



Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Eric Keränen, Nico Järvinen, Jere Salmensaari, Teemu Viljanen, Matias Vainio

OTP1 & 2 – Muistiinpanosovellus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Ohjelmistotuotantoprojekti 1 & 2

7.5.2021

Tekijät Sivumäärä Aika	Eric Keränen, Nico Järvinen, Jere Salmensaari, Teemu Viljanen, Matias Vainio 31 sivua 7.5.2021
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Opintojakso	Ohjelmistotuotantoprojekti
<p>Tiivistelmä</p> <p>Projektissa on toteutettu muistiinpanosovellus, joka on yksinkertainen tekstitiedostojen tekemiseen ja tallennukseen käytettävä ohjelma. Keskiössä on ollut toimiva käyttöliittymä sekä tietoturvallinen tallennus ulkoiseen tietokantaan. Sovellus on toteutettu osana Ohjelmistotuotantoprojekti 1 & Ohjelmistotuotantoprojekti 2 -kursseja. Toteutuksessa on hyödynnetty kursseilla läpikäytyjä asioita ja koodi on toteutettu Java-kielellä. Käytännön toteutuksessa on seurattu ketterää Scrum-metodologiaa ja suunnittelun apuna on käytetty tiimityöskentelyn apuohjelmaa Nektion-sovellusta.</p>	
Avainsanat	OTP1, OTP2, Projekti, Muistiinpanosovellus, Java

Sisällys

Sisällys	1
1 Johdanto	3
2 Ominaisuudet	4
2.1 Toteutuneet ominaisuudet	4
2.2 Jatkokehitys	6
3 Käyttöönotto	8
4 Sovelluksen käyttöohje	9
4.1 Kirjautuminen ja rekisteröityminen	9
4.2 Muistiinpanojen luominen ja päänäkyvä	11
4.3 Muistiinpanojen arkistointi ja poistaminen	12
4.4 Tekstinmuotoilu ja otsikko	12
4.5 Asetukset	13
5 Arkkitehtuuri	15
5.1 Yleiskuva arkkitehtuurista	15
5.2 Ohjain	17
5.3 Malli	18
5.4 Näkymä	20
6 Back end	23
6.1 Käytetyt teknologiat	23
6.2 Tietokanta	23
6.3 Yleiskuva arkkitehtuurista	23
6.4 Filter-luokat	24
7 Testaus	26
7.1 Sovelluksen testit	26

7.2	Testien automatisoiminen	27
7.3	Graafisen käyttöliittymän testaus	28
8	Kehittäjää kiinnostavaa	29
9	Yhteenveto	30
10	Linkkejä	31

1 Johdanto

Tässä dokumentissa kerrotaan tekemämme muistiinpanosovelluksen ominaisuuksista ja käydään läpi ohjelman käyttöohjeet, sen arkkitehtuuri (mukaan lukien back end), sekä testaus.

Projektin tarkoituksena oli luoda helppokäyttöinen muistiinpanosovellus yleiskäyttöön. Sovellus eroaa muista markkinoilla olevista muistiinpanosovelluksista sillä, että se on ilmainen, ei vaadi sähköpostin antamista rekisteröitymisessä ja sisältää piirto-ominaisuuden. Toisena tavoitteena oli myös luoda kevyempi versio Word -ohjelmasta, joka on helppo- ja nopeakäyttöisempi.

Tällä hetkellä sovelluksessa on mahdollista kirjoittaa muistiinpanoja ja tallentaa niitä pilveen sekä paikallisesti. Mikäli sovellukseen on kirjaututtu sisään, muistiinpanot tallennetaan pilveen kyseiselle käyttäjälle automaattisesti ilman erillistä tallenna-painikkeen napsautusta. Kirjautumatta käyttäjä pystyy luomaan muistiinpanoja, mutta automaattinen pilveen tallentaminen ei ole mahdollista vaan muistiinpanot tallentuvat paikallisesti käyttäjän kovalevylle.

2 Ominaisuudet

Tässä luvussa kerrotaan sovelluksen ominaisuuksista, toteutuneista ja jatkokehityksen puolelle jääneistä. Suurin osa alun perin suunnitelluista ominaisuuksista saatiin kokoon. Keskityimme toteuttamaan tärkeimmät ominaisuudet ensin ja niiden jälkeen seuraavaksi tärkeimmät. Tavoitteena oli saada perustoiminnallisuus kuntoon, jotta sovelluksesta saataisiin toimiva versio, joka toimisi luotettavasti jokapäiväisessä käytössä.

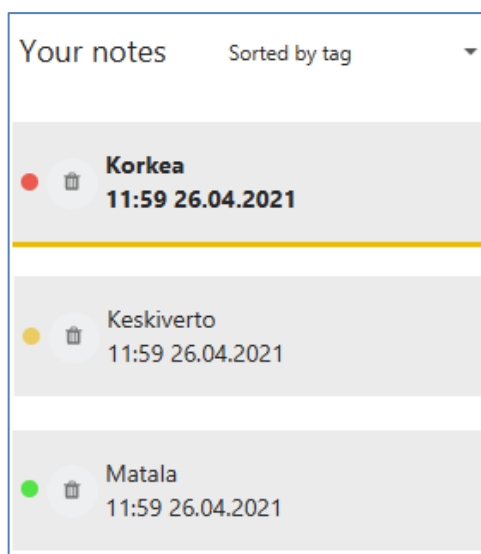
2.1 Toteutuneet ominaisuudet

Sovelluksen pääominaisuutena on muistiinpanojen kirjoittaminen ja tallentaminen. Käyttäjä voi rekisteröidä oman käyttäjäprofiilinsa, johon käyttäjä voi kirjautua sisään aina sovellusta käyttäessään. Internetyhteyden puuttuessa käyttäjä voi myös jatkaa sovelluksen käyttöä kirjautumatta. Käyttäjän omat muistiinpanot näkyvät listassa päänäkömman vasemmassa reunassa, mistä ne ovat helposti valittavissa tarkasteltaviksi ja muokattaviksi. Jokaisella käyttäjällä on omat muistiinpanonsa, eikä muiden käyttäjien muistiinpanoja voi nähdä. Muistiinpanot on salattu sotilaallisen luokan salauskryptografialla, joten ulkopuoliset tahot eivät pääse muistiinpanojen sisältöihin käsiksi.

Jos käyttäjä on kirjautuneena sisään, muistiinpanoihin tehdyt muutokset tallentuvat automaattisesti aina tietyn ajan kuluttua, kun uusia muokkauksia ei ole enää sinä aikana tehty. Muistiinpanoja on mahdollista myös tallentaa lokaalisti Save As-painikkeella. Save As-painike on myös ainoa tapa tallentaa muistiinpano, jos käyttäjä ei ole kirjautunut sisään.

Muistiinpanon tekstikentässä olevaa tekstiä pystyy muotoilemaan tekstikentän vasemmassa alakulmassa löytyvän työkalupalkin avulla. Tekstiä pystyy lihavoimaan, kursivoimaan ja alleviivaamaan. Lisäksi tekstin kokoa ja fonttityyppiä voi muuttaa. Tekstin muotoilu auttaa käyttäjää korostamaan muistiinpanoistaan oleellisia asioita esimerkiksi lihavoimalla tärkeitä kohtia.

Muistiinpanoja voi lajitella tiettyjen kriteerien perusteella. Kun muistiinpanoja on suuri määrä, tämä auttaa niiden jäsentelyssä ja löytämisessä. Muistiinpanoja voi lajitella nimen perusteella, päivämäärän perusteella (viimeisimmän muokkauksen päiväys näkyy muistiinpano listassa nimen alapuolella) sekä muistiinpanoon liitetyn prioriteetin mukaan. Prioriteetti voi olla *korkea*, *keskiverto* tai *matala*. Prioriteetti näkyy muistiinpanolistassa muistiinpanon nimen vasemmalla puolella värillisenä pallona. Punainen merkitsee korkeaa prioriteettia, keltainen keskiverto prioriteettia ja vihreä matalaa prioriteettia. Muistiinpanon päiväys ja prioriteetti on demonstroitu kuvassa 1.



Kuva 1. Muistiinpanoissa viimeisimmän muokkauksen päiväys näkyy nimen alapuolella ja prioriteetti muistiinpanon vasemmassa reunassa.

Kuvassa näkyvä päiväys päivittyy automaattisesti viimeisimpään kellonaikaan, jolloin muistiinpanoa on muokattu. Kuvassa 1 näkyy myös jokaisen muistiinpanon kohdalla roskakoripainike. Tätä painiketta napsauttamalla voi poistaa kyseisen muistiinpanon. Poistetut muistiinpanot katoavat lopullisesti: niitä ei voi palauttaa.

Sovelluksessa pystyy myös arkistomaan muistiinpanoja, jolloin ne eivät näy päänäytössä. Arkistointi on käytössä vain, jos käyttäjä on kirjautunut sisään. Arkistoidut muistiinpanot eivät kuitenkaan poistu tai katoa, vaan ne varastoidaan toiseen paikkaan pilven tietokannassa. Arkistointi on hyödyllistä esimerkiksi, jos käyttäjällä ei ole tarvetta tietyn muistiinpanon tarkastelemiseen lähiaikoina, mutta joskus myöhemmin sellainen tarve

saattaa ilmetä. Arkistoituja muistiinpanoja voi tuoda takaisin päänäkökymän listaan arkistointinäköymästä. Arkistoituja muistiinpanoja voi myös poistaa arkistosta lopullisesti, jos niitä ei enää tarvitse.

Kun käyttäjä on kirjautuneena, kaikki muistiinpanot salataan ja pakataan tallennettaessa. Tietokannassa näkyy vain salattu muistiinpano, mikä parantaa yksityisyydensuojaa ja turvallisuutta. Muistiinpanojen pakkaaminen vähentää tallennustilan käyttöä ja tekee muistiinpanoista kevyitä. Myös käyttäjien salasanat ovat salattuja.

Sovelluksen teemaa ja värimaailmaa voi vaihtaa. Teemoja on kaksi vaihtoehtoa: vaalea ja tumma. Laajennetun värimaailman vaihtoehtoja on neljä: punainen, keltainen, vihreä ja sininen. Sovelluksessa on myös jonkin verran animaatioita.

Myös sovelluksen käyttäjäystävällisyyteen on panostettu. Esimerkiksi ominaisuudet, jotka eivät ole käytettävissä, on poistettu käyttöliittymässä käytöstä siihen asti, kun ne ovat taas käytettävissä. Muistiinpanojen listasta näkee hyvin mikä listan elementti on aktivoitu. Painike, jolla luodaan uusia muistiinpanoja, on iso ja selkeästi plusmerkillä varustettu. Tällaisia käytettävyyssominaisuuksia on useita.

2.2 Jatkokehitys

Ensisijainen jatkokehitykseen jäävä toiminnallisuus on piirto-ominaisuus, jolla muistiinpanoon voi lisätä suoraan itse tekemiään piirroksia. Piirrokset auttavat kokonaisuuksien hahmottamista ja voivat vaikka selkeyttää koodipohjaa. Piirto-ominaisuus voisi mahdollisesti toimia niin, että käyttäjä voi piirtää muistiinpanon päälle erilliselle tasolle (layer) piirtotyökalulla, ja muistiinpanoon tallentuu sekä tekstitaso että piirrostaso.

Suojattu yhteys back end-palvelimelle on myös jatkokehityskohteenä. Sertifikaattien ja Javan välillä olevat ongelmat tulisi saada ratkottua, jolloin sertifikaatin kanssa olisi taattu https-yhteys palvelimelle. Tällä hetkellä palvelinyhteys on http-yhteys, joka ei ole suojattu. Muistiinpanojen tiedot ovat kuitenkin suojatut, sillä ne salataan ennen niiden lähettämistä yhteyden kautta palvelimelle.

Suunnitelmissa olisi myös ollut webikäyttöliittymä sovellukselle, jolloin käyttäjät olisivat päässeet käsiksi omiin muistiinpanoihinsa verkkoselaimen kautta. Täten muistiinpanoihin pääsisi käsiksi myös puhelimilla ja niitä voisi kirjoittaa ja muokata käytännössä missä vain ja milloin vain. Tällä hetkellä ulkoiseen tietokantaan ja muistiinpanoihin pääsee käsiksi vain työpöytäsovelluksen kautta.

3 Käyttöönotto

Tässä luvussa annetaan lyhyt ohjeistus siitä, miten projektin saa käyttöön.

Sovellus vaatii Apache-Maven asennuksen, jos ohjelma ei ole asennettu voi seurata ohjeita valmistajan sivuilta <https://maven.apache.org/install.html>

Sovelluksen voi asentaa hakemalla projektin repositoriosta (<https://gitlab.metropolia.fi/nicoja/ohjelmistotuotantoprojekti-1>) seuraavilla komennoilla.

1. Ensin siirry kansioon, johon haluat projektin sijoittaa `cd` komennolla.
2. Seuraavaksi lataa projekti omalle laitteelle käyttämällä `git clone` komentoa

```
git clone https://gitlab.metropolia.fi/nicoja/ohjelmistotuotantoprojekti-1.git
```

3. Siirry ladattuun kansioon

```
cd ohjelmistotuotantoprojekti-1
```

4. Käynnistä sovellus käyttämällä `maven javafx:run` komentoa

```
mvn javafx:run
```

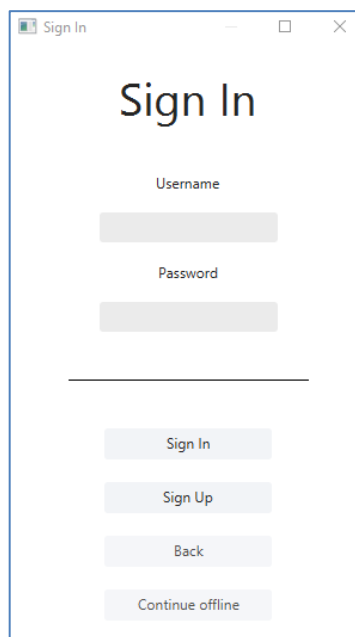
Sovelluksen back end vaatii myös Metropolian VPN-etäyhteyden, jonka asennus- ja käyttöohjeet löytyvät osoitteesta <https://wiki.metropolia.fi/pages/viewpage.action?pageId=149652071> .

4 Sovelluksen käyttöohje

Tässä luvussa käsitellään sovelluksen käyttöohjeet. Käydään läpi kirjautuminen, rekisteröityminen, päänäkymän käyttö, muistiinpanojen luonti, tekstinmuotoilun käyttö ja asetusten muokkaaminen.

4.1 Kirjautuminen ja rekisteröityminen

Sovelluksen käynnistyttyä tulee käyttäjän eteen kirjautumislomake, jossa käyttäjä voi kirjautua sisään, rekisteröitymään tai jatkamaan sovelluksen käyttöä kirjautumatta. Kirjautumislomake näkyy kuvassa 2.

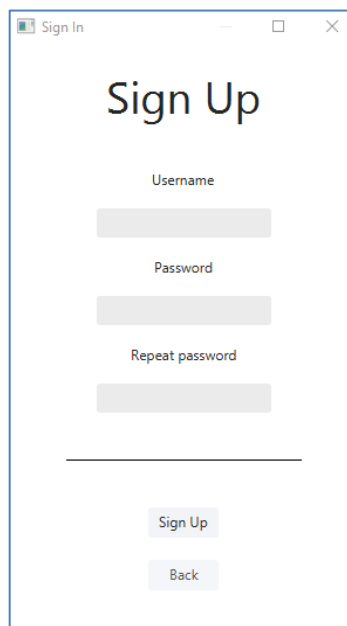


Kuva 2. Kirjautumislomakeessa pystyy kirjautumaan, rekisteröitymään tai jatkamaan kirjautumatta.

Napsauttamalla *Continue offline*-painiketta, joka on kirjautumislomakkeen alin painike, voi sovellusta käyttää kirjautumatta. Kirjautumatta monet ominaisuudet, kuten automaattinen tallennus ja arkistointi eivät ole käytössä. Kirjautumisessa käyttäjän on kirjoitettava

käyttäjänimensä *Username*-kenttään ja salasansansa *Password*-kenttään ja sitten napsautettava *Sign in*-painiketta.

Käyttäjän käyttäessä sovellusta ensimmäistä kertaa käyttäjällä ei ole käyttäjätunnusta. Halutessaan automaattisen tallennusominaisuuden ja arkistoinnin käyttöön, täytyy käyttäjän luoda käyttäjätunnus, jonka pystyy luomaan napsauttamalla kirjautumislomakkeessa *Sign Up*-painiketta (ks. Kuva 2). Kuvassa 3 on näytetty rekisteröitymislomake.

The image shows a web browser window with the title 'Sign In'. The main heading of the page is 'Sign Up'. Below the heading, there are three input fields: 'Username', 'Password', and 'Repeat password'. Each field has a light gray placeholder bar. Below the input fields, there is a horizontal line. Under the line, there are two buttons: 'Sign Up' and 'Back'. The 'Sign Up' button is highlighted with a light blue background.

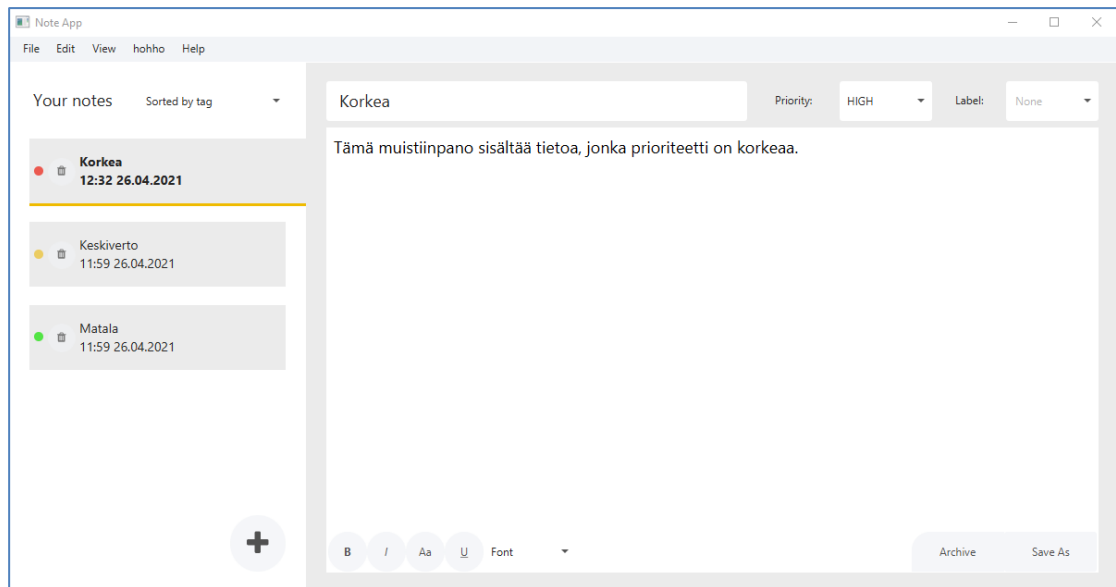
Kuva 3. Käyttäjänluomislomakkeessa käyttäjän täytyy antaa haluttu käyttäjätunnuksensa ja salasana kahteen kertaan.

Rekisteröitymislomakkeessa on kaksi salasanakenttää, joissa olevien salasanojen pitää täsmätä, jotta rekisteröityminen onnistuu. Käyttäjänimi ei saa olla käytössä kenelläkään muulla ja sen pitää olla vähintään yhden merkin pituinen.

Rekisteröityessä käyttäjän on kirjoitettava käyttäjänimensä *Username*-kenttään ja salasansansa *Password*-kenttään sekä *Repeat password*-kenttään ja sitten painaa *Sign Up*-painiketta. Rekisteröitymisen onnistuessa käyttäjä kirjataan automaattisesti sisälle sovellukseen.

4.2 Muistiinpanojen luominen ja päänäkö

Kun käyttäjä on siirtynyt kirjautumislomakkeesta eteenpäin päänäköön, voi käyttäjä luoda uuden muistiinpanon painamalla vasemmassa alareunassa näkyvää isoa + -painiketta (Kuva 4).



Kuva 4. Sovelluksen päänäkö. Sovelluksessa on avattuna vasemmasta listasta ensimmäinen muistiinpano.

Jos käyttäjä on kirjautuneena, muistiinpanot tallentuvat automaattisesti, joten käyttäjän ei tarvitse itse huolehtia muistiinpanojen tallentamisesta. Kirjautumatta käyttäjän on tallennettava itse muistiinpanonsa painamalla *Save As*-painiketta muistiinpanon oikeasta alareunasta tai *File*-valikon alta. *File*-valikko löytyy päänäkömön vasemmasta yläkulmasta (ks. Kuva 4).

Uudelle muistiinpanolle voi asettaa otsikon päänäkömässä näkyvän ison tekstikentän yläpuolella näkyvään tekstikenttään. Muistiinpanon sisältöä voi kirjoittaa isoon tekstikenttään, kuten kuvassa 4 demonstroidaan. Vasemmassa listassa olevat muistiinpanot hetkellisesti poistuvat käytöstä tallentamisen ajaksi. Muistiinpanon tallentuessa vasen lista aktivoituu uudestaan käyttöön.

Muistiinpanolle voi asettaa prioriteetin *Priority*: -kentässä, jossa on kolme prioriteetti tasoa; korkea, keskiverto ja matala. Prioriteetti näkyy kuvassa 4 vasemmalla olevassa listassa muistiinpanossa värillisenä pallona. Muistiinpanolle voi myös asettaa nimikkeen *Label*: -kentässä, jossa on muutama valmiiksi luotu nimike. Käyttäjä voi lisätä nimikkeitä itse kirjoittamalla *Label*: -kenttään.

Muistiinpanon voi tallentaa paikallisesti tekstitiedostoksi, painamalla päänäköymän oikeassa alareunassa olevaa *Save As*-painiketta, joka avaa tiedostojenhallinnan, missä voi valita tiedoston sijainnin ja nimen. Sama toiminnallisuus löytyy *File*-valikon alta.

4.3 Muistiinpanojen arkistointi ja poistaminen

Käyttäjän ollessa kirjautuneena, muistiinpanoja voi arkistoida napsauttamalla päänäköymän oikeassa alareunassa olevaa *Archive*-painiketta (ks. Kuva 4). Arkistoitujen muistiinpanojen listan saa tarkasteltavaksi *View*-valikon vaihtoehdosta *Change view -> Archive*. Kun käytössä on arkistonäkymä, oikealla alareunassa oleva *Archive*-painike muuttuu *Unarchive*-painikkeeksi ja sitä napsauttamalla muistiinpanon voi palauttaa päänäköymän listaukseen.

Muistiinpanoja voi poistaa napsauttamalla muistiinpanon nimen vasemmalla puolella olevaa roskakori painiketta tai *File*-valikosta *Delete*-painiketta (*File*-valikon alla oleva *Delete*-painike poistaa sillä hetkellä valittuna olevan muistiinpanon). Poistettuja muistiinpanoja ei pysty palauttamaan, joten sovellus pyytää aina vielä käyttäjältä varmistuksen muistiinpanon poistosta.

4.4 Tekstinmuotoilu ja otsikko

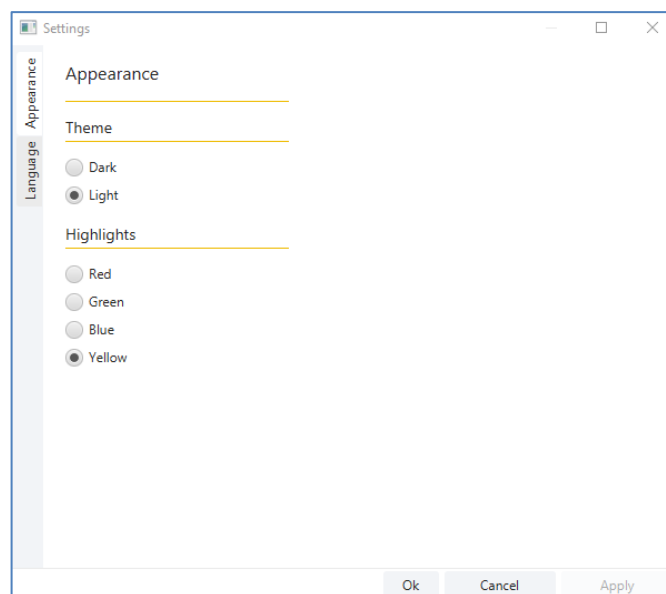
Päänäköymän ison tekstikentän vasemmassa alakulmassa on tekstinmuokkaustyökalupalkki (Kuva 4). Muokausvaihtoehtoina on tekstin lihavointi, kursivointi, alleviivaus, koon muutos, ja itse fontin muuttaminen. Lihavointi muuttaa maalatun tekstin paksuksi, kursivointi muuttaa maalatun tekstin vinoksi, alleviivaus alleviivaa maalatun tekstin ja koon muutoksella vaihdetaan tekstin kokoa.

Tekstin kokoja on vain kolme vaihtoehtoa; iso, keskikokoinen ja pieni. Fontin koon muuttaminen toimii maalaamalla kyseisen alueen, jonka fonttikokoa haluaa vaihtaa ja napsauttaa *Aa*-painiketta. Tämä muuttaa fontin isommaksi. Toinen napsautus muuttaa koon pieneksi ja kolmas napsautus palauttaa koon normaaliksi (keskikokoiseksi). Fontteja vaihdetaan valitsemalla *Font*-alavalikosta haluama fontti. Fontin pystyy vaihtamaan miin tahansa fonttiin, mikä löytyy asennettuna käyttäjän tietokoneelta.

Muistiinpanojen otsikoiden nimen pituus on rajoitettu 32 merkkiin, jotta nimet eivät olisi liian tiukasti pakattuna päänäköymän vasemman reunan listassa. Listan leveyttä pystyy muuttamaan ottamalla kiinni listan ja ison tekstikentän välisestä leikkauskohdasta ja liikkuttamalla hiirtä oikealle tai vasemmalle.

4.5 Asetukset

Käyttöliittymän kieltä sekä ulkonäköasetuksia on mahdollista muuttaa asetusvalikosta, joka löytyy päänäköymässä *File*-valikon alta *Settings*-painikkeella. Menemällä asetusvalikkoon aukeaa uusi ikkuna (Kuva 5).



Kuva 5. Asetusvalikko, jossa on kaksi välilehteä; yksi teeman sekä korostusvärien vaihtamiseen ja toinen kielen vaihtamiseen.

Kuvasta 5 näkyy, että asetusvalikossa on kaksi välilehteä; *Appearance* ja *Language*. *Appearance*-välilehdestä käyttäjä pystyy vaihtamaan sovellukseen teemaa sekä korostusväriä painamalla haluamansa värin radiopainiketta. Värin tai teeman valitseminen muuttaa sovelluksen värimaailmaa välittömästi, joten asetusvalikossa huomaa heti, miltä muutos näyttää. Käyttäjän ollessa tyytyväinen asetuksiin, käyttäjä voi painaa *Ok* -painiketta, jolloin asetukset tallennetaan ja asetusvalikko sulkeutuu. Painamalla *Cancel*-painiketta, asetukset palautuvat niihin asetuksiin, jotka olivat käytössä ennen asetusvalikon avaamista, jonka jälkeen asetusvalikko sulkeutuu. Painamalla *Apply*-painiketta, asetukset tallentuvat, mutta asetusvalikko ei sulkeudu. *Apply*-painiketta painamalla käyttäjä voi esimerkiksi tarkastella väriasetuksien jälkeen kieliasetuksia.

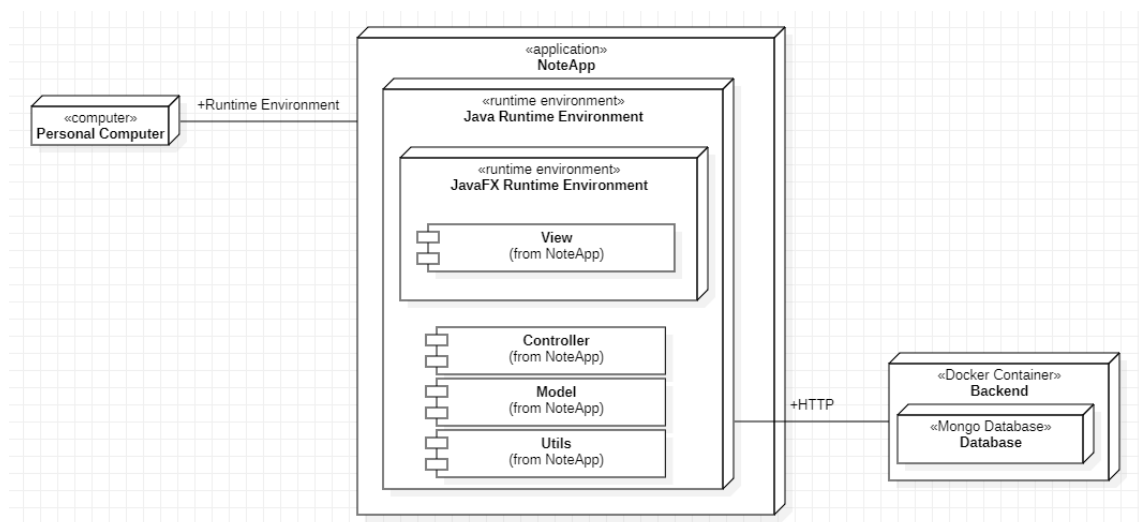
Kieliasetukset löytyvät asetusvalikon *Language* -välilehteä napsauttamalla. Kielivaihtoehtoina ovat Suomi, Englanti, Venäjä sekä Espanja. Kieltä vaihtamalla koko sovelluksen käyttöliittymän kieli vaihtuu välittömästi valittuun kieleen. Näin käyttäjä näkee heti, miltä käyttöliittymä näyttää uudella kielellä. Tapauksissa, joissa käyttäjä ei esimerkiksi osaa-kaan kyseistä kieltä, voi käyttäjä napsauttaa asetusvalikon *X*-painiketta, jolloin kaikki asetukset palautuvat ennalleen, ellei *Apply* -painiketta ole napsautettu sitä ennen.

5 Arkkitehtuuri

Tässä luvussa käsitellään sovelluksen arkkitehtuuria ja kerrotaan miten kehittämisessä hyödynnettiin MVC-suunnittelumallia sekä kuvataan sovelluksen luokkahierarkioita.

5.1 Yleiskuva arkkitehtuurista

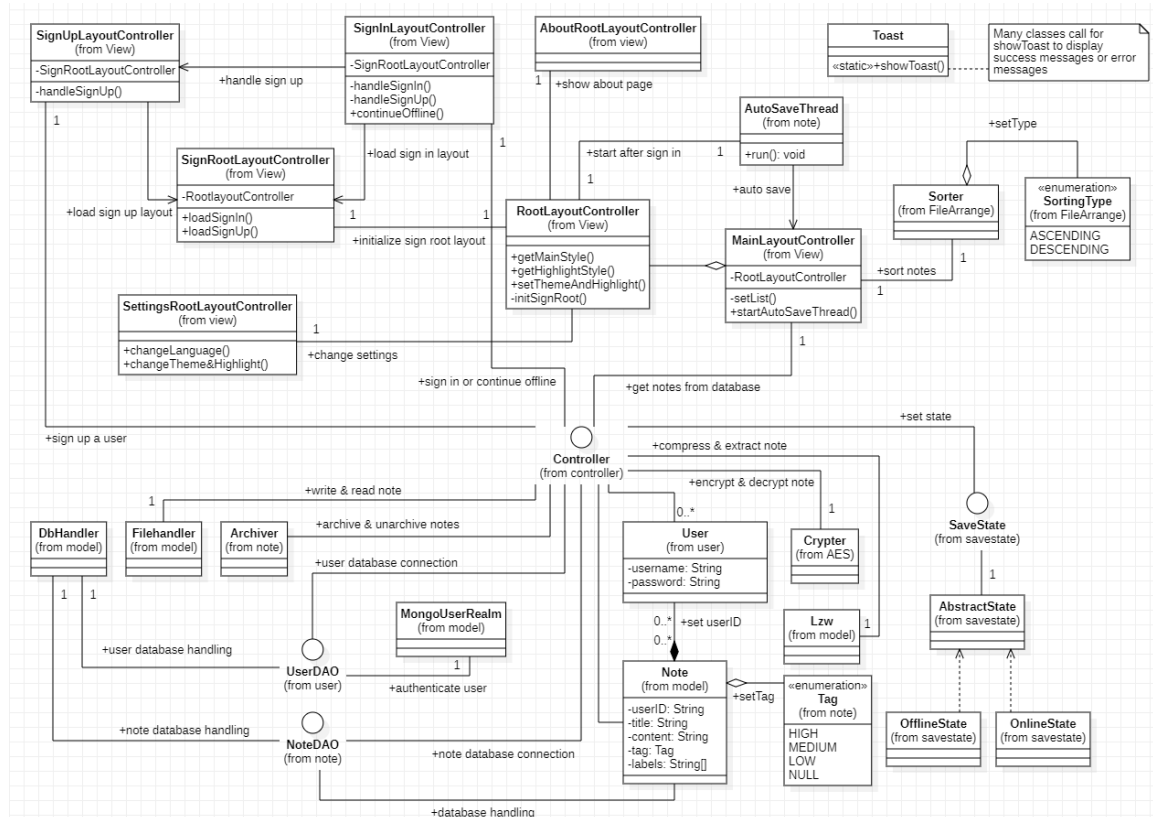
Sovelluksen kehittämisessä on hyödynnetty MVC-suunnittelumallia (Model, View, Controller), jonka tarkoituksena on irrottaa sovelluksen logiikka ja käyttöliittymä erillisiksi kokonaisuuksiksi. Keskustelu osien välillä hoidetaan ohjainluokkaa hyödyntäen. Yleiskuvaus sovelluksen arkkitehtuurista sisältää kolme osapuolta; käyttäjä, itse ohjelma sekä palvelin, jossa sijaitsee tietokanta (Kuva 6).



Kuva 6. Arkkitehtuurin yleiskuva. Näkymän tuottaa käyttäjälle JavaFX ja logiikan hoitaa Controller-, Model- sekä Utils-pakkausten sisältö. Sovellus on myös yhteydessä palvelimeen.

Kuvasta 6 näkee sovelluksen pääasiallisen arkkitehtuurin, jossa käyttäjä on yhteydessä sovellukseen Javan virtuaalikoneen kautta, joka pyörittää sovellusta. Sovelluksen graafisen käyttöliittymän tuottaa JavaFX. Sovellus on yhteydessä ulkopuoliseen palvelimeen, jossa sijaitsee sovelluksen tietokanta. Tietokantaan tallennetaan käyttäjät sekä muistiinpanot.

Sovelluksen sisällä kaikki tieto liikkuu ohjaimen eli Controller-luokan välityksellä. Käyttäjän tehdessä valintoja käyttöliittymässä, tieto tästä lähetetään Controller-luokalle, joka taas osaltaan kutsuu mallista oikeanlaista luokkaa, jonka tehtäväksi jää päivittää näkymä käyttäjälle. Käyttöliittymä ei siis tee itse näkymän päivittämiseen tarvittavia laskutoimituksia. Kuvassa 7 näkyy, kuinka Controller-luokka toimii mallin ja käyttöliittymän välissä ohjaimena ja toimittaa tietoa toiselta toiselle.

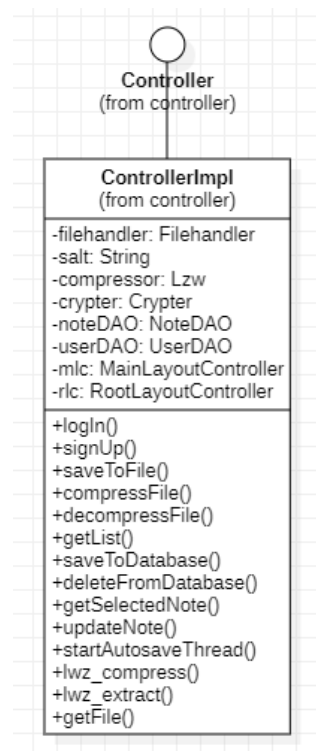


Kuva 7. Luokkakaavio sovelluksen rakenteesta. Controller-luokka välittää kaiken tiedon Mallin ja Näkymän välillä

Kuvassa 7 on esitelty sovelluksen pää rakenne. Keskellä kuvassa on Controller-olio (toeuttaa Controller-rajapinnan), joka toimii linkkinä sovelluksen eri osien välillä. Malli huolehtii esimerkiksi käyttäjästä, muistiinpanoista, sovelluksen tilasta, salauksesta, kompressoinnista ja monesta muusta asiasta. Näkymä huolehtii siitä, että nämä Mallin antamat toiminnallisuudet ovat käyttäjälle käsillä. Mallin ja näkymän osat on pilkottu palikoiksi, jotka toimivat yhdessä yhtenä kokonaisuutena.

5.2 Ohjain

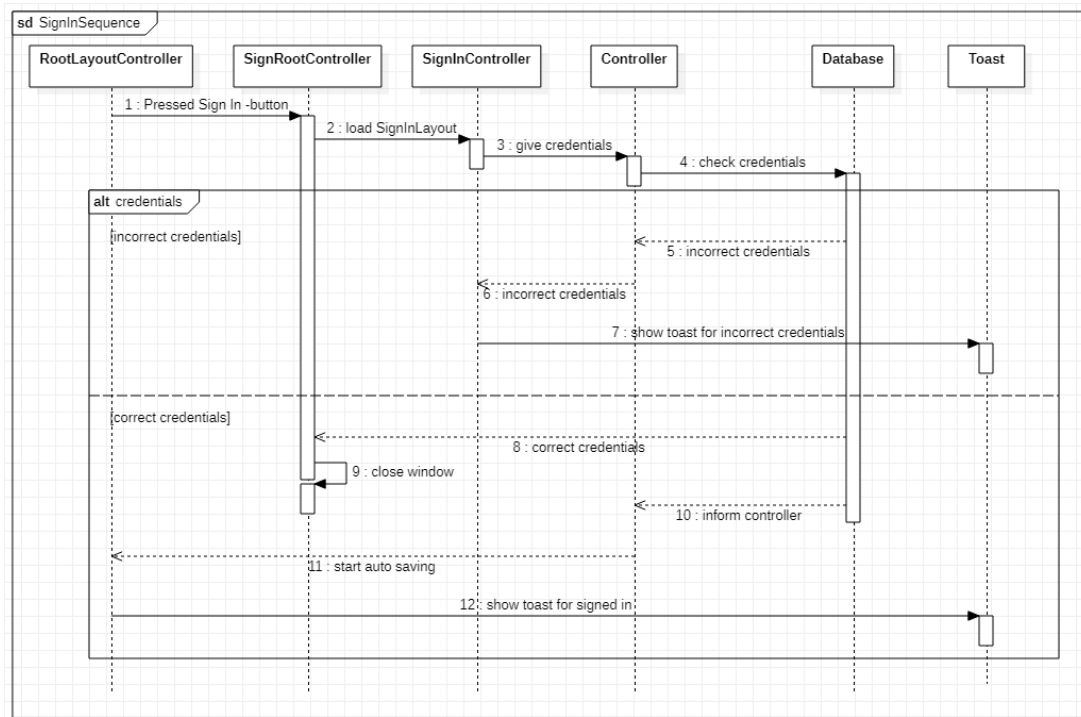
Ohjain eli Controller-luokka huolehtii mallin ja näkymän välisestä kommunikoinnista. Controller-luokan toiminta vaatii paljon toiminnallisuuden takaavia operaatioita, joilla käyttäjän tekemät muutokset toteutuvat käyttöliittymässä. Controller-luokka pitääkin sisällään oleellimmat sovelluksen toiminnassa tarvittavista metodeista (Kuva 8).



Kuva 8. Controller-luokan luokkakaavio. Luokka sisältää paljon tietoa ja metodeja, joiden avulla käyttöliittymässä tehdyt toiminnot välittyvät Mallille käsiteltäviksi, joka taas päivittää käyttöliittymää käsittelyjen mukaisesti.

Kuvasta 8 selviää Controller-luokan metodit ja luokkamuuttujat. Controller-luokka huolehtii siitä, että käyttöliittymässä tehdyt muutokset tai painikkeiden napsautukset välittyvät mallille, joka puolestaan prosessoi muutokset tai painikkeiden toiminnallisuuden.

Kuvassa 9 on demonstroitu kirjautumisprosessi, jossa näkyy kummatkin lopputulokset kirjautumisessa; kirjautuminen ja kirjautumisen epäonnistuminen.

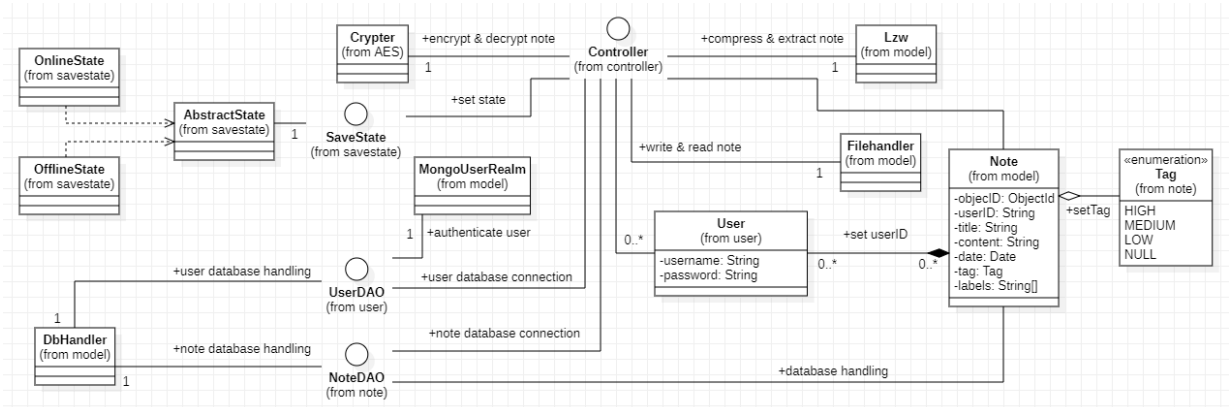


Kuva 9. Sekvenssikaavio kirjautumisprosessista.

Controller-luokka huolehtii esimerkiksi kirjautumisprosessista (kuva 9), kun käyttäjä on kirjoittanut käyttäjänimensä ja salasananansa kirjautumislomakkeeseen. Kirjautumistiedot välittyvät Controller-luokan kautta mallille, joka osaa lähettää kirjautumistiedot back end-palvelimelle (jätetty pois sekvenssikaaviosta prosessin selkeyttämiseksi). Back end-palvelin reagoi kirjautumistietoihin tarkastamalla, löytyykö tietokannasta kyseisiä tietoja, ja palauttaa vastauksen siitä, että löytyykö tietoja vai ei. Malli ottaa vastaan nämä tiedot palvelimelta ja jos tiedot löytyivät, käyttäjän tiedot ja muistiinpanot haetaan sovellukseen (kuvassa 9 kohta 10) ja jos tietoja ei löydy, ilmoittaa malli virheestä, joka näkyy käyttöliittymässä ilmoituksena. Toast-luokka ilmoittaa virheestä tai onnistumisesta käyttäjälle visuaalisesti käyttöliittymässä.

5.3 Malli

Malli-pakkaus sisältää sovelluksen sisäisen logiikan. Kuvassa 10 on esitetty Malli-pakauksen sisältämät luokat ja rajapinnat.



Kuva 10. Sovelluksen Malli-osion luokkakaavio.

Mallissa on määritelty käyttäjien sekä muistiinpanojen hallintaan keskittyvät luokat kuten myös tietokantayhteydessä hyödynnettävät luokat. Muistiinpanojen luomisessa, salauksessa sekä pakkauksessa käytetyt luokat löytyvät myös Malli-pakkauksesta.

Muistiinpanot ja käyttäjätiedot pidetään tallessa tietokannassa. Datan siirtäminen sovelluksen ja tietokannan välillä tapahtuu HTTP-kutsujen avulla. Kutsut ohjataan back end -palvelimelle, joka suorittaa varsinaisen tietokantaan tallentamisen tai sieltä hakemisen.

Käyttäjän lisääminen tapahtuu lähettämällä POST-pyyntö päätepisteeseen `/api/user`. Pyyntöön rungossa tulee olla käyttäjänimi sekä salasana. Onnistunut pyyntö palauttaa käyttäjänimen sekä käyttäjätunnisteen. Käyttäjän poisto tehdään lähettämällä DELETE-pyyntö päätepisteeseen `/api/delete/user/id`, jossa id on poistettavan käyttäjän tunniste. Pyyntöön ylätunnisteessa tulee olla Bearer token, joka vastaa käyttäjän tokenia.

Muistiinpanon lisääminen toimii lähettämällä POST-pyyntö päätepisteeseen `/api/notes/save/id`, jossa id on käyttäjän tunniste. Pyyntöön ylätunnisteen autentikointiosiossa tulee olla käyttäjän token muodossa *Bearer token*, jossa token on käyttäjän JWT-token. Muistiinpano lähetetään JSON-muodossa. Onnistunut pyyntö palauttaa juuri luodun muistiinpanon.

Muistiinpanojen hakeminen tapahtuu muuten samoin kuin lisääminen, mutta päätepisteinä on `/api/notes/id`, jossa id on käyttäjän tunniste, ja pyyntöön tyyppinä tulee olla GET. Onnistunut pyyntö palauttaa listan muistiinpanoja JSON-muodossa.

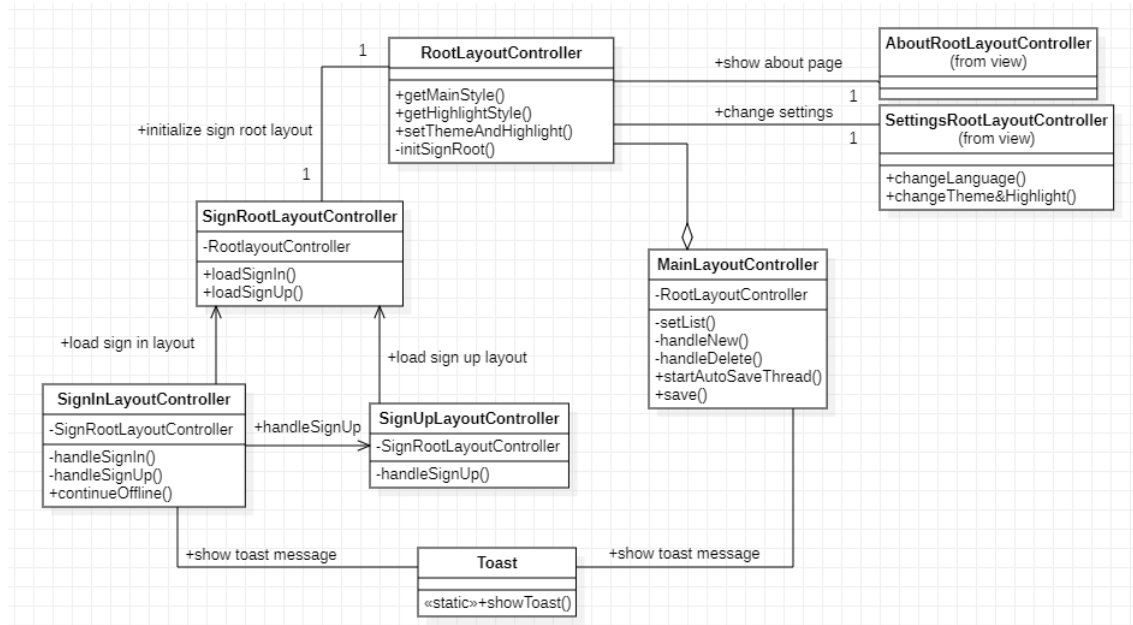
Muistiinpanojen poistamisessa suoritetaan DELETE-pyyntö päätepisteeseen `/api/notes/delete/käyttäjän_id/muistiinpanon_id`. Pyyntöön ylätunnisteessa tulee olla oikea token muodossa *Bearer token*, jossa token on käyttäjän JWT-token. Pyyntö palauttaa `true` mikäli poistaminen on onnistunut.

Ennen muistiinpanojen ja käyttäjien lähettämistä tietokantaan, muistiinpanojen sisältö sekä käyttäjätiedot pakataan ja salataan. Paketointi- ja salausalgoritmit ovat keskeisiä turvallisuuden ja nopeuden takaajia sovelluksessamme. Käyttäjien salasanat salataan AES 128 -algoritmilla. Algoritmin suolana käytettävä merkkijono luodaan käyttäjän luonnin yhteydessä, ja tallennetaan tietokantaan omalle rivilleen. Pakkausalgoritmina käytämme Lempel-Ziv-Welch (LZW) -algoritmia, joka keventää muistiinpanojen kokoa ja täten nopeuttaa tiedon siirtoa.

Malli-pakkaus huolehtii myös siitä, että käyttöliittymä näyttää oikealta ja tarjoaa oikeat ominaisuudet riippuen siitä, onko käyttäjä kirjautunut sisään vai ei. Malli-pakkauksessa oleva *SaveState*-rajapinta määrittelee sovelluksen tilan. Kuten kuvasta 10 näkyy, *SaveState*-rajapinnalla on kolme ilmentymää: abstrakti luokka *AbstractState* ja sitä toteuttavat konkreettiset luokat *OnlineState* ja *OfflineState*. State-luokissa on määritelty toiminnallisuus, joka näyttäytyy käyttöliittymälle samanlaisena. Sovelluksen tilaa voidaan muuttaa ohjelman ajon aikana tai käynnistyksen yhteydessä. Muutos tapahtuu kirjautumalla sisään. *OnlineState* pitää sisällään toiminnallisuudet, jotka käyttävät pilviominaisuuksia. Esimerkiksi *setList*-metodi hakee muistiinpanot tietokannasta, ja asettaa ne näkymään. *OfflineState:n* vastaava metodi päivittää vain näkymästä löytyvää muistiinpanolistaa.

5.4 Näkymä

Sovelluksen Näkymä-osassa olevat luokat (Kuva 11) huolehtivat käyttöliittymän päivittämisestä.



Kuva 11. Sovelluksen Näkymä-osion luokkakaavio.

Jokaiselle sovelluksen eri näkymälle on luotu erillinen FXML-tiedosto, joka tarvitsee toimiakseen erillisen ohjaimen, jolla saadaan näkymälle haluttu toiminnallisuus. Esimerkiksi painikkeiden toiminnallisuus luodaan kyseisen näkymän ohjaimessa. Kuten kuvasta 11 voi nähdä, näkymä-osan perustana ovat *RootLayoutController*- ja *MainLayoutController*-luokat. Nämä kaksi toistensa kanssa harmoniassa toimivaa luokkaa huolehtivat päänäköymästä ja niin ollen lähes kaikista niistä sovelluksen toiminnallisuuksista, jotka tarjotaan käyttäjälle. *RootLayoutController*-luokka huolehtii päänäköymän yläreunassa olevasta työkalupalkista, jossa on yhdessä paikassa useita ominaisuuksia, kuten eri valikkojen avaaminen, sekä käyttäjän ja muistiinpanojen hallinta. *MainLayoutController*-luokka huolehtii sovelluksen päätoiminnallisuudesta, joka tarjotaan käyttäjälle, eli muistiinpanojen luomisesta ja niiden sisällön muokkaamisesta ja päivittämisestä sekä muistiinpanojen listauksesta näkyviin.

Asetusvalikkoa ohjaava *SettingsRootLayoutController*-luokka löytyy Näkymä-osiosta ja hallitsee sovelluksen ulkonäköä sekä kieliasetuksia. Kaikki sovelluksessa olevat tekstit, jotka näkyvät käyttäjälle, haetaan sovelluksen resurssitiedostosta. Resurssissa sijaitse sovelluksen kaikki kielivaihtoehdot, jotka ovat tällä hetkellä Suomi, Englanti, Venäjä ja

Espanja. Asetusvalikossa kielen vaihdos saa kielenhallintaluokan (*Context*-luokka) kieli-vaihtoehtoon muuttumaan toiseen kieleen. Tämä muutos saa aikaan sen, että kaikki sovelluksen tekstit päivittyvät uuden kielen vastaaviin käännöksiin. Kielen vaihtaminen saa myös päivämäärän formatoonin muuttumaan kieleen sopivaksi. Värimaailman ja teeman vaihto toimivat samalla tavalla ja päivittävät koko sovelluksen välittömästi muuttamalla värit. Sovelluksen käyttämät väriasetukset haetaan CSS-tiedostoista resurssitiedoston sijaan.

Näkymä-osiossa oleva *SignRootLayoutController*-luokka huolehtii siitä, että käyttäjällä on mahdollisuus kirjautua ja rekisteröityä sovellukseen. *SignRootLayoutController*-luokka näkyy kuvassa 11 kiinnitettynä *RootLayoutController*-luokkaan, josta saa avattua kirjautumisenäkymän. *SignRootLayoutController*-luokka lataa oikean luokan näkyviin (*SignInLayoutController* tai *SignUpLayoutController*), joka tarjoaa oikeat toiminnallisuudet käyttäjälle. Kirjautumislomakkeessa (*SignInLayoutController*) käyttäjä päättää haluaako kirjautua vai jatkaa kirjautumatta. Kirjautumislomakkeen valinta päättää *SaveState*-rajapinnan käytettävän toteutuksen, niin kuin yllä kappaleessa 5.3 on selitetty. Kirjautumatta sovellukselle asetetaan uusi tila (*OfflineState*) ja malli ei täten tarjoa kaikki ominaisuuksia (arkistointi ja automaattinen pilveen tallennus) käyttäjälle. Kirjautumisen ohella sovellukselle asetetaan tila (*OnlineState*), joka puolestaan aktivoi kyseiset ominaisuudet.

6 Back end

Tässä luvussa käsitellään sovelluksen back end:in toteutusta sekä tietokantaratkaisua.

6.1 Käytetyt teknologiat

Sovelluksen back end päätettiin rakentaa Java-Spring sovelluskehiksen avulla, sillä Spring sisältää monta laajennusta, jotka ovat suunniteltu web-sovellusten rakentamiseen. Back endin suojaamiseen käytettiin Spring-security -kirjastoa, joka sisältyy Java-Spring -sovelluskehikseen. Kirjautuneen käyttäjän todennus hoituu käyttämällä Java-JWT (*JSON web token*) -kirjastoa.

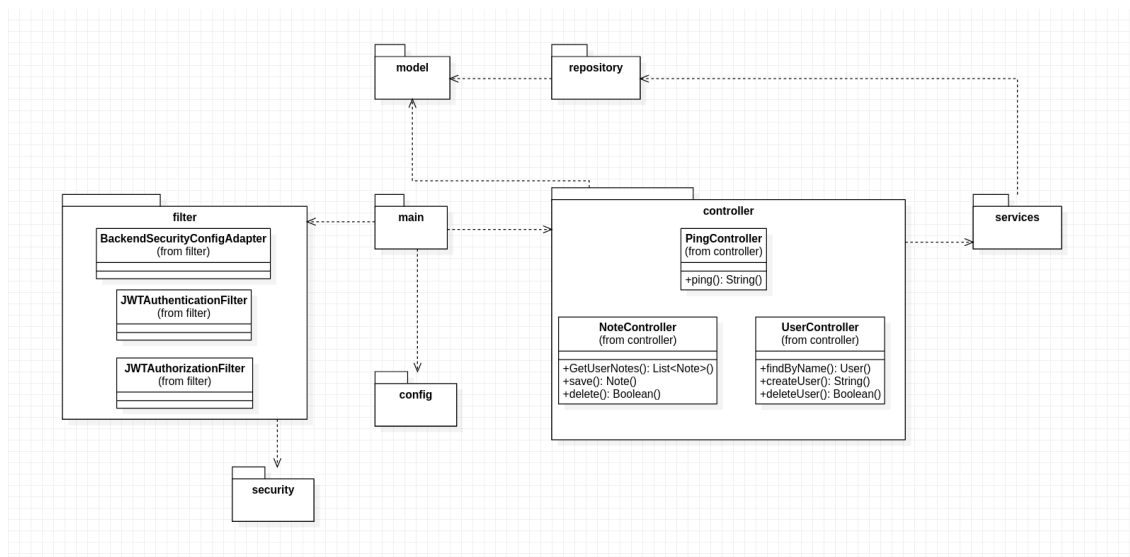
6.2 Tietokanta

Tietokantaratkaisumme perustuu MongoDB-tietokantaan, joka on NoSQL tietokanta. Valitsimme MongoDB-tietokannan sen vuoksi, että voisimme helposti varastoida isoja määriä tekstiä ja tallentaa muistiinpanoja tietokantaan helposti suoraan olioina. Myös nopeus vaikutti tietokantaratkaisun valintaan, sillä MongoDB on nopeampi kuin esimerkiksi MySQL, varsinkin palvelimella käytettynä. Nopeus olikin myös yksi sovelluksemme tärkeä ominaisuus. MongoDB-tietokanta sallii myös hyvät laajennusmahdollisuudet muistiinpanojen ja käyttäjien suhteen, koska tallennus tapahtuu suoraan olioina. Lisäsimme monesti projektin aikana erilaisia ominaisuuksia muistiinpanoihin ja MongoDB:n käyttö oli paljon parempi vaihtoehto tästä syystä.

6.3 Yleiskuva arkkitehtuurista

Sovelluksen back end:in toiminta tapahtuu suurimmaksi osaksi sovelluksen *controller*- ja *filter*-paketeissa (Kuva 12). Controller-paketti sisältää luokat, jotka ottavat vastaan HTTP-pyyntöjä muistiinpanojen ja käyttäjien luomiseen ja poistamiseen. Filter-pakettiin kuuluu sovelluksen tietoturvaan liittyvät luokat, kuten BackendSecurityConfigAdapter.

Sovelluskehysten *main*-luokka ottaa vastaan kaikki sovellukselle tulevat HTTP-pyynnöt, jotka se ohjaa eteenpäin ensin *filter*-luokille, jotka tässä sovelluksessa ovat *JWTAuthenticationFilter* ja *JWTAuthorizationFilter*. Sovelluksen *filter*-luokat hoitavat käyttäjän kirjautumisen ja kirjautuneen käyttäjän valtuutuksen tarkastamisen.



Kuva 12. Back end pakettirakenne

Kuvan 12 pakettikaaviosta näkee kuinka sovelluksen paketit liittyvät toisiinsa. *Main*-luokka keskustelee suoraan *filter*- ja *controller*-pakettien kanssa ja ottaa *config*-paketista MongoDB tietokannan konfiguraation. *Model*-paketti sisältää tietokannan mallit ja *repository*-paketti sisältää tietokantaan yhdistävien luokkien rajapinnat. *Services*-paketti sisältää *repository*-luokkien toteuttavat luokat ja *security*-paketti sisältää tietoturvaan liittyviä vakioita.

6.4 Filter-luokat

Sovelluksen *filter*-luokat ovat back end:in tietoturvan selkäranka. *BackendSecurityConfigAdapter*-luokka määrittelee koko sovelluksen tietoturvan. Tässä luokassa asetetaan Spring-sovelluskehysten uusia *filter*-luokkia, ja määritellään mitkä osoitteet turvataan filterillä, jos turvataan. *BackendSecurityConfigAdapter* määrittelee myös kirjautumiseen

käytettävän AuthenticationManager-luokan käyttäjäpalvelun ja salasana-kryptaus-luokan. JWTAuthenticationFilter vastaa käyttäjän kirjautumisen toteutuksesta, ja BackendSecurityConfigAdapter-luokan luoman AuthenticationManager-luokan kanssa tarkistaa annettujen kirjautumistietojen kelpoisuuden. *Filter* palauttaa asiakkaalle JWT-valtuuden, jota voidaan käyttää kutsujen valtuuttamiseen muualle back end:iin. JWTAuthorizationFilter hoitaa edellä mainitun JWT-valtuuden oikeanmukaisuuden todentamisen, ja antaa käyttäjälle luvat kommunikoida sovelluksen kanssa jos todentaminen onnistuu.

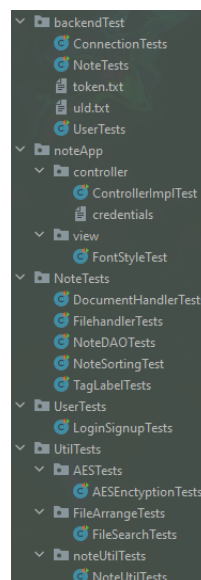
7 Testaus

Tässä luvussa kerrotaan sovelluksen toteutuksen aikaisesta testaamisesta, käytetyistä metodeista ja tuloksista.

Testaamista on tehty JUnit 5 sovelluskehiksen avulla, joka on yhdistetty Jenkins-palvelimeen testien automatisointia varten. Graafista käyttöliittymää on testattu manuaalisesti.

7.1 Sovelluksen testit

Sovelluksen testaamiseksi ja toiminnallisuuden takaamiseksi olemme luoneet jokaiselle mahdolliselle testattavalle luokalle omat testit. Testit tehtiin JUnit 5 sovelluskehiksen kautta, joka on suosituin testausväline Javalle. Loimme projektin alussa aina testit ennen konkreettisten luokkien luomista, jotta saatoimme ohjelmoida luokat testejä vasten, mikä takaa luotujen luokkien toimivuuden. Myöhemmin projektissa kuitenkin prosessi muuttui päinvastaiseksi; loimme ensin luokat, jonka jälkeen testasimme ne testeillä. Kuvassa 13 näkyy kaikki tekemämme testit JUnit 5 sovelluskehiksellä.



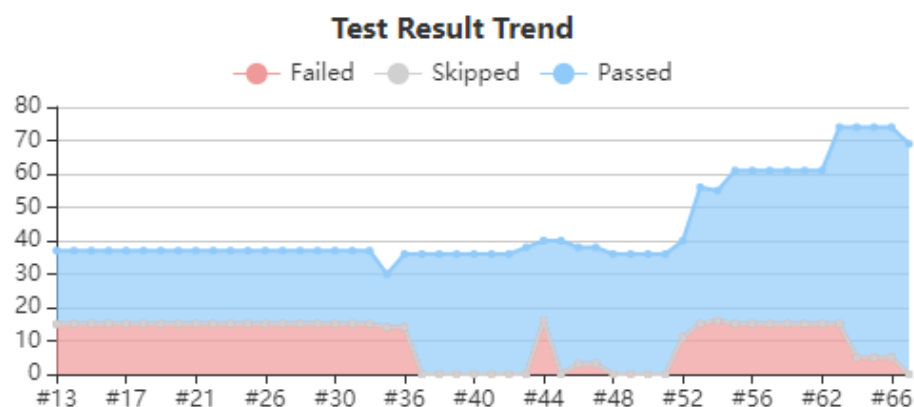
Kuva 13. Sovellukseen tehdyt JUnit 5 -testit.

Kuvasta 13 huomaa, että testejä on tehty eniten *Note*-luokalle, sillä se on sovelluksen tärkein luokka. *Note*-luokka huolehtii kaikesta muistiinpanoihin liittyvästä, kuten muistiinpanojen sisällöstä, päiväyksestä, tärkeydestä, omistajasta sekä nimikkeestä. Kirjautumisen, rekisteröitymisen ja back endin testit ovat myös olleet tärkeässä osassa ja kokonaisvaltaisia. Näin varmistimme, että yhteydet toimivat oikein ja että palvelimen ja sovelluksen välinen kommunikointi toimii kunnolla ja turvallisesti. Myös kaikki sovelluksen algoritmit, kuten salausalgoritmi ja järjestelyalgoritmi, on testattu moneen otteeseen toimiviksi ja luotettaviksi.

7.2 Testien automatisoiminen

Projektin ollessa puolivälissä otimme Jenkins-palvelimen käyttöön, jonka avulla automatisoimme testausprosessia. Tarkastimme Jenkinsin avulla aina satunnaisin väliajoin testiemme kattavuuden ja sen, tarvitsiko testejä muokata, koska olimme lisänneet luokkiimme ominaisuuksia. Jenkinsin käyttöönotto ei toiminut ongelmitta, sillä käytimme Javan versiota 11 ja toimivat ohjeet olivat vain Javan versiolle 8. Tämä johti aiheen ylimääräiseen opiskeluun, mikä vei jonkin verran aikaa.

Kuvassa 14 näkyy katsaus testiemme tuloksiin Jenkinsin käyttöönoton jälkeen.



Kuva 14. Jenkins-palvelimen automaattisten testien tulokset.

Aluksi osassa testeistämme oli paljon ongelmia saada muistiinpanot ja käyttäjät tietokantaan. Nämä vaikeudet näkyvät kuvassa 14 koontiversioon 36 saakka, jonka jälkeen

saimme muistiinpanot ja käyttäjät hyvin tietokantaan. Tämän jälkeen ei ole ollut suuria ongelmia testien kanssa ja suurimmalta osin testiajot ovat onnistuneet pääasiassa hyvin ja vakaasti.

Harmittavasti Jenkins-palvelimen tekemänä osaa testeistä ei kuitenkaan voi ajaa, koska Jenkins ei osaa ottaa huomioon niiden JavaFX-tarpeita. Nämä JavaFX ongelmat näkyvät koontiversiosta 52 koontiversioon 63 saakka. Paikallisesti Mavenilla ajettuna nämäkin testit menevät läpi.

7.3 Graafisen käyttöliittymän testaus

Graafisen käyttöliittymän testaukseen ei ole tehty yhtään mekaanista testiä, sillä sen testaaminen on hyvin hankalaa koodatuilla testeillä. Sen sijaan testasimme käyttöliittymän manuaalisesti itse lukuisten yritysten ja erehdysten kautta. Otimme myös pari ulkopuolista ihmistä mukaan aina välillä testaamaan käyttöliittymäämme.

Ohjelmistotuotantoprojekti 1 -kurssin ohella kulki myös Käyttäjäkeskeinen suunnittelu-kurssi, josta oli paljon hyötyä käyttöliittymän parantamisessa. Kurssilla käytiin läpi käyttöliittymää ja monet opiskelijat tekivät parannusehdotuksia, kuten myös opettaja teki. Nämä parannusehdotukset todettiin hyviksi ja ne toteutettiin sovelluksen käyttöliittymään ja testattiin manuaalisesti toimiviksi.

Käyttöliittymä on myös luonteeltaan hyvin vapaa, joten virheitä ei voi käyttäjän toimesta tulla paljoa. Vapaalla tarkoitetaan tässä tapauksessa toimintoja, joissa käyttäjällä on mahdollisuus tehdä lähes mitä vain. Vapaita tekstikenttiä ohjelmassa on kolme (lukuun ottamatta kirjautumis- / rekisteröitymislomakkeita), joista kaksi ovat päänäkylässä. Näihin päänäkylässä oleviin tekstikenttiin käyttäjä voi kirjoittaa mitä haluaa, ainoastaan otsikossa maksimipituus on asetettu 32 merkkiin, jotta näkymän vasemmalla puolella oleva lista ei menisi liian tukkoon. Asetusvalikossa oleva kolmas tekstikenttä, joka asettaa oletuspolun muistiinpanojen tallentamiselle, voi sisältää virheellistä tietoa käyttäjän toimesta. Olemme kuitenkin tähän ongelmaan tarjonneet käyttäjälle painikkeen, joka asettaa käyttäjän valitseman polun automaattisesti tekstikenttään, jos tällaista ominaisuutta haluaa käyttää.

8 Kehittäjää kiinnostavaa

Tässä luvussa esitellään sovelluksesta muutama sellainen ominaisuus, jotka saattavat olla erityisen kiinnostavia sovelluskehittäjien mielestä.

Jatkokehityksen kannalta kehittäjää kiinnostavia asioita voisi olla nykyisen *RichTextFX*-kirjaston vaihtaminen parempaan tai kyseisen kirjaston opettelu läpikotaisin. Kyseisen kirjaston kanssa oli lukuisia ongelmia, sillä dokumentaatio oli puutteellista eikä käynyt kaikkia perusteellisia asioita läpi. Kirjaston tarjoama tekstikenttä on myös vaikeakäyttöinen meidän kaltaisessamme sovelluksessa, jossa käytämme samaa tekstikenttää monien muistiinpanojen näyttämiseksi.

Muistiinpanojen lista sovelluksessa on tehty mukautetuilla listasoluilla, joihin tieto päivittyy dynaamisesti, riippuen muistiinpanosta. Muistiinpanon listasolun nimi, päiväys sekä prioriteetti päivittyy sen mukaan dynaamisesti, mitä käyttäjä tekee muistiinpanolle. Kielen vaihto vaikuttaa päiväyksen formatointiin. Listasolut voi nähdä kuvassa 1.

Tekstin muokkausominaisuudet toimivat puhtailla CSS-tyylittely säännöillä. Säännöt määritellään tekstikenttään riippuen siitä, mitä tyyliä on napsautettu (ks. Kuva 4). Säännöille on määritelty oma luokka nimeltä *FontStyle*, joka lisää tekstikenttään CSS-sääntöjä käyttäjän valintojen mukaan oikeaan paikkaan (maalatun tekstin kohdalle). Näitä sääntöjä voisi muokata niin, että ne toimisivat paremmin yhdessä.

Ohjelman asetusvalikon logiikka on lainattu IntelliJ IDEA ohjelmistoympäristön asetusvalikosta. Tumman teeman sävyt ovat samanlaisia kuin Discord-pikaviestintäsovelluksessa. Inspiraatio sovelluksessa oleviin informatiivisiin ponnahdusikkunoihin on otettu Android-puhelinten ponnahdusikkunoista. Sovelluksen päänäkökulma on saanut vaikutteita Windows-järjestelmän tiedostojen selausikkunasta yksinkertaisuuden vuoksi.

9 Yhteenveto

Sovelluksemme on muistiinpanosovellus, joka toteuttaa arkkitehtuuriltaan MVC-ohjelmistoarkkitehtuuria. Idean sovellukseen saimme siitä, kun tutkimme erilaisia kehitysvaihtoehtoja ja totesimme, että ei ole monia ilmaisia muistiinpanosovelluksia, joissa olisi myös piirto-ominaisuus. Halusimme myös panostaa helppokäyttöisyyteen, turvallisuuteen sekä nopeuteen.

Toteutimme sovellukseen monia mielenkiintoisia ominaisuuksia. Tärkeimpinä toimiva muistiinpanojen luonti, muistiinpanojen personoiminen ja muistiinpanojen sekä käyttäjien turvassa pitäminen. Lisäksi toteutimme muun muassa automaattisen pilveen tallentamisen, erilaisia tekstinmuokkausmahdollisuuksia, perusteellisen käyttäjäturvallisuuden, käyttäjäystävällisen käyttöliittymän, sekä paljon muuta.

Toteutimme projektin hyödyntäen ketterää Scrum-metodologiaa ja työskentelyaika oli jaettu lyhyisiin sprintteihin. Sprinttejä suunnitellessa hankalinta oli kehitysajan ennustaminen, ja monien ominaisuuksien toteuttamiseen meni paljon enemmän aikaa kuin olimme alun perin arvelleet. Paljon aikaa kului uusien kirjastojen ja ohjelmien opettelussa, johon kulunutta aikaa oli vaikea arvioida. Tämä aiheutti muun muassa sen, että jouduimme jättämään haluamamme piirto-ominaisuuden pois.

Loppujen lopuksi saimme kuitenkin toteutettua vakaasti toimivan kokonaisuuden, joka toteuttaa lähes kaikki haluamamme pääominaisuudet. Voimme olla erittäin tyytyväisiä ryhmämme työskentelyyn ja työmme hedelmiin.

10 Linkkejä

Projektin GitLab: <https://gitlab.metropolia.fi/nicoja/ohjelmistotuotantoprojekti-1>

Projektin JavaDoc:

<https://gitlab.metropolia.fi/nicoja/ohjelmistotuotantoprojekti-1/-/tree/master/docs/Java-Doc>

Projektin back end: <https://gitlab.metropolia.fi/jerejs/memoable-backend>

Back end JavaDoc:

https://gitlab.metropolia.fi/jerejs/memoable-backend/-/tree/master/backend_javadoc

Java 11 asennusohjeet:

<https://www.oracle.com/java/technologies/javase-jdk11-downloads.html>

MongoDB asennusohjeet: <https://docs.mongodb.com/manual/installation/>

Docker asennusohjeet: <https://docs.docker.com/engine/install/>

Maven asennusohjeet: <https://maven.apache.org/install.html>

VPN asennusohjeet:

<https://wiki.metropolia.fi/pages/viewpage.action?pageId=149652071>