Utvikling av nettleserspill
Systemdokumentasjon

Versjon 1.0

Revisjonshistorie

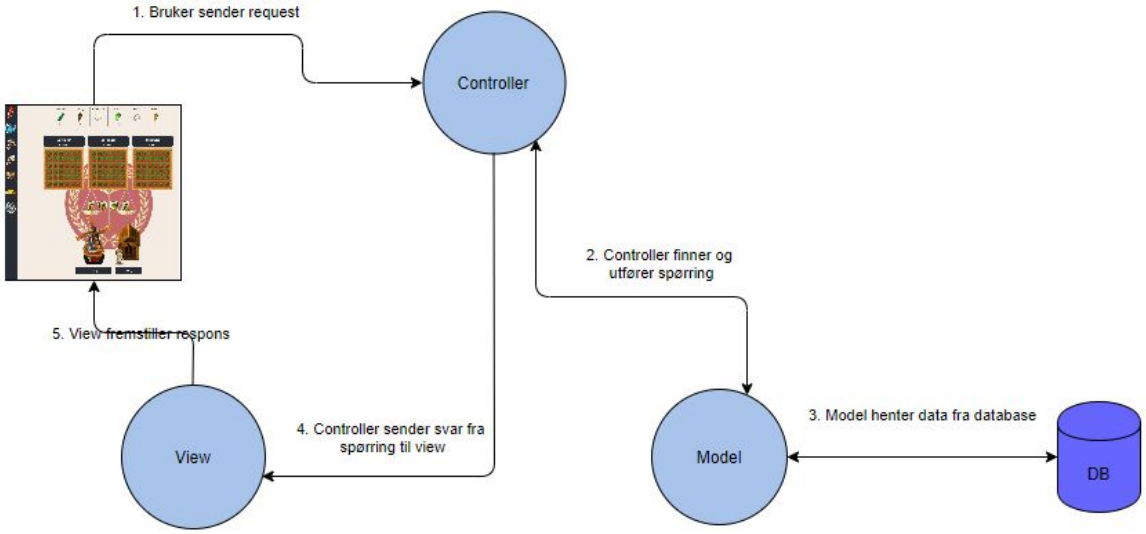
Dato	Versjon	Beskrivelse	Forfatter
20/mai/19	1.0	Siste versjon	Karl Peter Skjelvik Trond Ugelstad Martin Wangen

Innholdsfortegnelse

1. Introduksjon

Dette dokumentet er skrevet i forbindelse med bacheloroppgave 103, våren 2019. Hensikten med dette dokumentet er å gi en overordnet oversikt over systemet til produktet.

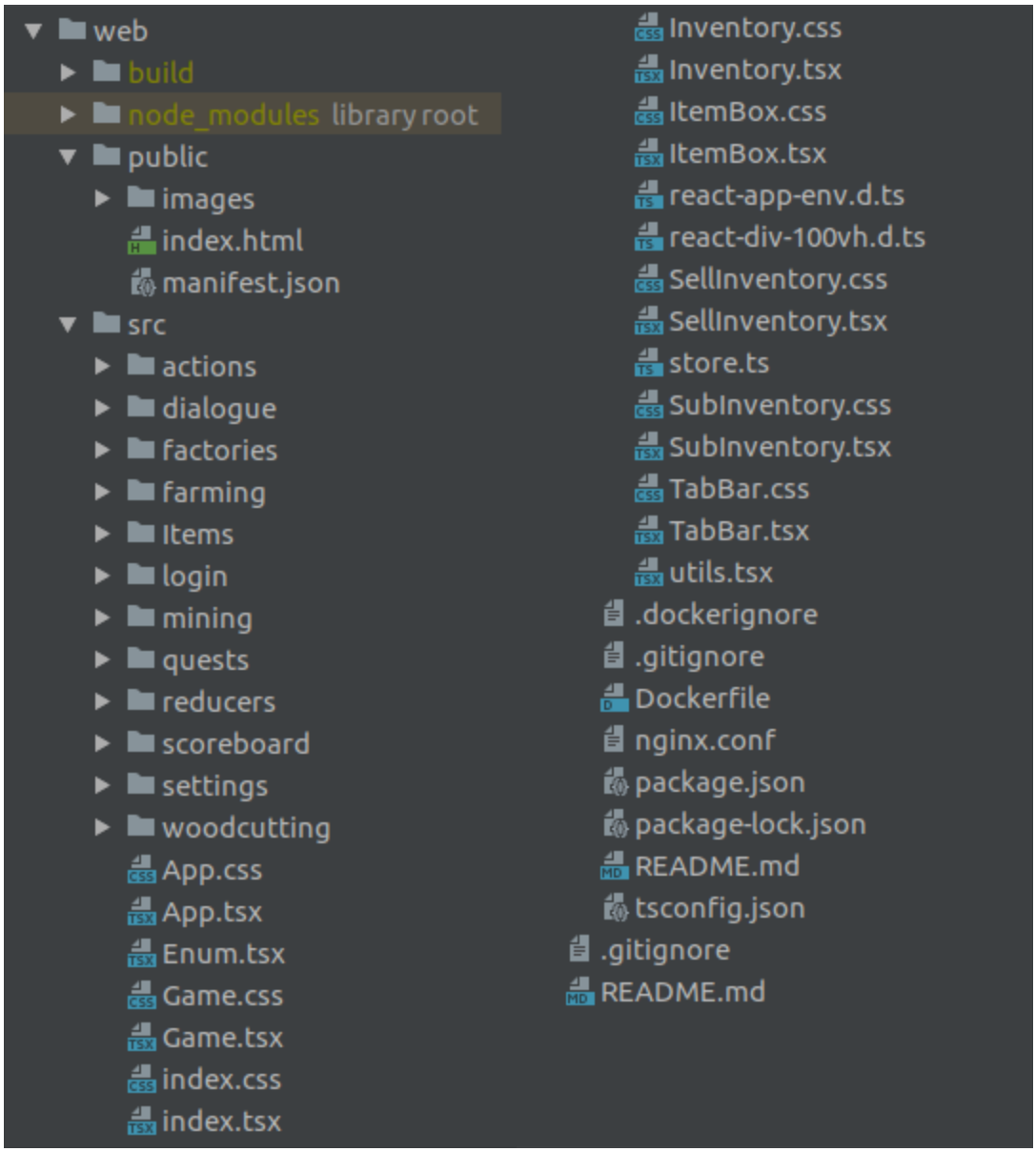
2. Arkitektur



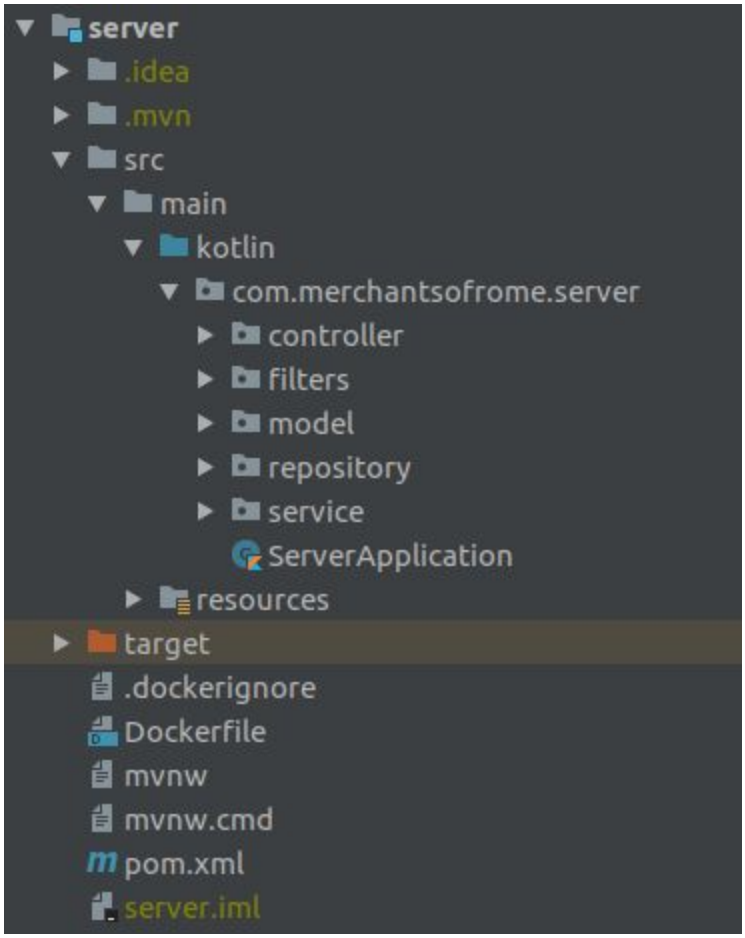
Figuren over viser hvordan Model-View-Controller. “Model” holder på data, “View” presenterer data og “Controller” utfører funksjonalitet.

3. Prosjektstruktur

Frontend er kategorisert først i 2 mapper, “public” som inneholder de statiske delene av nettsiden, herunder “index.html” og bilder, og “src” som inneholder hovedsakelig TypeScript-filer og CSS-filer. “build” og “node_modules” er mapper som genereres under bygging. Under “src” er filene for det meste kategorisert ut i fra hvor i spillet filene brukes.

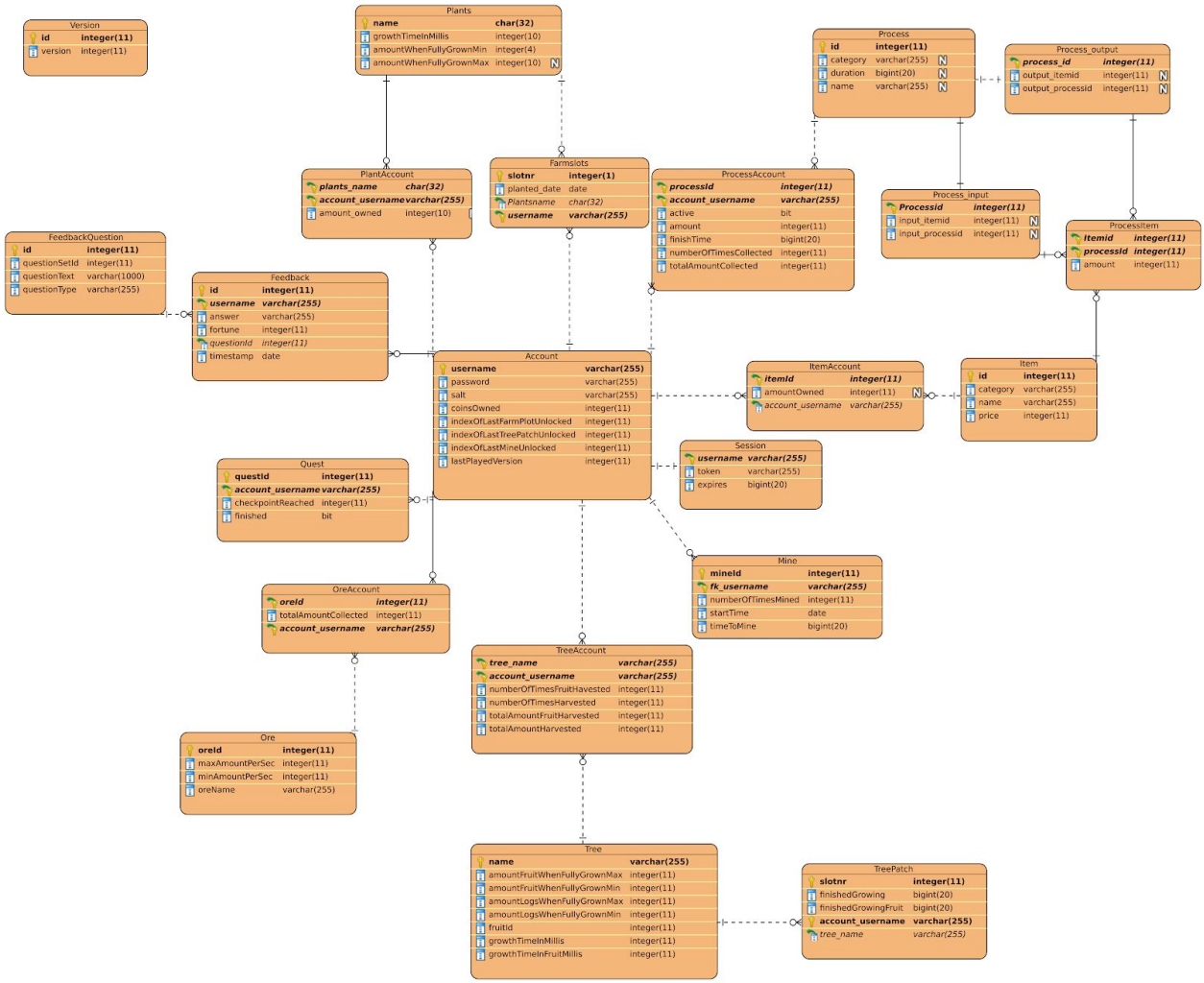


Backend ligger all koden i ”src/main/kotlin/com.merchantsofrome.server/”, og den er her kategorisert med “model” og “controller” fra MVC-modellen, “service” for ekstra logikk til komponenter fra “controller”, “filters” for filtre, og “repository” for koden som går direkte mot databasen. I “src/main/resources” ligger ressurser som benyttes av serveren, som for eksempel informasjon om tilkobling til databasen. På rota ligger konfigurasjonsfiler for blant annet Maven og Docker.



Disse bildene er et overordnede bilder over hvordan prosjektet er strukturert. Se vedlegg “Prosjektstruktur.png” for en mer detaljert visning.

4. Databasemodell



5. Server-tjenester

URL	Metode	Beskrivelse
AccountController		
/api/createNewAccount	POST	Lager ny bruker
/api/accounts	GET	Henter bruker via brukernavn i header
/api/accounts/rules	GET	Henter hvorvidt en bruker har godtatt reglementet

/api/accounts/rules	POST	Godtar reglementet for en bruker
/api/accounts/changepassword	PUT	Bytter passord for en bruker
/api/accounts/scoreboard/coins	GET	Henter informasjon om hvor mye penger alle brukere har
/api/accounts/scoreboard/wheatHarvested	GET	Henter informasjon om hvor mye hvete alle brukere har høstet
/api/accounts/scoreboard/fortune	GET	Henter informasjon om hvor mye penger alle brukere har tjent tilsammen
DBController		
/api/dbcontroller	POST	Oppdaterer databasen
FarmPlotController		
/api/farmPlots	GET	Henter alle farmplots for en bruker
/api/farmPlots/plant	PUT	Planter noe i en eller flere farmplots
/api/farmPlots/harvest	PUT	Høster fra en eller flere farmplots
/api/farmPlots/buyNewFarmPlot	POST	Kjøper en ny farmplot for en bruker
FeedbackController		
/api/feedback/questions	GET	Henter spørsmål til brukeren
/api/feedback/submit	POST	Lagrer brukerens svar
ItemController		
/api/inventory	GET	Henter beholdningen til brukeren
/api/items/prices	GET	Henter prisene på tingene i beholdningen
/api/inventory/sell	PUT	Når bruker selger en ting i beholdningen
MineController		
/api/mining/collectfrommine	POST	Når bruker henter fra gruvene
/api/mining/buymine	POST	Når bruker kjøper ny gruve

/api/mining/getallminesatstart	GET	Henter alle gruvene til en bruker ved start
PlantController		
/api/plants/{name}	GET	Henter planten basert på variabelen
/api/plants/unlocked	GET	Henter plantene som en bruker har åpnet
/api/plants/harvested	GET	Henter totalt hvor mye bruker har høstet av hver plante
ProcessController		
/api/process	GET	Henter alle aktive prosesser for bruker
/api/process/start	PUT	Når bruker starter en prosess
/api/process/collect	PUT	Når bruker henter ferdig produkt fra prosess
QuestController		
/api/quests	GET	Henter alle quester som bruker har
/api/quests/{questnr}/{checkpoint}	PUT	Når bruker gjør et checkpoint i en quest
SessionController		
/api/login	POST	Når bruker logger inn
TreeController		
/api/trees/harvested	GET	Henter totalt hvor mye bruker har høstet av hver frukt
TreePatchController		
/api/treePatches	GET	Henter alle tree patch som bruker har
/api/treePatches/plant	PUT	Når bruker planter et tre
/api/treePatches/harvestFruit	PUT	Når bruker høster frukt fra et tre
/api/treePatches/chopTree	PUT	Når bruker kapper et tre
/api/treePatches/buyNewTreePatch	POST	Når bruker kjøper ny tree patch

103

UpgradeController		
/api/upgrade/seeds/{name}	PUT	Når bruker kjøper nytt frø
/api/upgrade/fruits/{name}	PUT	Når bruker kjøper ny frukt

6. Sikkerhet

https://merchantsofrome.com tjenes fra Kantegas egen server. Den benytter HTTP/2 som igjen betyr at det benyttes TLS og all data overført mellom klient og server er kryptert. Vi har derfor valgt å ikke hashe passordene frontend, ettersom vi likevel hasher backend før vi lagrer i databasen. Vi forstår likevel at enkelte brukere er skeptiske til at vi ikke hasher frontend da de ikke kan vite med sikkerhet at vi ikke tar vare på deres passord i klartekst, dog kan dette igjen ses på som en fordel da brukerne kan velge å bruke et nytt passord for å ikke røpe sine mest brukte passord, og på den måten hindre gjenbruk av passord.

Ved oppretting av bruker genereres en tilfeldig salt som legges til enden av passordet. Deretter hashes passord+salt før både salt og det hashede passordet lagres i databasen. Ved å legge til en unik salt for hver bruker sørger det for at to brukere som velger samme passord ikke får samme hash i databasen. Skulle passordene mot formodning lekkes vil da alle være entydige og må knekkes hver for seg. Det er også stor sannsynlighet for at ingen av passordene matcher noen av passordene fra kjente databaser med lekkede passord.

Som sesjonshåndtering benytter vi et filter som ligger utenpå REST-tjeneren vår. Ved innlogging opprettes sesjon som inneholder en access token, en autogenerert streng, brukernavn og gyldighetsrom som varer i 8 timer. Sammen med respons sendes access token dersom innlogging var suksessfull. Denne lagres statisk hos klienten, og sendes som header i alle fremtidige requests. Når en request når serveren sjekkes den i filteret om token og brukernavn kombinasjonen finnes i databasen. Om de gjør det settes gyldighetsrommet til 8 timer frem i tid, og requesten utføres av serveren.

Vi bruker i dag Jenkins for CI/CD. Ved bruk av Jenkins og plugins til Jenkins har vi beskyttelse mot cross-site scripting og SQL-injections. Ettersom vi bruker Hibernate har vi er vi

uansett beskyttet mot «native SQL» injections, men vi har har trolig lite beskyttelse mot HQL-injections. Vi har per dags dato ikke fattet tiltak da HQL er en god del mer restriktivt enn SQL og vi ikke har gjort det kjent at vi benytter Hibernate som ORM.

7. Installasjon og kjøring

For å kjøre spillet lokalt så trengs kun å kalle "main"-metoden i ServerApplication.kt, og å kalle "npm start" i mappen "web". Det er også mulig å bruke de to "Dockerfile" som ligger i web og server for å kjøre det opp i Docker. For selve https://merchantsofrome.com har vi fått assistanse fra Kantega AS, som har tatt seg av driftingen av serverene. Det fulle oppsettet er beskrevet under.

Det ble satt opp 3 servere for å hoste MerchantsOfRome:

- En databaseserver som holder prod og dev databasene
- En byggserver som kjører Jenkins
- En appserver som kjører prod og dev av selve spillet

Databaseserver:
Kjører 2 stk MariaDB-containere hvor den ene containeren holder prod-databasen og den andre dev-databasen

Byggserver:
Kjører 1stk Jenkins-container og 1stk Nginx-proxy-container for SSL-terminering foran Jenkins. Jenkins er konfigurert med 2 byggjobber, en for bygg og deploy til prod, og en tilsvarende for dev.
Dev-jobben bygger og deployer alt som commites til master-branch, mens prod-jobben bygger og deployer alle git tags/releases. Selve bygget og deploy blir utført av en Jenkins-slave som her er appserveren som hoster spillene.

Appserver:

Kjører som en single-node docker swarm for rullende oppgraderingsfunksjonalitet. Kjører både prod og dev-miljø i hver sine separate docker stacks. En stack (eller et miljø) består av en frontend-container og en backend-container, hvor hver disse kjører med 2 replicaer. Dvs. at serveren kjører totalt 8 containere fordelt slik:

- 2stk dev frontend
- 2stk dev backend
- 2stk prod frontend
- 2stk prod backend

Foran appserver står et Nginx-failover-cluster hvor det er definert 2 vhoster, en for prod og en for dev. For SSL er det brukt LetsEncrypt med certbot.

8. Kontinuerlig integrasjon

CI/CD for produksjonen er opprettholdt gjennom Jenkins ved bruk av Docker Swarm. Dette gjelder både ved bruk av Pre-production sandbox og ved serveren som hoster spillet. For Pre-production sandbox bygger Jenkins til https://dev.merchantsofrome.com. Dette skjer hver gang det pushes til master. Når det pushes til master med tag bygger Jenkins til https://merchantsofrome.com, som er serveren som hoster spillet.