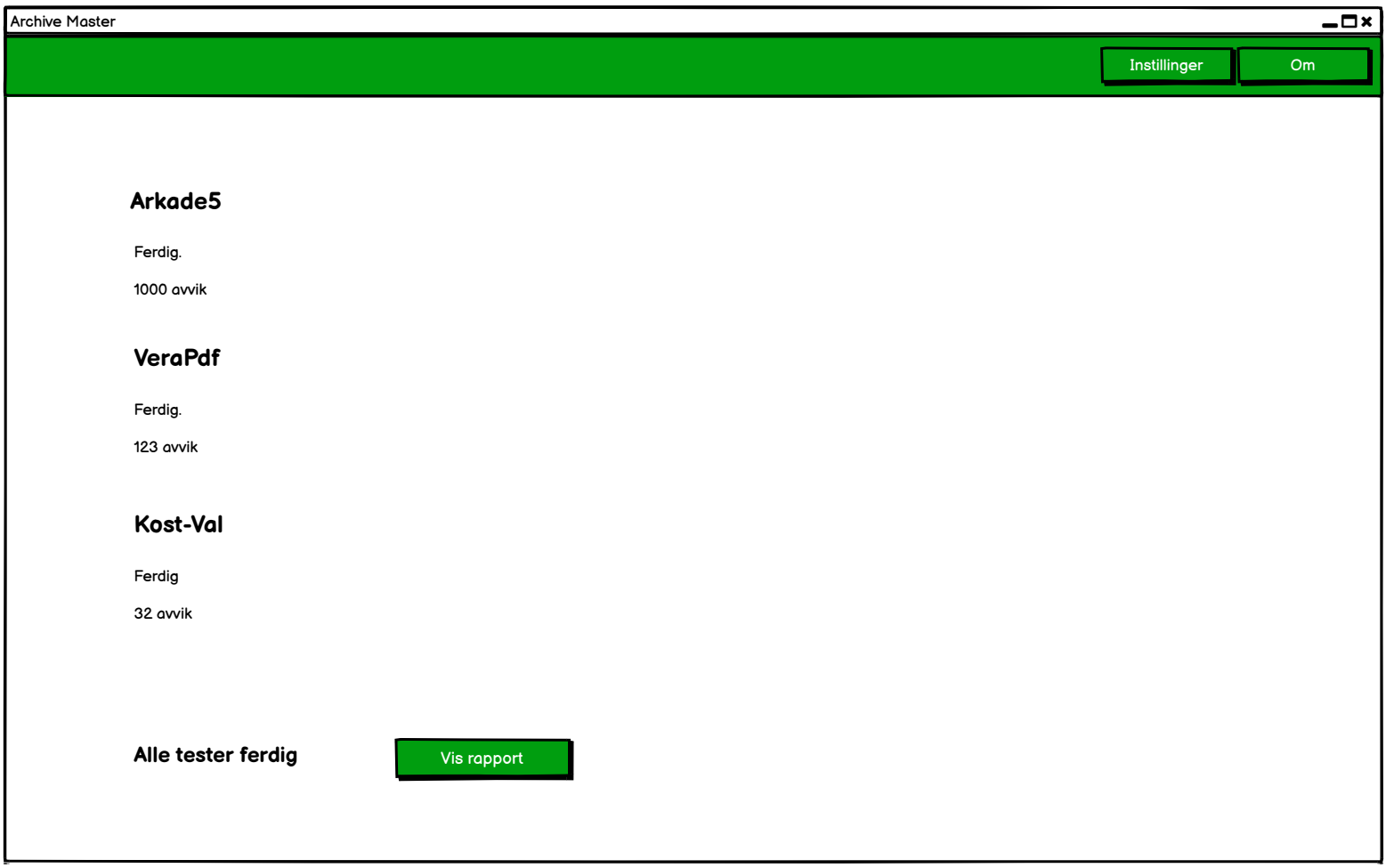
Designskisser

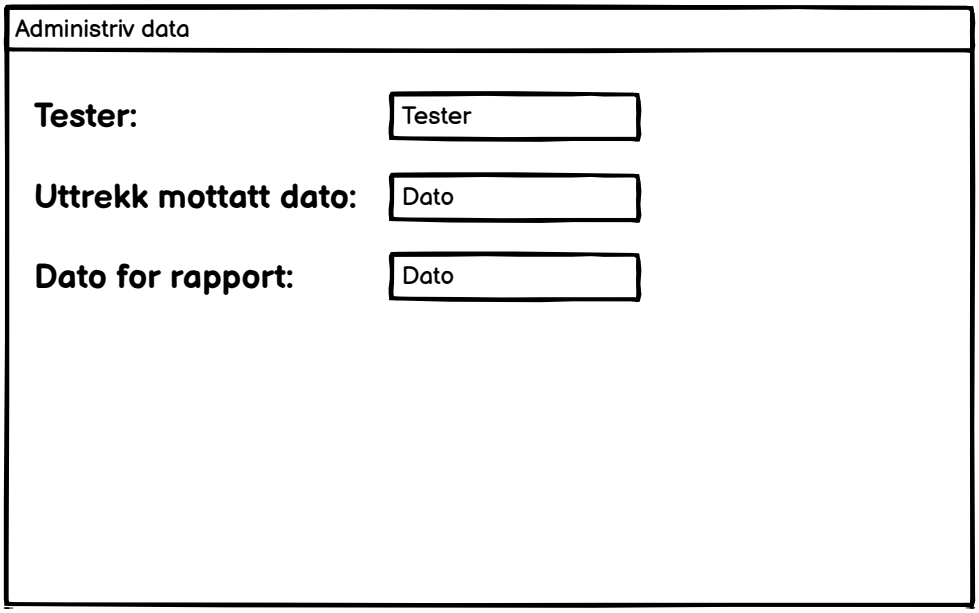
7 sider.

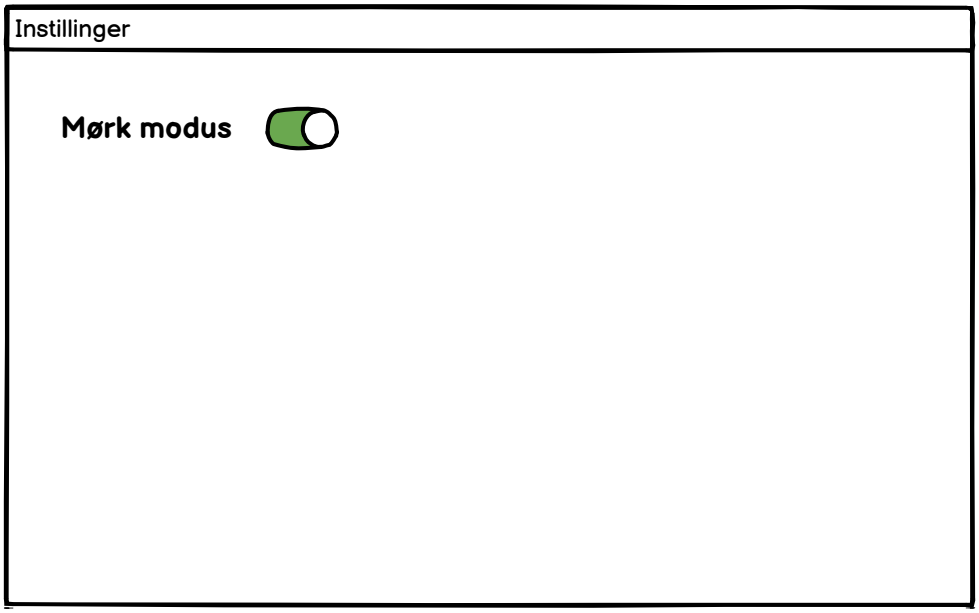

Archive Master
Instillinger
Om
Last opp arkivuttrekk
Last opp pakket uttrekk
Last opp ferdig testet uttrekk
Arkivuttrekk som skal testes:
Navn på arkivutrekk
Velg tester
Start testing
Ferdig testet arkivuttrekk:
Navn på arkivutrekk
Skriv rapport
Velg rapportformat:
docx
odf

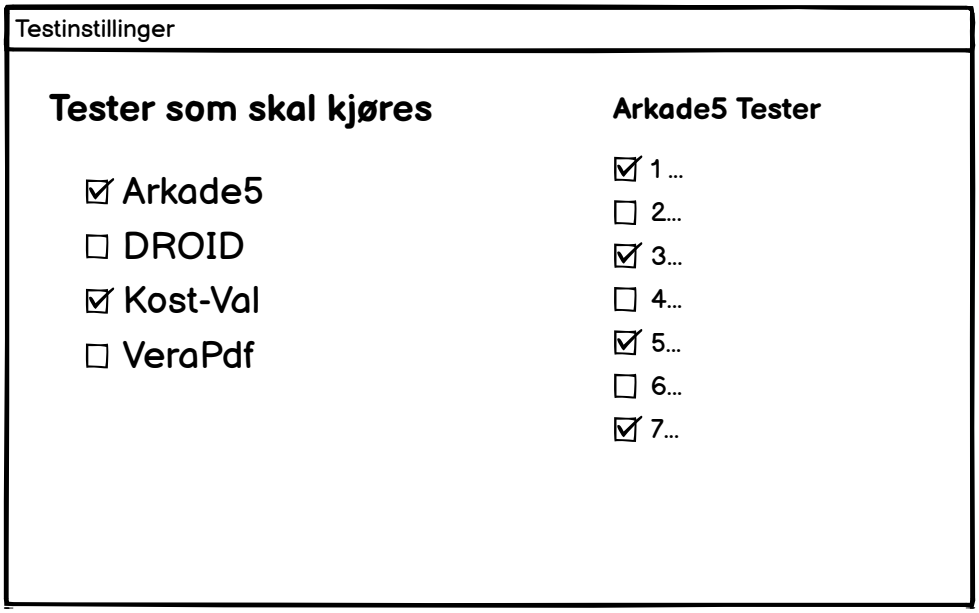

Archive Master
Instillinger
Om
Last opp arkivuttrekk
Last opp pakket uttrekk
Last opp ferdig testet uttrekk
Arkivuttrekk som skal testes:
uttrekk1
Velg tester
Start testing
Ferdig testet arkivuttrekk:
uttrekk1
Skriv rapport
Velg rapportformat:
docx
odf
Informasjon om uttrekket
- UttrekksID: 1234234
- Kommune/ kunde: Sel kommune
- Kontaktperson: Bob Jensen
- Uttrekksformat: noe
- Produksjonsdato for uttrekket: 01.02.20
- Uttrekk mottatt dato: (Manuelt)
- Test utført av: (Manuelt)
- Dato for rapport: (Manuelt)
Rediger


Archive Master
Instillinger
Om
Arkade5
Ferdig.
1000 avvik
VeraPdf
Ferdig.
123 avvik
Kost-Val
Tester...
Kjører tester...
Skriv rapport


Archive Master
Instillinger
Om
Arkade5
Ferdig.
1000 avvik
VeraPdf
Ferdig.
123 avvik
Kost-Val
Ferdig
32 avvik
Alle tester ferdig
Vis rapport


Administriv data
Tester:
Tester
Uttrekk mottatt dato:
Dato
Dato for rapport:
Dato


Instillinger
Mørk modus


Testinstillinger
Tester som skal kjøres
☑ Arkade5
☐ DROID
☑ Kost-Val
☐ VeraPdf
Arkade5 Tester
☑ 1 ...
☐ 2...
☑ 3...
☐ 4...
☑ 5...
☐ 6...
☑ 7...

Vedlegg I

Brukertest 2 testtabell

4 sider.

Test tabell nr. 1

Nr.	Funksjon/Kvalitet	Forutsetning	Aksjon/handling	Input	Forventet resultat
1	Åpne programmet via .jar filen.		Åpne arkivmester.jar filen.		Programmet starter og brukergrensesnittet viser forsiden med en meny, “Laste inn pakket uttrekk” knapp, andre deaktiverte knapper og en tom administrative data liste på høyre siden.
2	Last inn pakket uttrekk.	1	Klikke på “Last inn pakket uttrekk” knapp.	Mappe med .tar og metadata .xml fil.	Filutforsker åpnes, kan navigere til riktig mappe og åpne den. Administrative data oppdateres, “Start testing” og “Velg tester” -knappene blir aktivert.
2.1	Last inn annet pakket uttrekk.	2	Klikke på “Last inn pakket uttrekk” knappen igjen for å bytte gjeldende uttrekk.	Mappe med .tar og metadata .xml fil.	Samme forventet resultat som nr. 2, samtidig at forrige uttrekksdata blir fjernet og erstattet med det nye uttrekket.
2.2	Last inn ugyldig pakket uttrekk.	1	Klikke på “Last inn pakket uttrekk” knapp.	Mappe uten .tar og metadata .xml fil.	Feilmelding popper opp som forteller brukeren at “Mappen inneholder ikke .tar og .xml”.
3	Åpne innstillinger.	1	Klikke på “Innstillinger” knapp.		Brukergrensesnittet bytter til innstillinger siden. En liste over alle fil lokasjoner brukt av programmet skal vises med redigerings knapper, lagre og avbryt knapp.
3.1	Endre innstillinger.	3	Klikke på “Rediger fil lokasjon” knapp.	Ny fil lokasjon på datamaskinen.	Filutforsker åpnes, kan navigere til riktig lokasjon og velge den. Listen oppdateres, “Lagre innstillinger” knappen blir aktivert.
3.2	Lagre innstillinger.	3	Klikke på “Lagre innstillinger” knapp.		Den oppdaterte listen blir skrevet til konfigurasjonsfilen og programmet tar i bruk de nye forandringene. Brukergrensesnittet bytter til forsiden.
3.3	Tilbakestill innstillinger.	3	Klikke på “Tilbakestill”		Alle fillokasjoner blir tilbakestilt til standard.

			knapp.		
3.4	Avbryte forandringer.	3	Klikke på “Tilbake” knapp.		Ingen forandringer blir lagret og brukergrensesnittet bytter til forsiden.
4	Åpne administrative data.	1, 2	Klikke på “Rediger informasjon” knapp.		Brukergrensesnittet bytter til administrative data siden. En liste over uttrekkets administrative data, lagre og avbryt knapp vises.
4.1	Skrive/endre administrative data og lagre.	4	Skrive i tekstfeltet i andre kolonne i listen.	Ny administrative data.	Kan skrive tekst i tekstfeltene og klikke “Lagre administrative data” knappen. Brukergrensesnittet bytter til forsiden med oppdatert informasjon i listen på å høyre side.
4.2	Avbryte forandringer.	4	Klikke på “Avbryt” knapp.		Ingen forandringer blir lagret og brukergrensesnittet bytter til forsiden.
4.3	Avbryte ved å klikke på “Innstillinger” knappen.	4	Klikke på “Innstillinger” knapp.		Ingen forandringer blir lagret og brukergrensesnittet bytter til innstillinger siden.
5	Åpne valg av deltester.	1, 2	Klikke på “Velg tester” knapp.		Brukergrensesnittet bytter til velge deltester siden. En liste over deltester, lagre og avbryt knapp vises.
5.1	Velge egendefinert XQuery.	5	Huke av på en XQuery som skal kjøres.		Kan huke av en eller flere Xqueries.
5.2	Velge deltester og lagre.	5	Huke på eller av for å inkludere eller ekskludere en deltest.		Kan huke av eller på deltestene og klikke “Lagre tester” knappen. Brukergrensesnittet bytter til forsiden og programmet inkluderer nå bare de deltestene som er huket av på.
5.3	Avbryte forandringer.	5	Klikke på “Avbryt” knapp.		Ingen forandringer blir lagret og brukergrensesnittet bytter til forsiden.

5.4	Avbryte ved å klikke på “Innstillinger” knappen.	5	Klikke på “Innstillinger” knapp.		Ingen forandringer blir lagret og brukergrensesnittet bytter til innstillinger siden.
6	Start testing.	1, 2	Klikke på “Start test” knapp.		Brukergrensesnittet bytter til testsiden. En liste over alle deltestene med status, deaktivert “Lag rapport” knapp, deaktivert “Pakk til AIP knapp” og “Teste et nytt uttrekk” knapp vises. Det er også en helhetlig test status og en kopi av uttrekkets metadata på høyresiden. Testingen av uttrekket startes og foregår i bakgrunnen. “Innstillinger” knappen blir deaktivert. “Lag rapport” knappen blir aktivert når testen er fullført.
6.1	Teststatuser blir oppdatert.	6	Automatisk av programmet.		Statusene til hver deltest skal være “Ingen” hvis ekskludert, “Venter” hvis inkludert, men venter på en annen deltest til å bli ferdig, “Kjører” hvis er i gang, det kan bare være en deltest som kjøres om gangen, og “Ferdig” hvis deltesten har kjørt og er ferdig. Over “Lag rapport” knappen skal det stå “Kjører tester..” og en animasjon kjøres.
6.2	Egendefinerte XQueries kjøres.	6	Automatisk.		Dersom en eller flere egendefinerte XQueries har blitt valgt blir disse kjørt etter de fire standard-testene. Resultatet fra disse havner i egne .txt filer i mappen for gjeldene uttrekk i temp mappen.
6.3	Generer rapport.	6	Klikke på “Lag rapport” knapp.		Automatisk skriving av sluttrapporten begynnes og når fullført vil “Pakk til AIP” knappen bli aktivert. Teststatusen over knappene gir tilbakemelding når rapporten er klar med dens fil-lokasjon.
6.4	Pakk til AIP	6.3	Klikke på “Pakk til AIP” knappen.		Teksten over knappen endres til “Pakker til AIP” og en animasjon kjøres. Etter noen sekunder

					endres teksten til lokasjonen av AIP mappen. AIP mappen finnes i temp mappen under arkivet som testes.
6.5	Test nytt uttrekk	6	Klikke på “Test nytt uttrekk” knappen.		Programmet vil hoppe til forsiden igjen, resette programmet og være klart til å teste et nytt uttrekk.
7	Programmet skal ikke krasje.	1			Programmet gjør ingen uforventede avslutninger eller frysninger. Hvis det er noe feil så skal det komme melding i brukergrensesnittet.
8	Programmet skal kunne kjøre over flere dager.	1			Programmet gjør ingen uforventede avslutninger eller frysninger under testing av uttrekk som kan vare i flere dager.
9	Programmet skal kjøres uten internett.	1			Programmet skal kunne åpnes og brukes til sitt fulle uten internettilkobling.
10	Alle bruker handlinger skal bli utført på maksimum 1 sekund, utenom “test” og “lag rapport”.	1	Hvilken som helst handling.		Når man for eksempel klikker på en knapp eller laster opp uttrekk så skal den handlingen ikke ta mer enn 1 sekund på å bli utført.

M. Sustad, O. Keogh, T. Ellefsen, E. Bjarnason

Automatisk rapportskriving og testing av digitale arkiv

NTNU
Kunnskap for en bedre verden