Sanna Kukkonen, Sebastian Lundin, Joona Nylander, Aarni Pesonen, Teemu Tallskog

Proma - Projektinhallintaohjelmisto

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintäteknologian tutkinto-ohjelma

Toteutusdokumentti

5.5.2022

Sisällys

1	Johdanto	1
2	Tuotteen vaatimukset	2
3	Käyttäjäroolit ja käyttötapaukset	3
4	Ohjelmiston tietomalli	4
5	Ohjelmiston rakenne	6
6	Ohjelmiston toiminta	9
7	Kehitysprosessi ja kehitysvaiheen tekniikat	11
8	Testauksesta	13
9	Yhteenveto	13

1 Johdanto

Toteutusdokumentissa käsitellään ryhmä 8:n OTP-1 ja OTP-2 toteutuksien aikana kehitettyä ohjelmistotuotantoprojektia Proma projektinhallintasovellusta. Dokumentissa määritellään sovelluksen toiminta, rakenne sekä tietomallit sellaisella tarkkuudella, joka antaa lukijalle selkeän kuvan ohjelmistotuotteesta.

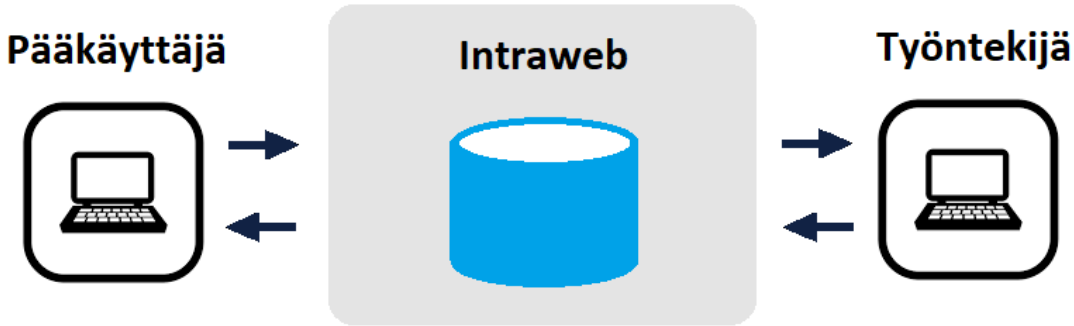
Proma on pääsääntöisesti yritysten sisäiseen käyttöön suunniteltu ohjelmisto, jonka avulla työntekijöiden on mahdollista suoraviivaistaa projektiensa ja niihin liittyvien tehtävien hallintaa, sekä henkilökohtaisten työaikatietojen tallentamista. Sovellus tarjoaa yhteisen projektinhallinta-alustan saman projektin parissa työskenteleville henkilöille.

Sovelluksen asennukseen ja käyttöön liittyvät ohjeet löytyvät kokonaisuudessaan erillisestä dokumentista Proma_käyttöohjeet. Dokumentti kattaa sovelluksen asennuksen ja käytön pääkäyttäjän sekä työntekijän näkökulmasta.

2 Tuotteen vaatimukset

Proman tarkoituksena on tarjota käyttäjilleen tehokas ja miellyttävä projektinhallintatyökalu. Sen kehitystyön lähtökohtana toimivat edellä mainittujen ominaisuuksien lisäksi käytettävyys sekä tietoturvallisuus. Sovelluksen käytön tulee olla nopeaa ja vaivatonta. Sovelluksen ei tule myöskään kerätä käyttäjistään henkilökohtaisia tietoja.

Käyttöliittymän on oltava toiminnallisuudeltaan selkeä ja sen on tarjottava käyttäjäkunnalle riittävä määrä opastusta tarvittaessa. Tärkeimmäksi määriteltyyn ominaisuuteen, käyttäjäkohtaiseen työaikakirjanpitoon, tulee kehitysvaiheessa suunnata kaikki tarvittavat resurssit. Käytön opittavuuteen ja muistettavuuteen on kiinnitettävä erityisesti huomiota. Mahdollisten virheiden määrä pyritään minimoimaan validoimalla syötteet sekä pitämällä sovelluksen mallimaailma erillään käyttöliittymästä. Tietoturvallisuutta lisää sovelluksen tarkoituksenmukainen käyttö yrityksen sisäverkossa.



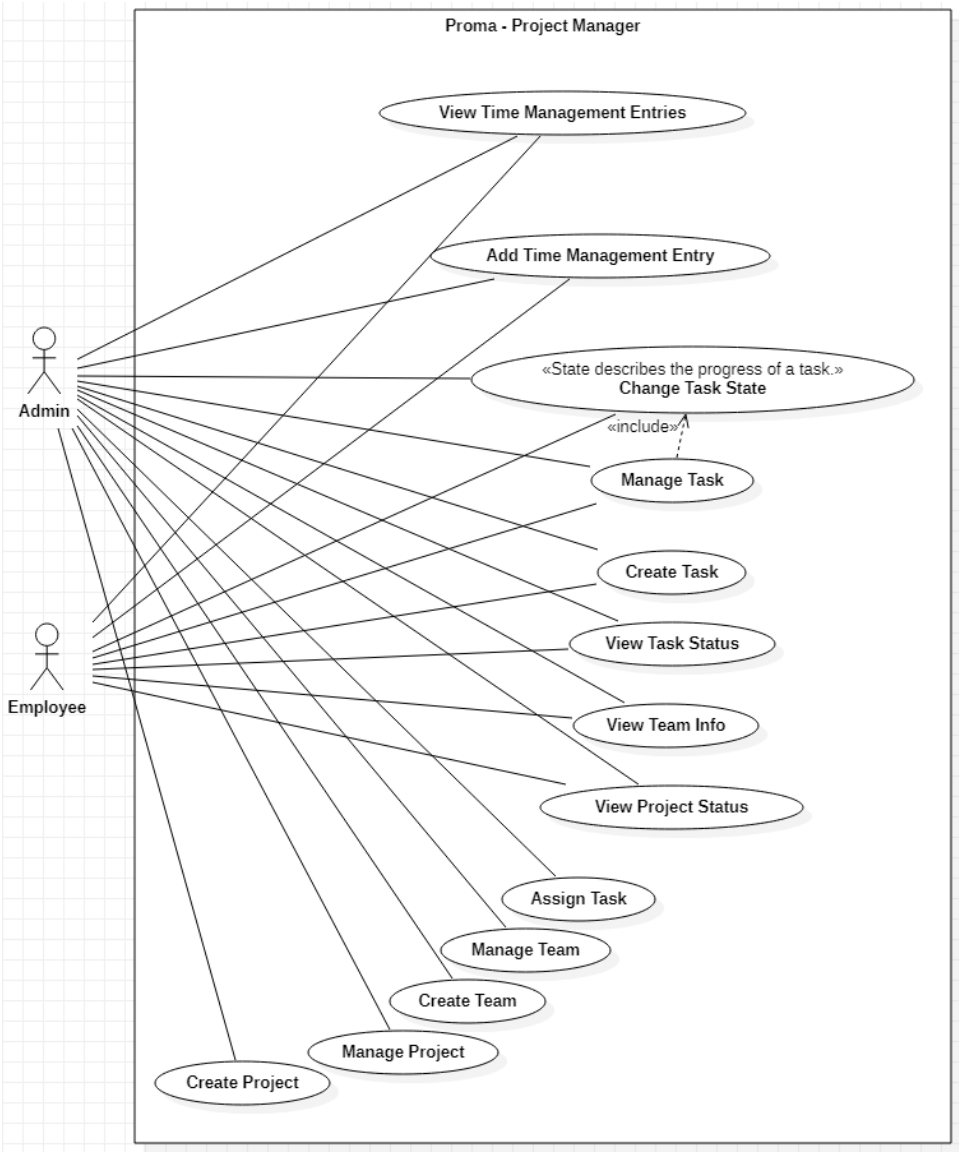
Kuva 1: Proma yrityksen käytössä

Tietokannan CRUD-operaatioita ei tule voida syöttää sovellukseen manuaalisesti, jotta SQL-injektiota ei pääse syntymään. Sovellukseen syötettyjä tietoja tulee pystyä muokkaamaan jälkikäteen. Tuotantoversiossa oltava mukana myös kattava muutoshistoria-listaus.

Ohjelma asennetaan työntekijän henkilökohtaiselle päätelaitteelle, josta sitä käytetään yrityksen sisäverkossa (Intraweb). Yrityksen sisäverkkoa projektityön yhteydessä simuloi Metropolian Educloud.

3 Käyttäjäroolit ja käyttötapaukset

Proma tukee kahdentyyppisiä käyttäjärooleja. Nämä ovat pääkäyttäjä (Admin) sekä työntekijä (Employee). Pääkäyttäjän oikeuksilla varustettu tili oikeuttaa uusien projektien luomiseen, sekä projektin alle määriteltyjen tehtävien osoittamisen yksittäisille työntekijöille, tai työntekijöistä koostuville ryhmille (Team).

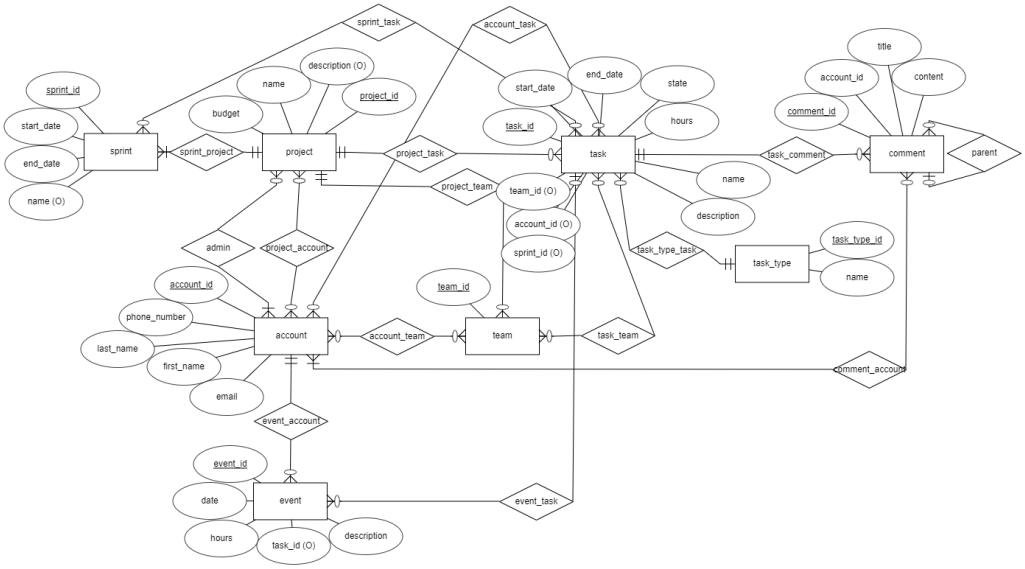


Kaavio 1: Käyttäjäroolien tarjoamat toiminnallisuudet

Tuotantoversiossa yksittäisen projektin luoja voi jakaa pääkäyttäjän käyttäjäroolin myös projektin yhteyteen liitetyille työntekijöille.

4 Ohjelmiston tietomalli

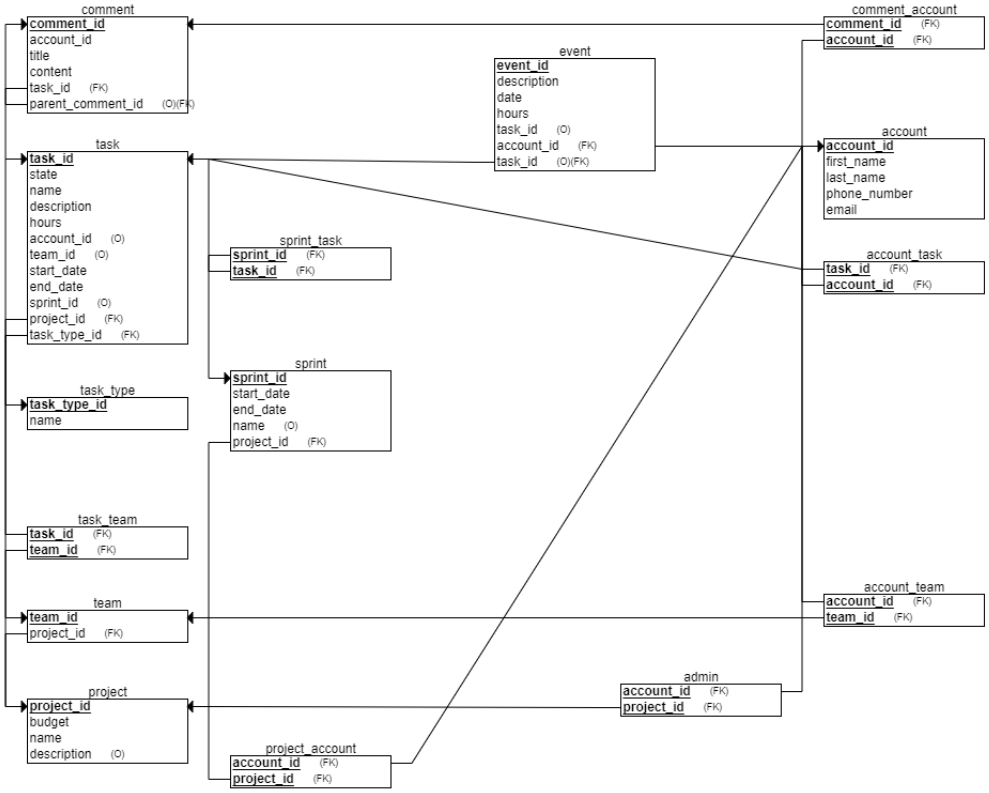
Alla on kuvattu Proman rakenteellinen toteutus ER- (entity-relation) sekä relaatiokaavioina. Tietokantaratkaisu on projektin mittakaavan huomioiden yksinkertainen ja tehokas.



Kuva 2: Tietokantakaavio

Tietokantarakenteen keskipisteessä on Account-luokkaan pohjautuva taulu, jonka ympärille muu kokonaisuus rakentuu. Pääkäyttäjän käyttäjäroolilla luotujen projektien yhteyteen saadaan liitettyä työntekijöitä eli tietokantaan tallennettuja ilmentymiä Account-luokasta. Projektien alle voidaan luoda scrum-tyyppisiä työjaksoja (Sprint), ryhmiä sekä tehtäviä (Task). Projektityöntekijöitä voidaan

jakaa edelleen ryhmiin. Tehtäviä voidaan osoittaa joko ryhmille tai yksittäisille käyttäjille. Muita tärkeitä huomioita tehtävistä ovat niille annettavissa oleva tyyppi sekä tila. Näiden avulla saadaan välitettyä tieto tehtävän luonteesta sekä sen parissa tehdyn työn edistymisestä. Tehtävien alle pystytään jättämään myös kommentteja, joihin voidaan myös vastata.

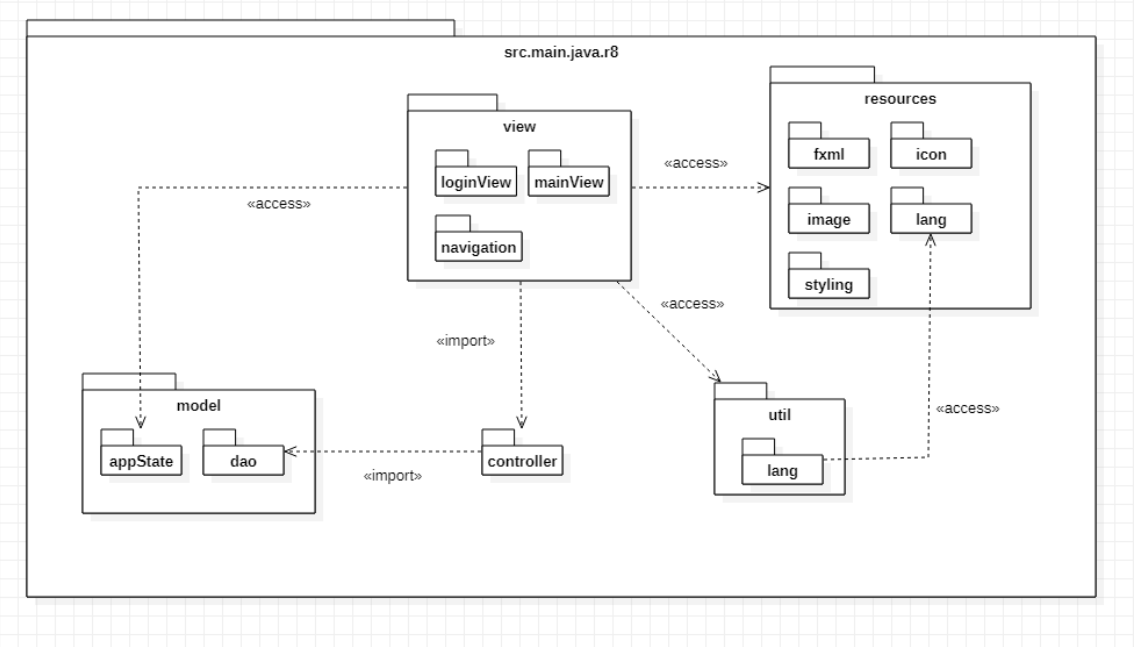


Kuva 3: Sovelluksen relaatiomalli

Yllä esitetty relaatiokaavio tarjoaa toisen näkökulman tietokantarakenteeseen, esitellen visuaalisesti niin kutsutut ”monen suhde moneen” yhteydet, joita muodostuu pareille projekti ja käyttäjätili, käyttäjätili ja ryhmä, käyttäjätili ja tehtävä, ryhmä ja tehtävä, sprint ja tehtävä sekä tehtävä ja kommentti. Kommenttitaulun suhde itseensä edustaa ominaisuutta kommenttiin vastaaminen. Käyttäjätilin ylös kirjaamat työtapahtumat tallentuvat omaan tauluunsa event.

5 Ohjelmiston rakenne

Sovelluksen rakenne hyödyntää MVC-mallia, jonka mukaisesti luokat on jaoteltu pakkauksiin. Proman toiminnallisuuden vaatimien lukuisten näkymien ohjainluokat on sijoitettu view-pakkauksen alle, näkymälle osoitettuun pakkaukseen. Käyttöliittymän ulkoasun rakenteen määrittelevät fxml-tiedostot sekä näiden muotoiluun käytetyt css-tiedostot löytyvät resursseista.

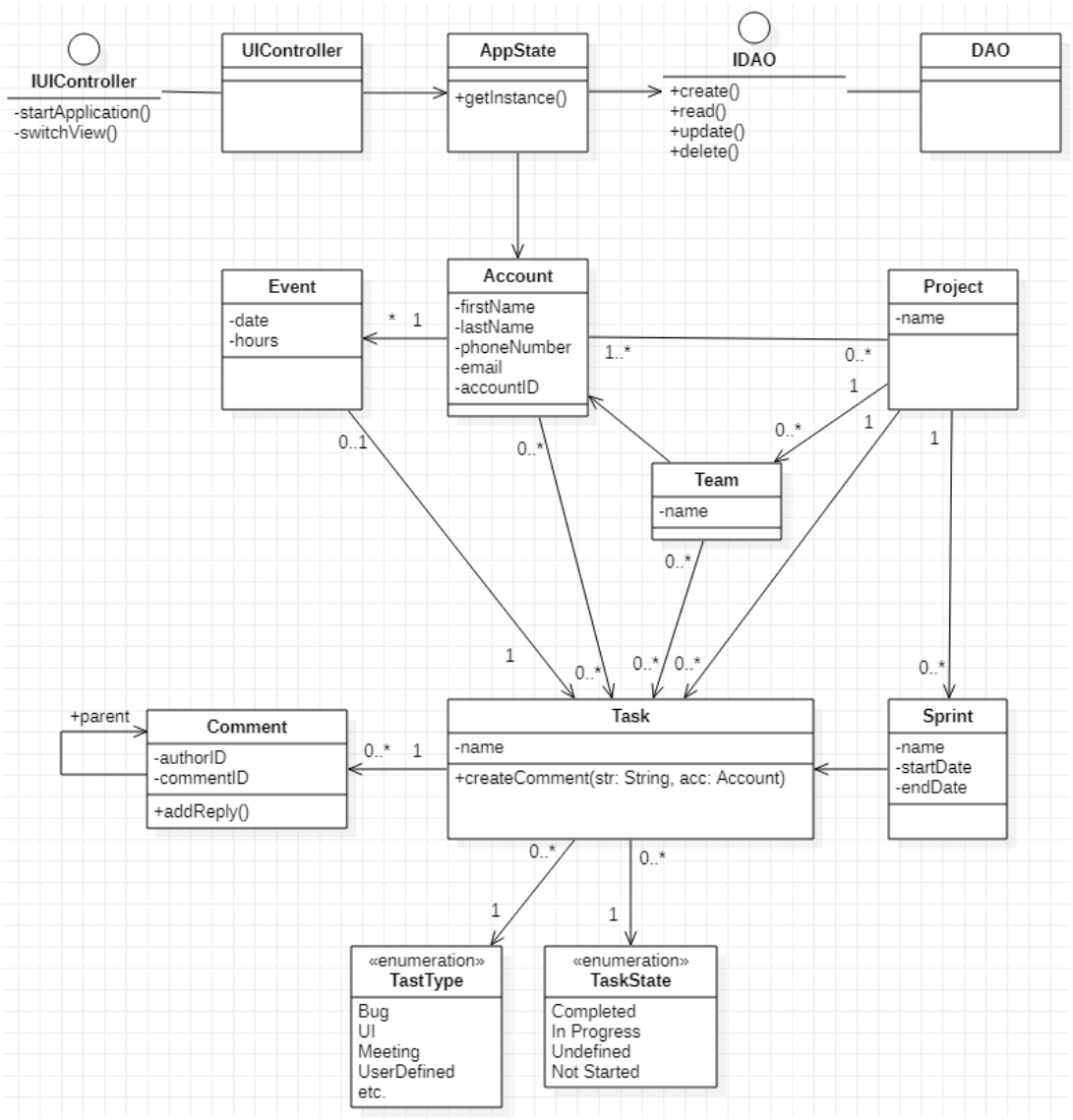


Kaavio 2: Pakkauskaavio

Pakkauskaavio kuvaa sovelluksen rakenteen yksinkertaisuudessaan. Pakkauksen view alle on jaoteltu sovelluksen yli kolmekymmentä näkymää. Kaaviossa esitetyt loginView sekä mainView edustavat niin kutsuttuja parent-näkymiä. Parent-näkymiin on määritelty ennalta elementit, joiden sisään kyetään lataamaan alinäkymiä. Esimerkkinä tästä toimii mainView, jonka vasemman- sekä yläreunan navigaatioelementit pysyvät staattisina, mutta muu näkymän sisältö voi vaihdella käyttäjän antamien navigaatiosyötteiden perusteella. Parent-näkymien pakkauksiin on sijoitettu niiden yhteyteen ladattavien alinäkymien ohjainluokat, mutta niitä ei ole esitetty kaaviossa tilankäytöllisistä syistä.

Model pakkaus sisältää sovelluksen toiminnallisuuden kannalta oleelliset luokat, kuten AppState, tietokantayhteyteen vaadittavan DAO:n abstraktin yliluokan sekä tätä perivät konkreettiset toteutukset. Tietokannan tauluja vastaavat luokat löytyvät luonnollisesti myös tämän pakkauksen alta. Alipakkaukseen util on sijoitettu kieleistykseen sekä työaikatietojen export-ominaisuuteen (.xls formaatti) liittyvät apuluokat. AppState -singletoniin tallennetaan sovelluksen käytön aikana vaadittavia tietoja, kuten sisäänkirjautuneen käyttäjän tili sekä navigaation apuviitteitä.

Proman luokkarakenne on suunniteltu käytännössä sisäkkäiseksi. Account-luokan ilmentymän sisään voidaan tallentaa lista käyttäjätilin yhteyteen assosioiduista projekti-olioista. Näiden alle voidaan viedä projektikohtaiset sprint-, ryhmä-, sekä task-tiedot. Työaikatiedot haetaan tietokannasta tarvittaessa yksittäisen käyttäjätilin, projektin tai sprintin perusteella.

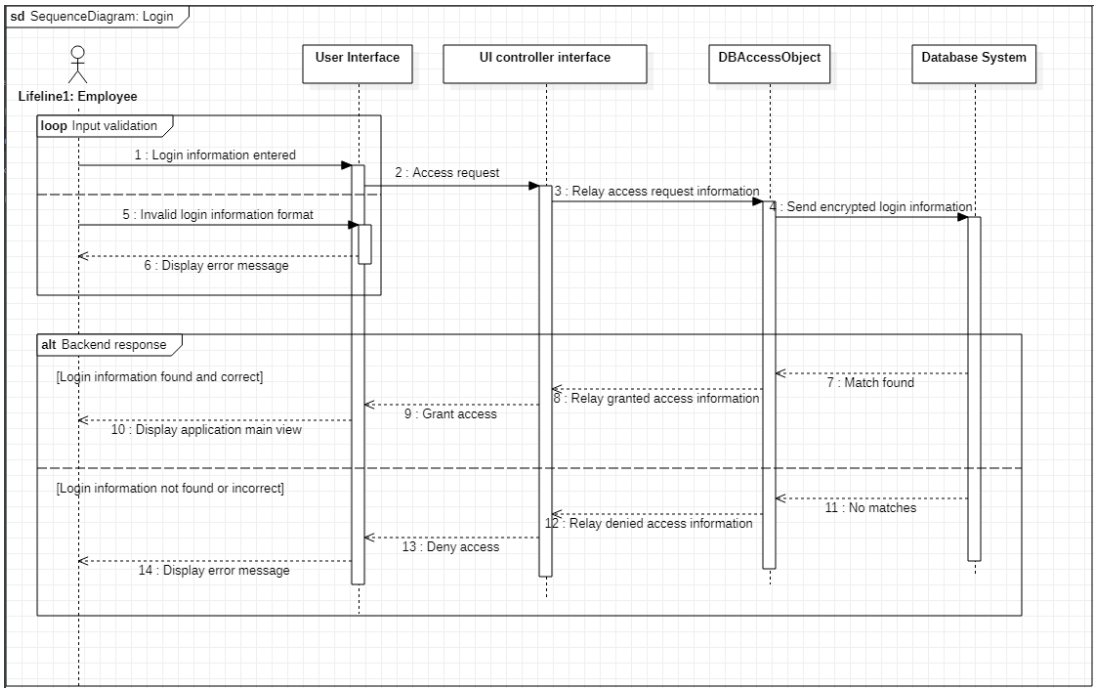


Kaavio 3: Sovelluksen luokkarakenne

Kun sovellus käynnistetään, ladataan ensimmäisenä sisäänkirjautumisnäkymä. Näkymän ohjain luo käynnistyksen yhteydessä AppState -singletonin, jota hyödynnetään tiedonsiirtoon käyttöliittymän eri näkymien välillä. Käyttöliittymän näkymien ja tietokantayhteyden tiedonvälittäjänä toimii Controller-niminen luokka, joka sisältää viitteet abstraktia DAO-luokkaa periviin konkreettisiin DAO-toteutuksiin. Näin ollen tietokantahakujen yhteydessä käyttöliittymälle ei tarvitse tarjota suoraa yhteyttä DAO-olioihin, eikä paljastaa ylimääräisiä hakumetodeja.

6 Ohjelmiston toiminta

Proman toiminnan kuvaukseen valikoitiin käyttötapauksiksi sisäänkirjautuminen sekä olemassa olevan tehtävän tilan muokkaaminen. Kaikki sovelluksen käyttötapaukset noudattavat läheisesti alla esitettyjen kaavioiden mukaisia toimintamalleja, jotka antavat riittävän kattavan kuvan yleisestä toiminnallisuudesta. Sisäänkirjautumisen sekvenssissä tulee käytyä läpi vaiheet syötteiden vastaanottamisesta tietokannan paluuviestin mukaisen palautteen esittämiseen käyttöliittymässä.



Kaavio 4: Sisäänkirjautuminen sekvenssikaaviona

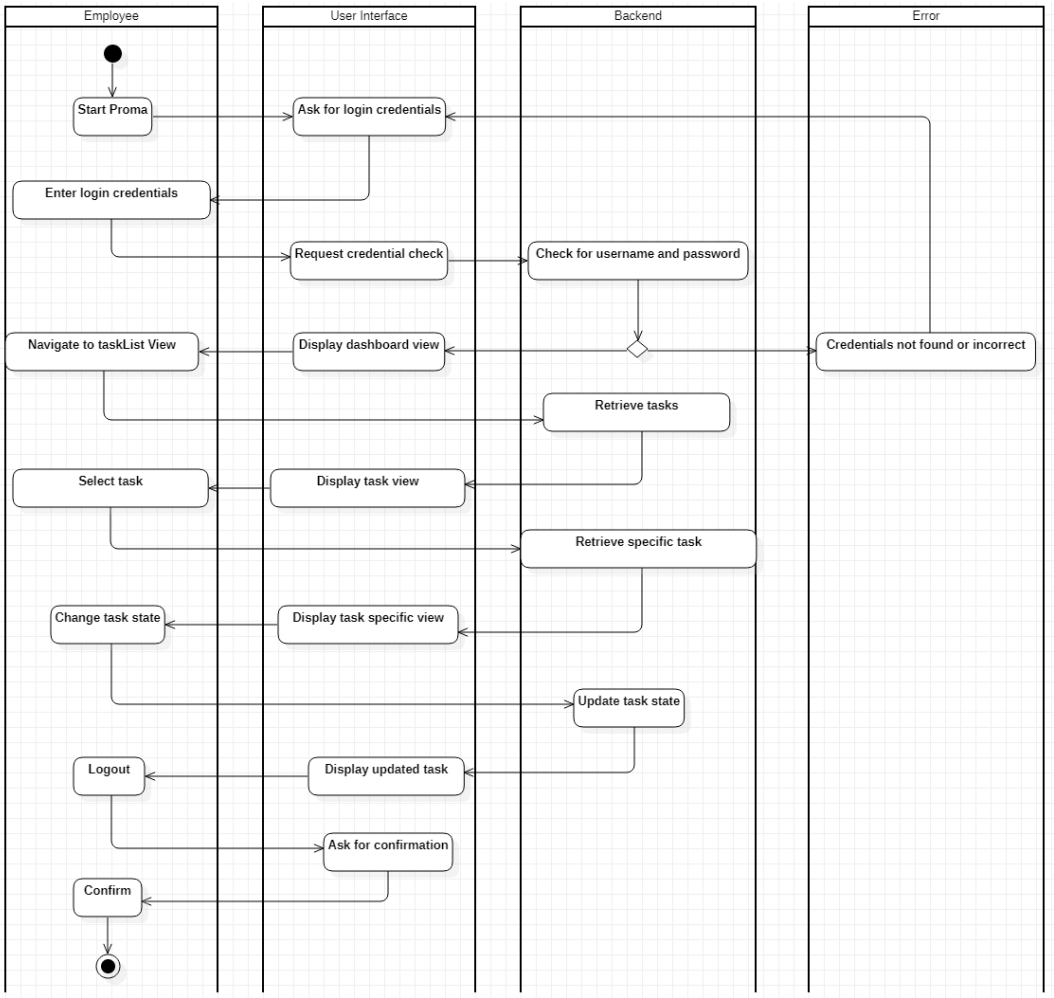
Kaavion vaiheet eriteltynä:

- Käyttäjä syöttää kirjautumistiedot. Jos syöte ei mene läpi validoinnista, näytetään virheilmoitus.

- Syötteiden formaatin ollessa hyväksytty, käyttöliittymän kontrolleri vie tiedot DBAO:lle, joka salaa kirjautumistiedot sekä lähettää kyselyn tietokantajärjestelmälle.

- Syötteiden vastatessa tietokannasta löytyvää käyttäjätunnus-/salasana-paria, palautetaan käyttäjän tiedot tietokannasta. Epäonnistuneesta kirjautumisyrityksestä välitetään tieto takaisin käyttöliittymään, joka johtaa virheilmoituksen esittämiseen. Vastineen löytyessä, AppState tallentaa tietokannasta palautuneet tiedot ja ilmoittaa käyttöliittymälle, joka avaa sovelluksen varsinaisen työnäkymän.

Olemassa olevan tehtävän tilan muokkaaminen on esitetty aktiviteettikaaviona. Lähtötilanteen oletetaan vaativan sovelluksen käynnistämisen.



Kaavio 5: Aktiviteettikaavio, tehtävän tilan muokkaaminen

Aktiviteettikaavion läpikäynti vaiheittain:

- Ohjelmiston käynnistäminen ja kirjautumistietojen syöttäminen. Jos tiedoille ei löydy vastinetta tietokannasta näytetään virheilmoitus. Vastinparin löytyessä sisäänkirjautumisen onnistuminen ja käyttöliittymän näkymän vaihto.

- Navigointi tehtävälistausnäkymään, tietokantahaku ja haun mukaisten tulosten esittäminen käyttöliittymässä.

- Halutun tehtävän valitseminen listasta ja tehtävän tietojen esitys käyttöliittymässä.

- Tehtävän tilan muokkaaminen, muutoksen vieminen tietokantaan, päivittyneen tilan esittäminen käyttöliittymässä.

- Uloskirjautuminen ja toiminnon suorittamisen vahvistus.

7 Kehitysprosessi ja kehitysvaiheen tekniikat

Proman kehitys lähti liikkeelle kattavalla suunnitteluvaiheella. Projekti-idean synnyttyä suunnittelu aloitettiin tietokannan rakenteesta. Tietokantasuunnittelun pääasiallisena työkaluna käytettiin ERDPlus web-sovellusta. Yksilö-yhteys eli ER-malli (entity-relationship) laadittiin ensin, joka kääntyi vaivattomasti samalla sovelluksella relaatiomalliksi. Tietokantamallinuksen tuotos säilyi kehitystyön ajan lähes muuttumattomana, muutamaa pientä lisäystä lukuun ottamatta.

Käyttöliittymän sekä ohjelmiston rakenteen suunnittelu suoritettiin yhtäaikaisesti. Käyttöliittymän malli työstettiin ensin JustInMind-sovelluksella, jonka pohjalta aloitettiin kehitys JavaFX:n SceneBuilderilla. Tämän jälkeen päätettiin ryhmäläisten henkilökohtaisista vastuualueista, sekä aloitettiin käyttäjätarinoiden vaatimusten mukaisten työtehtävien suorittaminen. Kahden ensimmäisen sprintin aikana työstettiin lähes yksinomaan projektin pohjaa, sillä jokainen käyttäjätarina sisälsi lukuisia yhteisiä vaatimuksia, kuten tietokantaintegraation. DAO:n toiminnallisuuden valmistuttua päästiin työstämään käyttöliittymän ja

tietokannan yhteistoimintaa. Ensimmäinen käyttäjätarina saatiin valmiiksi suhteellisen myöhäisessä vaiheessa, vasta neljännen sprintin puolella. Sovelluksen osien kehitystyö oli kauttaaltaan modulaarista ja palikat asettuivat kohdilleen OTP1-toteutuksen loppupäässä.

OTP2-toteutus aloitettiin keskittymällä koodin refaktorointiin ja sovelluksen toiminnallisuuden viimeistelyyn. Yhteys käyttöliittymän ja tietokannan välillä oli aikaisemmin toteutettu AppState:n välityksellä. Yhteys delegoitiin refaktoroinnin yhteydessä omalle ohjainluokalle. Jakson loppupuolella DAO:t kävivät läpi suuren refaktoroinnin, jossa hyödynnettiin Javan abstraktia luokkatoteutusta geneerisen CRUD-metodit sisältävän DAO-luokan luomiseen. Konkreettisille DAO-toteutuksiin sisällytettiin vain metodit, joiden tarve oli luokkakohtaista. Toteutusjaksolla sovellukseen lisättiin lokalisaatio-ominaisuus, joka lisäsi UI:n englanninkielisen tekstityksen rinnalle suomenkielisen vaihtoehdon. Tämän myötä UI:n elementtejä jouduttiin muokkaamaan, sillä sanojen suomenkieliset vastineet vaativat usein enemmän rvitilaa. Esimerkiksi painikkeiden koko ei automaattisesti skaalautunut sisällön mukaan. Suurin osa käyttäjätarinoista saatiin valmiiksi jakson loppupuolella. Toteuttamatta jäivät dynaamiset tietokantamuutosten laukaisemat, käyttäjälle esitettävät ilmoitukset sekä niin kutsuttujen tooltippien tekstitys.

Projektin tietokantana toimii MariaDB, johon otetaan yhteys Hibernatella. Jokaisella ryhmän jäsenellä oli käytössään paikallinen tietokanta testausta varten ja sovelluksen päätietokanta pyöri Educloudissa. Käyttäjien salasanojen salaukseen käytettiin bCrypt kirjastoa, ennen kuin ne vietiin tietokantaan. ControlsFX kirjastoa hyödynnettiin useiden UI elementtien luomiseen.

8 Testauksesta

Model- ja DAO luokan testit suoritettiin JUnit:illa. Jatkuva integraatio toteutettiin Jenkinsin avulla, joka oli ryhmälle uusi tuttavuus. Kun Jenkinsin käyttöön päästiin sisään, se osoittautui hyödylliseksi työkaluksi. JavaFX:lle tehtiin testejä, mutta näitä ei saatu Jenkinssillä suoritettua, eikä näihin uhrattu sen enempää aikaa. Sovelluksen luonteen huomioon ottaen nämä olisivat todennäköisesti olleet testeistä hyödyllisimmät, sillä suurin osa UI:n toiminnallisuudesta olisi hyötynyt testeistä. Käyttöliittymän testaus päädyttiin suorittamaan manuaalisesti kehitystyön yhteydessä ja havaitut bugit yritettiin korjata pääsääntöisesti heti, kun sellaisia havaittiin. DAO-testauksen vuoksi tietokantayhteydestä vastaavien luokkien kanssa ei tullu kehitystyön aikana suurempia ongelmia.

9 Yhteenveto

Projektin aihevalinta oli kunnianhimoinen ja sen valmiiksi saattamiseksi vaadittava työmäärä suuri. Kehitystyö suunniteltiin kuitenkin huomioiden keskimääräistä pidempi, kahden periodin mittainen kehitysaika. Suunnitteluvaiheessa ei kuitenkaan kyetty huomioimaan OTP-2 toteutuksen aikana suoritettavien muiden kurssien tehtävien tekemiseen kuluvaa ylimääräistä aikaa, jonka johdosta projektin työstämiseen ei jäänyt sen mittakaavan edellyttämää aikaa.

Käyttöliittymän kehityslähtökohtina käytettiin käyttäjäkeskeisen suunnittelun ohjeistoa ja menetelmiä. Tämänhetkinen käyttöliittymän ulkoasu sekä ohjelmiston MVC-mallin model-kokonaisuus noudattavat läheisesti periodin alussa tehtyä suunnitelmaa. Sovelluksen vaatimukset käyttöliittymälle ja sen toteuttamiseen kuluva aika kuitenkin aliarvioitiin suunnitteluvaiheessa. Tätä kuvastaa hyvin alun perin noin kymmenen kaavaillun näkymän lukumäärän kasvu kehitystyön edetessä yli kolmeenkymmeneen. Tietokannan rakenteeseen tehtiin kehitysprosessin aikana muutaman lisäys, mutta kokonaisuus vastaa läheisesti alkuperäistä suunnitelmaa.

OTP1 -toteutuksen kuluessa saatiin kasaan sovelluksen pohja sekä perustoiminnallisuus. Seuraavan periodin puolelle jäi toiminnallisuuden hienosäätö, edistyneempien ominaisuuksien implementointi sekä käyttöliittymän näkymien sisällön tarkennus ja viimeistely. Projektin toisen vaiheen kehitystyö painottuikin refaktorointiin, viimeistelyyn ja lokalisaatioon sekä dokumentointiin.

Pääasiallisiksi sovelluksen jatkokehityskohteiksi jäävät tietokannan muutoksia kuunteleva notifikaatio- ja päivitystoiminto, mahdollisten projektipohjien (template) integroiminen, tooltippien viimeistely, käyttöliittymän väriteemat. Jos tietokannassa tapahtuneet muutokset aiheuttaisivat sovelluksen tietojen automaattisen päivittymisen, tietokantaan ei tarvitsisi näkymien latauksen yhteydessä suorittaa uusia hakuja, vaan tiedot voitaisiin tällöin hakea paikallisesta muistista.

Sovelluksen nykyversiota käyttäessä nousee esille joidenkin käyttäjätarinoiden määrittelemien tehtävien suorituspolkujen kankeus. Tämä on seurausta projektin kehitystyön aikaisesta näennäisestä laajuuden kasvusta. Suunnitteluvaiheessa ei osattu ottaa huomioon jokaisen käyttöliittymän näkymän optimaalista suorituspolkua. Toisaalta tärkeimpänä pidetystä ajanhallintanäkymästä saatiin helppokäyttöinen, informatiivinen ja tehokas. Jatkokehityksen näkökulmasta sovelluksen käyttöliittymä hyötyisi uudesta iteraatiokierroksesta, jotta sen käyttäjäystävällisyyttä saataisiin kokonaisuudessaan parannettua.