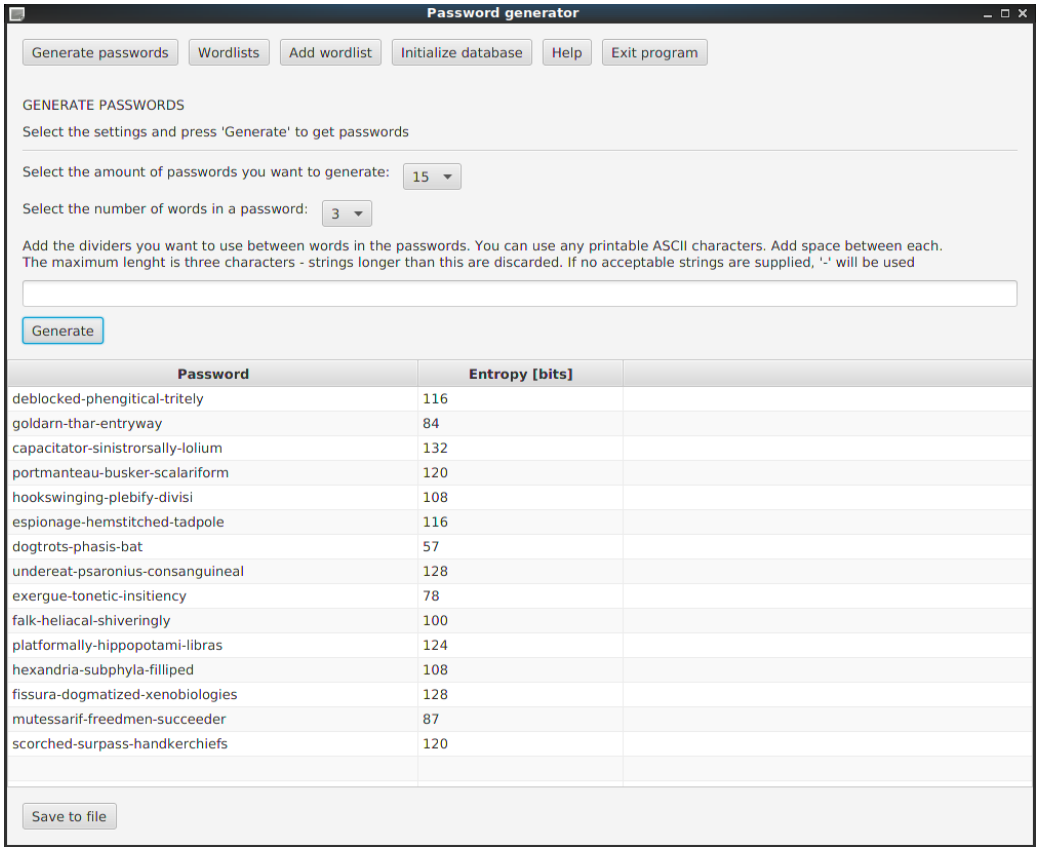
Ohjelmistotekniikka-kurssin loppuraportti

Kurssille AYTKT20002_k2019 – palautettu 1.5.2019

Antti Kosonen
013761802



Sisällysluettelo

1 VAATIMUSTENMÄÄRITTELY 1
2 KÄYTTÖOHJE 2
3 SOVELLUSLOGIIKKA 4
3.1 Ohjelman rakenne yleisellä tasolla 4
3.2 Graafinen käyttöliittymä 5
3.3 Tietokanta, tietokantayhteys sekä mallioliot 6
3.4 Sanalistan lisäys 7
3.5 Sanalistojen katsominen 8
3.6 Sanalistan poisto 8
3.7 Salasanojen generointi 9
3.8 Salasanojen talletus tiedostoon 11
4 TESTAUS 12
4.1 Yksikkötestaus 12
4.2 Integraatiotestaus 13
4.3 Järjestelmätestaus 13
5 PÄIVÄKIRJA JA ITSEREFLEKTIO 16
5.1 Ensimmäinen viikko 16
5.2 Toinen viikko 17
5.3 Kolmas viikko 17
5.4 Neljäs viikko 18
5.5 Viides viikko 18
5.6 Kuudes viikko 19
5.7 Seitsemäs viikko 19
5.8 Kahdeksas viikko 20
5.9 Yhdeksäs viikko 20
5.10 Kymmenes viikko 20
6 LIITTEET 21

1 VAATIMUSTENMÄÄRITTELY

Ihmiset käyttävät usein salasanoja joita on vaikea muistaa ja kirjoittaa, mutta jotka ovat tietokoneellisesti helppo murtaa. Oikeasti hyvän salasanan – helposti muistettavan ja pitkän – koostamiseen on olemassa hyviä ohjeita, joita ei ikävä kyllä usein käytetä (https://xkcd.com/936/). Tälläisiä salasanoja voisi luoda myös ohjelmalla. Valitsin harjoitustyöksi tälläisen ohjelman kirjoittamisen.

Käyttäjärooleja ohjelmalla on vain yksi. Ohjelman toiminnalliset vaatimukset ovat:

- Käyttäjä voi ladata ohjelmaan sanalistoja, joista salasana koostetaan
- Käyttäjä voi lisäksi ladata listoja sanoista, joita salasana ei saa sisältää (esimerkiksi yleisten salasanojen listoja)
- Sanalistoista ja niiden lisäyksen ajankohdista pidetään kirjaa, ja käyttäjä voi poistaa lataamiaan sanalistoja. Tätä varten sanat laitetaan SQL-tietokantaan, johon luodaan taulut myös lisäyksille.
- Käyttäjä voi katsella ohjelmaan talletettuja sanalistoja
- Sanojen lisäksi käyttäjä voi määritellä sääntöjä luotavalle salasanalle, kuten
    - Käytettyjen sanojen lukumäärä
    - Sanojen välissä käytetyt merkit
- Salasanoja luodaan käyttäjän haluama määrä, joka voidaan myös tallentaa tiedostoon käyttäjän niin halutessa
- Generoiduille salasanoille lasketaan niiden vahvuutta kuvaava lukema

Projektisuunnitelma:

1. iteraatio: Sanalistoja voi lisätä ja katsoa, ja niistä voi muodostaa yjksinkertaisia salasanoja

2. iteraatio: Salasanan muodostamiseen voi luoda yksinkertaisia sääntöjä. Sanalistoihin tallennetaan niiden luontiajankohta. Sanalistoja voi poistaa.

3. iteraatio: Ohjelmalla on yksinkertainen graafinen käyttöliittymä

4. iteraatio: Graafinen käyttöliittymä kattaa ohjelman koko toiminnallisuuden. Sanalistoja voi ladata tiedostoista ja salasanalistoja voi tallentaa tiedostoon.

5. iteraatio: Kullekkin generoidulle salasanalle lasketaan Shannon-entropian avulla minimipakkauskoko, jotta käyttäjä voi arvioida sen vahvuutta.

2 KÄYTTÖOHJE

Käyttöohje – kuten ohjelman kommentointikin - on kirjoitettu englanniksi, sillä ohjelma on suunnattu kansainväliselle yleisölle:

This is a program to generate passwords that are both secure and easy to remember. You can load lists of words to the program and use them to generate passwords. The logic behind it has perhaps been best explained by the cartoon called XKCD:

https://xkcd.com/936/

There already are some apps to generate good passwords, but they all are missing some functionality that I think is central. Users should be able to use their own wordlists - this way they can generate passwords in their own language, and even dialect. They can choose words words that are familiar to them, but unfamiliar to others (like words connected with their hobbies, names of local places etc.)

Installation:

To use this application you need to have the Java JRE 8 which can be downdloaded here:

https://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html

You also need the H2 database engine, which you can get here:

https://h2database.com/html/main.html

This application has only been tested on a linux-system – it might not work correctly on other systems.

After you have installed JRE 8 and the H2 engine, you need to download either the source code, or just the jar-file from the /target -directory.

To download the source code run the following command:

git clone https://github.com/anttihtkosonen/correcthorse

After downloading run the code with an IDE of your choice, or you can compile it to bytecode and run it on the command line of your operating system.

To get the jar-file, run the following command:

wget https://github.com/anttihtkosonen/correcthorse/tree/master/target/ Passwordapplication-1.0-SNAPSHOT.jar

You can run the jar with the following command:

java -jar Passwordapplication-1.0-SNAPSHOT.jar

Usage:

You need to first add some wordlists to the database, and after that you can use the words to generate passwords. There are two kinds of wordlists: whitelists and blacklists. Whitelists include words that are used to generate passwords, and blacklists include words that are banned from them. Only words that are

on a whitelist and are not on any blacklist are used to generate passwords. You can find useful wordlists here:

https://github.com/imsky/wordlists

If you are adventurous, you can add every word in the english language (please note, that this will take several minutes for the application to process):

https://raw.githubusercontent.com/dwyl/english-words/master/words_alpha.txt

It is advisable to use lists of common passwords as blacklists. For example, here is a list of the 10 000 most popular passwords – a good candidate for a blacklist:

https://raw.githubusercontent.com/danielmiessler/SecLists/master/Passwords/darkweb2017-top10000.txt

In it's current state of development the application only accepts words made of letters. Words that include numbers, special characters or any characters outside the ASCII letter range (65-90 and 97-122) are not added to the database, but rejected altogether. Also, words longer than 30 characters are rejected. If there are any duplicates on the list, they will be removed.

To add a wordlist, do the following:

1. Go to the "Add wordlist" tab in the application.
2. Give your wordlist a name (max. 50 characters) in the small text field at the top
3. Select whether it is a blacklist or not, at the dropdown menu
4. Paste the words in the text area below. Each word should be on its own line – if you include words delimited by spaces, tabs, semicolons, commas or other characters, they will be rejected. Alternatively you can use the "Load list from file" -button to load a list from a .txt -file to the field.
5. Press "Add list"

To generate passwords, do the following:

1. Go to the "Generate password" tab in the application
2. Use the dropdown menus to select how many passwords you want to generate, and how many words should be in a password
3. In the text field add the characters that you want to use in the password between words. These will be used randomly. If none are added, '-' will be used throughout.
4. Press "Generate"
5. After the passwords have been generated, you can save them to a file by pressing "Save to file" at the bottom of the window

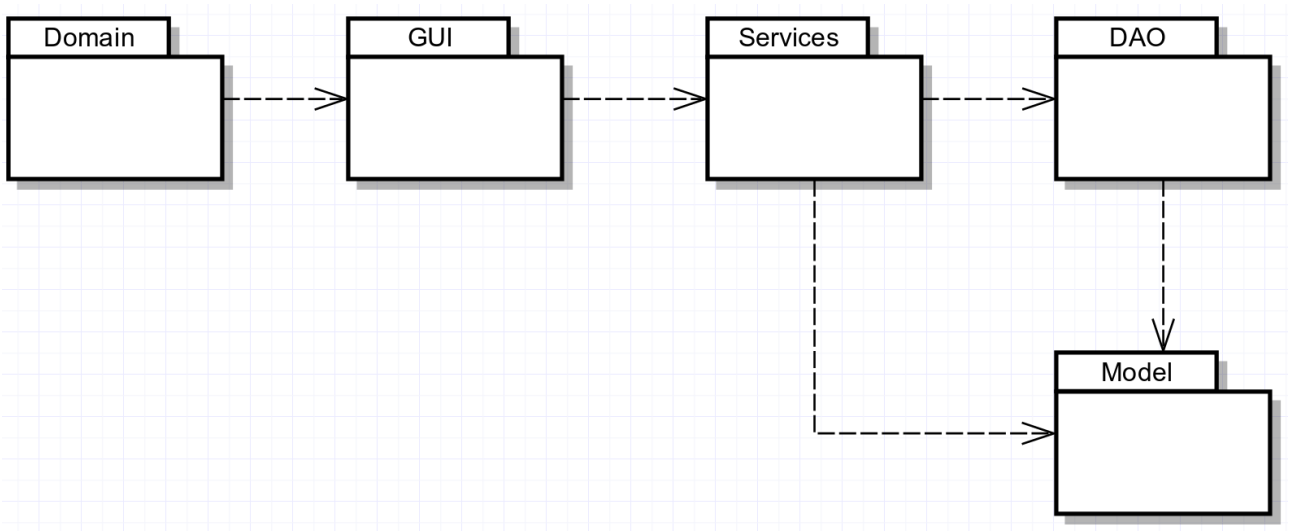
The application also calculates the entropy of each password. The number represents the theoretical minimum number of bits required to store the password in memory – it is the Shannon-entropy of the characters rounded up and multiplied by the number of characters. As a rule of thumb a larger number is the better. It should be noted however, that a large number of bits itself does not ensure that the password

is strong – the words used should also be uncommon, otherwise the password is vulnerable to dictionary-based attacks where an attacker uses lists of common words to brute-force the password. Use of rare or specialized words is strongly recommended.

3 SOVELLUSLOGIIKKA

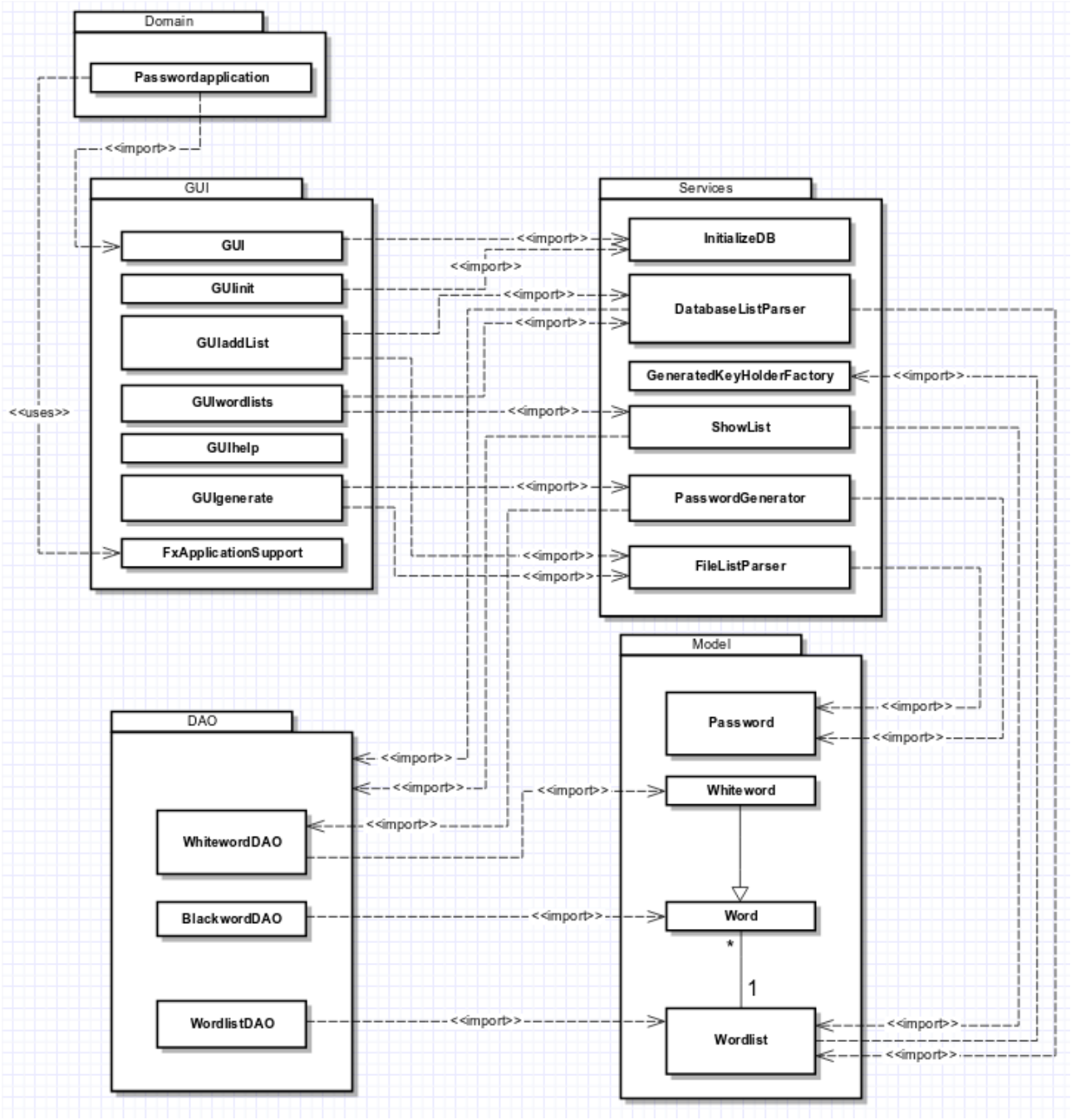
3.1 Ohjelman rakenne yleisellä tasolla

Yleisellä tasolla käyttöliittymän *GUI*-pakkauksesta kutsutaan *Services*-pakkauksen palveluita, jotka edelleen kutsuvat *DAO*-pakkauksen palveluita, jotka toteuttavat tietokantaoperaatiot. Sekä *DAO*-, että *Services*-pakkaukset käyttävät *Model*-pakkausta olioiden luontiin ja käsittelyyn. *Domain*-pakkauksessa on ohjelman käynnistävä pääluokka, joka käynnistää graafisen käyttöliittymän.



Drawing 1: Ohjelman pakkausrakenne

Ohjelman toiminnalla on kaksi päähaaraa: sanalistojen hallinta (lisäys, katselu ja poisto) sekä salasanat (luonti ja tallennus). Seuraavissa aliluvuissa on selvitetty tarkemmin ohjelman toiminnallisuudet.



Drawing 2: Ohjelman luokkarakennekaavio

3.2 Graafinen käyttöliittymä

Ohjelman käynnistyessä ohjelman pääluokka *Passwordapplication* kutsuu *GUI* –luokkaa, joka luo ohjelman pääikkunan. Tähän käytetään apuna *FxApplicationSupport* –luokkaa, joka luo oikean kontekstin Spring-bootin ja JavaFX:n yhteistoiminnalle. *Passwordapplication* sijaitsee *Domain*-pakkauksessa, kun taas kaikki graafiseen käyttöliittymään liittyvät luokat – mukaan lukien *FxApplicationSupport* – sijaitsevat *GUI*-pakkauksessa.

Käynnistyessään ensimmäisen kerran, *GUI* lataa ikkunan salasanojen generointia varten luokasta *GUIgenerate*. Ikkunan yläreunassa on aina setti näppäimiä, joiden avulla käyttäjä voi vaihtaa ikkunasta toiseen – näitä ovat:

- Salasanojen generointi ja talletus (*GUIgenerate*)
- Sanalistojen selailu ja poisto (*GUIwordlists*)
- Sanalistojen lisäys (*GUIaddlist*)
- Käyttöohjesivu (*GUIhelp*)
- Tietokannan alustus (*GUIinit*)
- Ohjelman sulkeminen (toiminnallisuus pääluokassa *GUI*)

3.3 Tietokanta, tietokantayhteys sekä mallioliot

Ohjelma käyttää tietokantana H2 *SQL*-tietokantaa, johon otetaan yhteys Java Database Connectivity –rajapinnan kautta *JdbcTemplate*-kirjastoa käyttäen. Mikäli tietokantatiedostoa ei ole olemassa ohjelman käynnistyessä, se luodaan automaattisesti. Ohjelman käyttöliittymässä on myös ikkuna, jossa käyttäjä voi alustaa tietokannan. Tällöin kaikki tietokannan taulut pudotetaan, ja luodaan uudestaan.

Tietokannassa on kolme taulua. *Wordlist*-taulussa ovat sanalistojen metatiedot: nimi, luontiaika, tieto siitä onko kyse kiellettyjen sanojen listasta vai tavallisesta, sekä taulun pääavaimena tietokannan luoma id-numero. Itse sanat ovat kahdessa eri taulussa – kiellettyjen listojen sanat *Blackword*-taulussa , ja tavalliset sanat *Whiteword*-taulussa. Kustakin sanasta tallennetaan itse sana, sekä sen listan id-numero johon sana kuuluu. Lisäksi *Whiteword*-tauluun tallennetaan myös Boolean-tyyppinen arvo, joka kertoo onko sana aktiivinen (sanaa ei ole *Blackword*-taulussa) vai epäaktiivinen (on myös *Blackword*-taulussa). Tämä johtuu siitä, että sellaisia *Whiteword*-taulun sanoja jotka ovat myös *Blackword*-taulussa ei käytetä salasanoissa. Kyse on tietokannan denormalisoinnista, jonka tarkoituksena on nopeuttaa salasanojen generointiprosessia.

Ohjelmassa on kullekkin tietokannan taululle oma Data Access Object –luokka, joka huolehtii tietokantaoperaatioista tuon luokan kohdalla. Nämä sijaitsevat *DAO*-pakkauksessa. Näiden luokkien metodit tekevät ainoastaan tietokantaoperaatioita, eivätkä sisällä varsinaista ohjelmistologiikkaa.

Ohjelmalogiikassa sanalistoja ja sanoja käsitellään olioina: sanalistoja *Wordlist*-oliona, ja sanoja *Word*-oliona (kielletyt sanat) tai *Whiteword*-olioina (tavalliset sanat). *Whiteword* perii *Word*-olion, sillä näissä ovat samat ominaisuudet, lukuunottamatta aktiivisuusstatusta, joka koskee vain tavallisia sanoja. Näiden lisäksi ohjelmassa käytetään myös *Password*-oliota, joka sisältää generoidun salasanan sekä tämän entropian.

3.4 Sanalistan lisäys

Kun käyttäjä lisää sanalistan ohjelmaan *GUIaddlist*-luokan ikkunassa, häntä pyydetään antamaan sanalistalle nimi, valitsemaan dropdown-valikosta onko kyseessä musta lista vai ei, ja antamaan listan sanat. Musta lista on lista sanoja, joita ei saa käyttää salasanoissa (tähän sopii esimerkiksi lista yleisistä salasanoista). Sanaa käytetään salasanassa vain jos se on sanalistalla, eikä se ole millään mustalla listalla.

Käyttäjä voi joko leikata ja liimata sanalistan sanat ikkunan tekstikenttään, tai hän voi ladata ne tiedostosta. Jälkimmäisessä tilanteessa luodaan *FileChooser*-kirjastoluokan avulla käyttöjärjestelmän valintaikkuna, jossa käyttäjä voi valita tiedoston, jossa sanalista on. Tämän jälkeen kutsutaan *getListFromFile*-metodia luokasta *FileListParser* joka on *Services*-pakkauksessa, ja tälle syötetään käyttäjän valitseman tiedoston sijainti. Tämä metodi avaa kyseisen tiedoston, lukee sen sisältämät merkit Stringiin, ja palauttaa tämän takaisin *GUIaddlist*-luokalle, joka syöttää ne tekstikenttään.

Kun sanalistan tiedot on tavalla tai toisella täytetty, käyttäjä painaa "Add list"-nappia lisätäkseen sanalistan tietokantaan. Tällöin kutsutaan *Services*-pakkauksessa olevan *DatabaseListParser*-luokan *addList*-metodia, jolle syötetään parametreinä käyttäjän antamat tiedot sekä senhetkinen aika Timestamppina. Tämä metodi luo ensin sanalistan tiedoista *Wordlist*-olion ja syöttää sen tietokantaan *WordlistDAO*-luokan *insert*-metodilla, joka palauttaa listan id-numeron tietokannassa.

Kun sanalista on lisätty, *addList*-metodi käy seuraavaksi läpi jokaisen sanalistan rivin. Ensiksi tarkistetaan, onko sana jo aiemmin lisättyjen sanojen listalla, ja jos näin ei ole kutsutaan saman luokan *parseLine*-metodia, antaen sille parametrina kyseisen rivin Stringinä, listan blacklist-statuksen Boolean-arvona sekä listan id-numeron. Tämä metodi tekee sanalle ensin erilaisia tarkastuksia, ja hylkää sen mikäli se ei vastaa sanalle asetettuja vaatimuksia. Tämän jälkeen kutsutaan edelleen *addWord*-metodia, joka tarkistaa onko lisättävä sanalista kiellettyjen sanojen lista vai ei, ja tarvittavat muutokset tietokantaan tehdän tämän pohjalta.

Mikäli kyse on mustan listan sanasta, *addWord* -metodi lisää sanan tietokantaan *BlackwordDAO*-luokan *insert*-metodilla, ja tämän jälkeen se asettaa sanan mahdolliset ilmentymät epäaktiivisiksi *Whiteword*-taulussa *WhitewordDAO*-luokan *setInactive*-metodilla.

Jos taas kyse on tavallisesta sanasta, *addWord* -metodi selvittää ensin löytyykö sana jo tietokannan *Blackword*-taulusta *BlackwordDAO*-luokan *find*-metodilla, ja sanan aktiivisuusstatus asetetaan vastaamaan tätä. Tämän jälkeen sana lisätään tietokantaan *WhitewordDAO*-luokan *insert*-metodilla.

Mikäli sana lisätään tietokantaan, *parseLine* lisää sen myös lisättyjen sanojen listalle, ja kun *addList*-metodin *for*-loopin seuraavassa iteraatiossa *parseLine*-metodia kutsutaan antamalla parametriksi tämä päivitetty listä, ei tuota sanaa lisätä tietokantaan enää toista kertaa.

Sanan lisäys heittää *SQLException*-virheen mikäli jokin tietokantaoperaatio syystä tai toisesta epäonnistuu, ja *IOException*-virheen mikäli sanalistan haku käyttöjärjestelmästä epäonnistuu. Nämä virheet otetaan käyttöliittymätasolla kiinni, ja käyttäjälle esitetään virhettä vastaava virheviesti.

3.5 Sanalistojen katsominen

Käyttäjän siirtyessä sanalistojen katseluikkunaan *GUIwordlists*-käyttöliittymäolio luo ikkunaan tekstikentän, ja täyttää siihen sanalistojen tiedot kutsumalla *Services*-pakkauksessa olevan *ShowList*-luokan *showAll*-metodia.

Tämä metodi hakee ensin listan tietokannassa olevista sanalistoista *Wordlist*-olioina *WordlistDAO*-luokan *list*-metodilla. Tämän jälkeen se käy jokaisen listan olion läpi, ja luo kustakin Stringin, joka kertoo listan tiedot ihmisen luettavassa muodossa, sekä sanalistan kymmenen ensimmäistä sanaa. Nämä Stringit lisätään toiselle listalle, jonka metodi lopuksi palauttaa. Käyttöliittymässä nämä Stringit ladataan käyttäjän nähtäville.

Ikkunan lataaminen heittää *SQLException*-virheen mikäli jokin tietokantaoperaatio syystä tai toisesta epäonnistuu - tämä otetaan käyttöliittymätasolla kiinni, ja käyttäjälle esitetään virhettä vastaava virheviesti.

3.6 Sanalistan poisto

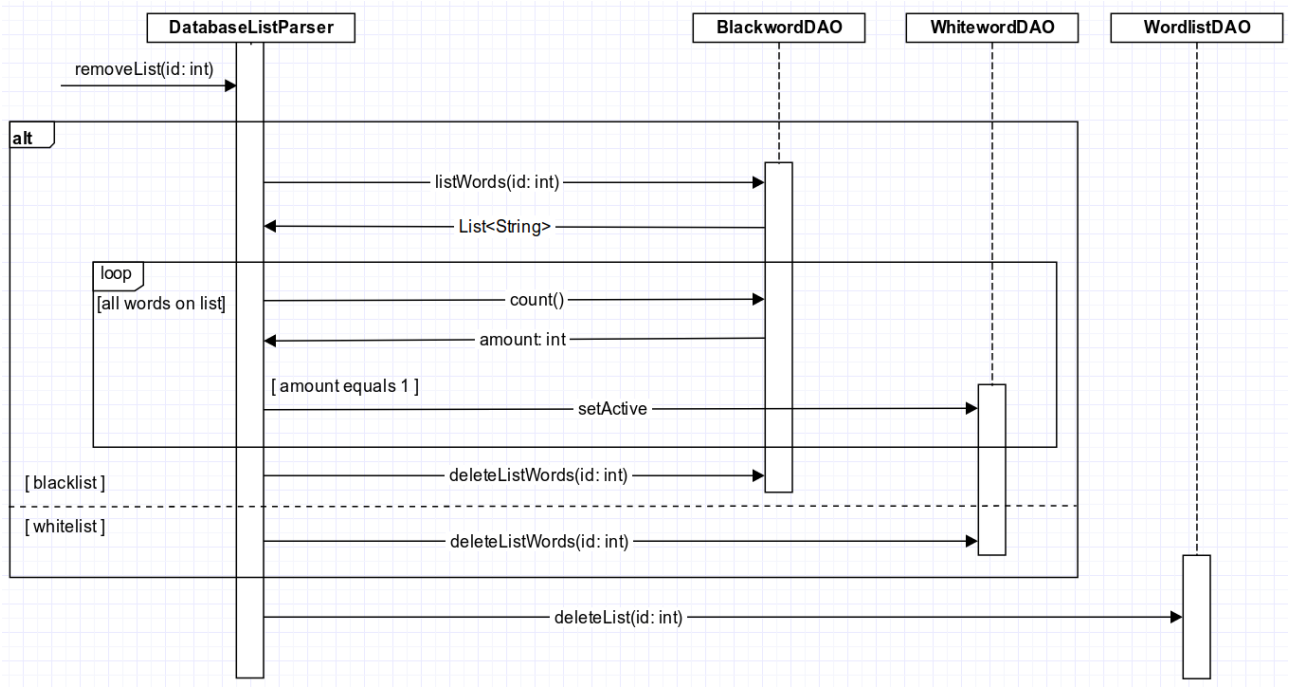
Edellisessä aliluvussa kuvatussa sanalistojen katsomisikkunassa on mahdollisuus valita sanalista ja poistaa se painamalla "Delete selected"-nappia. Näin tehtäessä käyttöliittymä ensin hakee valitun sanalistan id numeron tietokannassa kyseisen sanalistan esittely-Stringistä ja sen jälkeen kutsuu *DatabaseListParser*-luokan *removeList*-metodia antaen parametriksi tämän numeron.

Tämä metodi aluksi selvittää listan blacklist-statuksen *WordlistDAO*-luokan *readBlacklistStatus*-metodilla. Mikäli lista on kiellettyjen sanojen lista, tarkoittaa listan sanojen poistaminen tietokannasta sitä, että joidenkin tavallisten sanojen aktiivisuusstatus saattaa muuttua. Näin tapahtuu kuitenkin vain siinä tapauksessa, että kyseistä sanaa ei löydy toiselta kiellettyjen sanojen listalta *Blackword*-taulusta.

Mikäli poistettava lista on kiellettyjen sanojen lista, metodi ensimmäisenä hakee tietokannasta kaikki sanalistan sanat kutsumalla *BlackwordDAO*-luokan *listWords*-metodia. Tämän jälkeen listan jokaisen sanan kohdalla selvitetään *BlackwordDAO*-luokan *count*-metodilla ilmeneekö kyseinen sana toiseen tai useamaan kertaan *Blackword*-taulussa. Mikäli sana ilmenee toiseen kertaan, mitään ei muuteta, mutta jos sana on ainoa laatuaan, niin siinä tapauksessa kyseisen sanan kaikki ilmentymät *Whiteword*-taulussa muutetaan aktiivisiksi käyttämällä *WhitewordDAO*-luokan *setActive*-metodia. Kun kaikki sanalistan sanat on käyty läpi, ne poistetaan *Blackword*-taulusta käyttämällä *BlackwordDAO*-luokan *deleteListWords*-metodia, joka poistaa kaikki sanat jotka vastaavat parametrina annettua sanalistan id-numeroa.

Mikäli poistettava lista ei ole kiellettyjen sanojen lista, on operaatio huomattavasti yksinkertaisempi: sanat voidaan suoraan poistaa käyttämällä *WhitewordDAO*-luokan *deleteListWords*-metodia.

Kun sanalistan sanat on poistettu tietokannasta, metodi lopuksi poistaa itse sanalistan tiedot *Wordlist*-taulusta *WordlistDAO*-luokan *deleteList*-metodilla.



Drawing 3: Sanalistan poiston sekvenssikaavio

Kun tietokantaoperaatiot on suoritettu, käyttöliittymäolio lopuksi poistaa sanalistan esittely-Stringin käyttäjälle esitettävien sanalistojen listasta ja päivittää ikkunan näkymän.

Poisto-operaatio heittää *SQLException*-virheen mikäli jokin tietokantaoperaatio syystä tai toisesta epäonnistuu - tämä otetaan käyttöliittymätasolla kiinni, ja käyttäjälle esitetään virhettä vastaava virheviesti.

3.7 Salasanojen generointi

Salasanojen generointisivulla – jonka luo GUIgenerate-luokka – käyttäjä voi luoda salasanoja tietokannassa olevista sanoista haluamillaan säännöillä. Käyttäjä pyydetään valitsemaan alasvetovalikoilla generoitavien salasanojen lukumäärä ja salasanaan käytettävien salasanojen lukumäärä, sekä listaamaan sanojen väliin lisättävät välimerkit. Nämä tiedot täytettyään käyttäjä painaa ”Generate”-nappia jolloin ohjelmistologiikka generoi salasanat ja listaa ne ikkunaan uuteen tekstikenttään.

Salasanat generoidaan kutsumalla *Services*-pakkauksessa olevan *PasswordGenerator*-luokan *getPasswords*-metodia, parametreinaan käyttäjän antamat tiedot. Tämä metodi hakee tietokannasta salasanoihin tarvittavan määrän sanoja *WhitewordDAO*-luokan *listNActiveStrings*-metodilla.

Tämä *listNActiveStrings*-metodi on varsin monimutkainen, joten se ansaitsee lähempää tarkastelua. Aluksi se käyttää saman luokan *countActiveWords*-metodia selvittääkseen onko tietokannassa riittävästi sanoja, ja palauttaa tyhjän listan, mikäli ei ole. Jos sanoja on riittävästi, se hakee tietokantataulun suurimman id-numeron ja sen jälkeen aloittaa while-loopin, joka pyörii niin kauan, että riittävä määrä sanoja on saatu kerättyä. Tässä loopissa luodaan toinen while-looppi jonka sisällä generoidaan satunnaisluku, koitetaan hakea tuota numeroa vastaava sana tietokannasta, ja mikäli tämä onnistuu, loopista poistutaan. Kun tällä

tavoin on haettu tietokannasta satunnainen sana, tehdään sille vielä kaksi tarkistusta while-loopin iteraation sisällä: tarkistetaan, että se on aktiivinen, ja että sitä ei ole jo aiemmin lisätty listalle. Mikäli sana läpäisee molemmat testit, se lisätään listalle, ja mennään seuraavaan while-loopin iteraation.

Kuten jo aiemmin kerrottiin, *listNActiveStrings* palauttaa suoraan tyhjän listan, mikäli tietokannassa on vähemmän aktiivisia sanoja kuin salasanoihin tarvitaan, mutta on myös mahdollista, että vaikka aktiivisia sanoja on riittävästi, niin eri sanalistoilla esiintyy samoja sanoja. Tällöin saattaa syntyä tilanne, että tietokannassa ei todellisuudessa ole riittävästi uniikkeja aktiivisia sanoja salasanojen generointiin, eikä *listNActiveStrings*-metodin while-loop pääty koskaan. Tämän estämiseksi metodiin on lisätty laskuri, joka lopettaa while-loopin kun se on ajettu läpi 10n kertaa, jolloin palautetaan tyhjä lista.

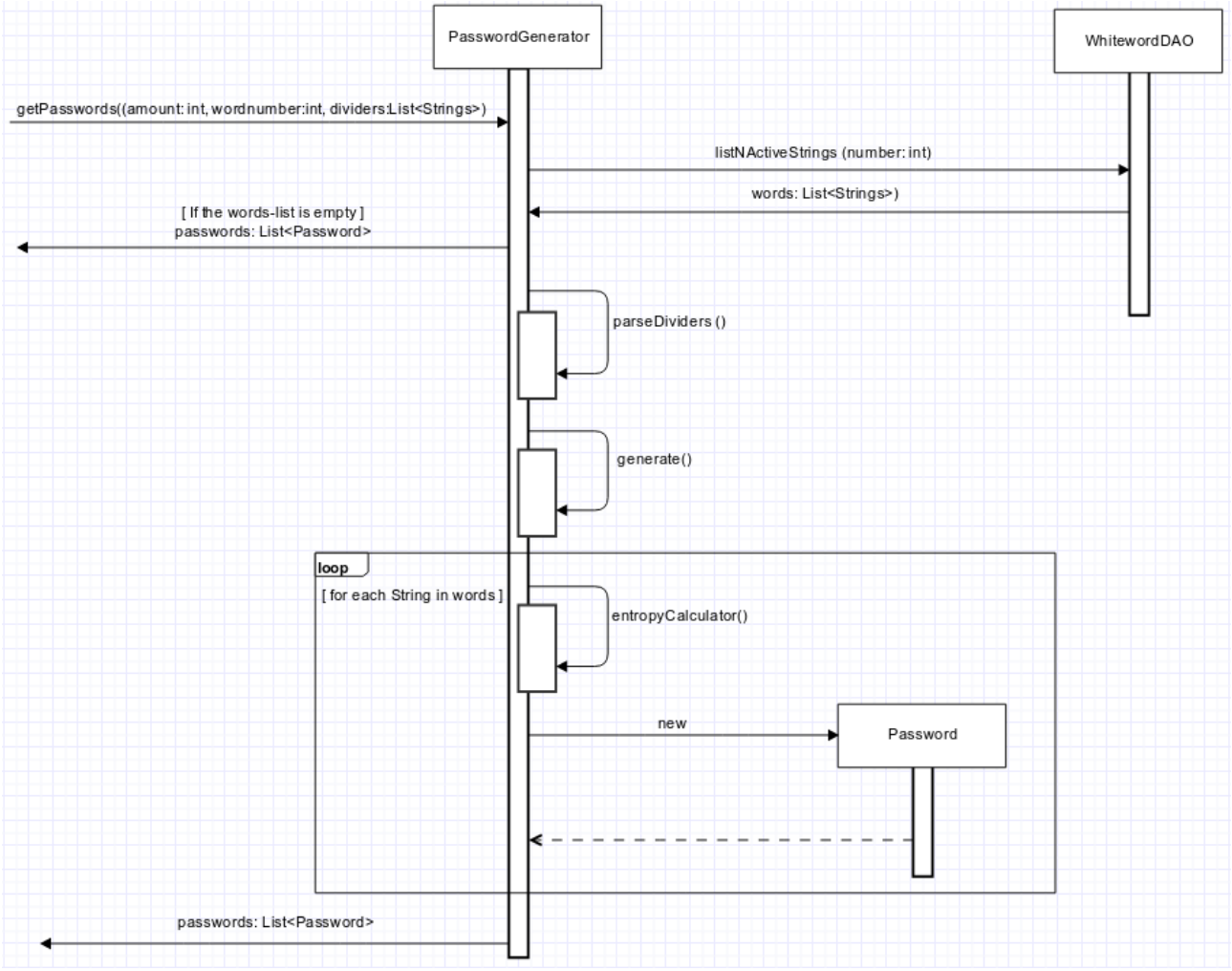
Kun *listNActiveStrings* on palauttanut listan sanoja *getPasswords*-metodille, tarkastaa tämä palautetun listan pituuden, ja mikäli tämä lista on tyhjä, se palauttaa suoraan tyhjän listan käyttöliittymämetodille. Tällöin ”Generate”-napin painaminen tuottaa virheilmoitusikkunan, jossa käyttäjälle kerrotaan asiasta. Mikäli sanalista on saatu onnistuneesti haettua, metodi tarkastuttaa käyttäjän antaman välimerkkilistan saman luokan *parseDividers*-metodilla.

Tämä *parseDividers*-metodi käy läpi käyttäjän antaman listan välimerkeistä, ja poistaa kaikki ne, jotka eivät vastaa vaatimuksia (korkeintaan 3 merkkiä pitkä merkkijono, vain printattavia ASCII-merkkejä). Mikäli käyttäjä ei antanut yhtään hyväksyttävää merkkiä, metodi lisää väliviivan listalle ja palauttaa sen.

Kun sekä sanalista että välimerkkilista ovat valmiina, *getPasswords* kutsuu saman luokan *generate*-metodia antaen parametreina nämä listat sekä salasanojen lukumäärän ja käytettävien sanojen määrän. Tämä metodi luo käyttäjän haluaman pituisen listan Stringejä, joihin on annetuista sanoista sekä välimerkeistä koottu salasanoja. Tämä palautetaan takaisin *getPasswords*-metodille.

Viimeisenä vaiheena *getPasswords* käy tämän palautetun listan lävitse, ja luo jokaisesta Stringistä Password-olion, jolle myös lasketaan entropia kutsumalla saman luokan *entropyCalculator*-metodia. Salasanan vahvuuden mittarina käytetään sitä bittien määrä, joka teoriassa minimissään tarvitaan merkkijonon pakkaamiseen – *entropyCalculator* laskee tämän arvon laskemalla salasanan merkkien Shannon-entropian, ja kertomalla tämän bittiarvon (ylöspäin pyöristettynä) merkkien lukumäärällä.

Lopuksi *getPasswords* palauttaa luomansa listan Password-olioita käyttöliittymälle, joka luo ikkunaan taulukon ja lataa salasanat sekä entropiat tämän kahteen sarakkeeseen.



Drawing 4: Salasanan luontiprosessin sekvenssikaavio

3.8 Salasanojen talletus tiedostoon

Kun käyttäjä on edellisessä aliluvussa kuvatulla tavalla luonut listan salasanoja, on hänellä myös mahdollisuus tallentaa sanalista tiedostoon. Näppäin sanojen tallentamiseen luodaan käyttöliittymän ikkunaan samalla kuin salasanataulukkokin, ja käyttäjän painaessa tätä käyttöliittymä luo *FileChooser*-kirjastoluokan avulla käyttöjärjestelmän valintaikkunan, jonka avulla käyttäjä valitsee tiedostolle nimen sekä sijainnin. Nämä tiedot tallennetaan *File*-olioon, jonka jälkeen kutsutaan *saveListToFile*-metodia luokasta *FileListParser* joka on *Services*-pakkauksessa, ja tälle annetaan sekä ohjelman tuottama *Password*-olioiden lista, että tämä *File*-olio argumentteina.

Aluksi *saveListToFile*-metodi avaa *FileWriter*-kirjastoluokan avulla tiedoston kirjoitettavaksi, ja tämän jälkeen *for*-loopissa kustakin listan *Password*-oliosta kirjoitetaan salasana-osuus tähän tiedostoon. Lopuksi tiedosto suljetaan.

4 TESTAUS

4.1 Yksikkötestaus

Yksikkötestejä ohjelman luokille on kirjoitettu yhteensä 49. Näitä ei käydä tässä yhteydessä lävitse jokaista erikseen, vaan niitä selostetaan pakkaustasolla, ja vain kiinnostavampia yksittäisiä testejä käydään lävitse tarkemmin.

Käyttöliittymän luokkia *GUI*-pakkauksessa ei testattu yksikkötesteillä, eikä testejä kirjoitettu myöskään *Domain*-pakkauksessa olevalle ohjelman pääluokalle, joka ei tee muuta kuin käynnistää ohjelman. Muissa pakkauksissa testattiin jokaisen luokan jokainen metodi kahta poikkeusta lukuunottamatta: *Services*-pakkauksessa jätettiin testaamatta tietokannan alustamisen tekevä luokka sekä *GeneratedKeyHolderFactory*-luokka jotka testataan järjestelmätestausvaiheessa.

4.1.1 DAO

Tässä pakkauksessa testien rivikattavuus on 86% ja haaraumakattavuus 90%. Puuttuvat rivit ja haarat joko liittyvät *Rowmapper*-olioihin - joiden yhteydessä jostain syystä laskettiin väliin jääviä rivejä vaikka tosiasiassa näitä rivejä ajetaan myös testejä suoritettaessa – sekä erään virhetilanteen kiinniottamiseen liittyvään *catch*-lauseeseen, jota ei yksikkötesteissä tullut vastaan. Yksi ohitettu haarauma oli *listNActiveStrings*-metodin rajatapaukseen liittyvä *while*-loopin pysäytyslause, joka testataan järjestelmätestausvaiheessa.

Koska tämän pakkauksen luokkien metodit tekevät yksinomaan tietokantaoperaatioita, ne eivät palauta mitään arvoja joita voisi tarkistaa testatessa. Tästä syystä testit tehdään ottamalla tietokantaoperaatioissa käytetyt parametrit kiinni *ArgumentCaptor*-kirjastoluokkaa käyttäen, ja ne tarkistetaan. Mikäli tietokantaoperaatioita tehdään väärillä parametreilla, testit epäonnistuvat.

Testi *WordlistDAO*-luokan *insert*-metodille muodostui hankalaksi, sillä ohjelmalogiikka vaatii, että tämä metodi palauttaa lisätyn sanalistan tietokanta-avaimen arvon. Tämä toteutettiin *GeneratedKeyHolder*-kirjastoluokalla, mutta tekoilmentymän luominent tästä oliosta ei ole kovin yksinkertaista. Ratkaisuna oli luoda kokonaan uusi ”tehdasluokka” *GeneratedKeyHolderFactory* joka luo avaimen, ja testaustvaiheessa luodaan tästä tehdasluokasta tekoilmentymä.

4.1.2 Models

Tässä pakkauksessa testien rivi- ja haaraumakattavuudet ovat 100%. Testit ovat varsin yksinkertaisia, sillä luokissa on vain konstruktoreja, gettereitä ja settereitä. Konstruktorit ja getterit testatataan yhdellä testillä, joka luo konstruktorin avulla olion ja sen jälkeen hakee gettereillä oliomuuttujien arvot. Setterit testataan samalla tavalla: arvo asetetaan, ja sen jälkeen haetaan getterillä.

4.1.3 Services

Tässä pakkauksessa testien rivikattavuus on 94% ja haaraumakattavuus 100%. Ohitetut rivit liittyvät *InitializeDB*- sekä *GeneratedKeyHolderFactory*-luokkiin, joille ei kirjoitettu yksikkötestejä, mutta joiden toiminnallisuus testattiin järjestelmätestausvaiheessa. Tämän luokan testit ovat logiikaltaan kaikkein monimutkaisimpia, ja niiden kehittäminen oli siksi varsin antoisaa.

DatabaseListParser-luokassa oli useita metodeja jotka eivät palauta mitään, ja niitä testattiin luomalla tekoilmentymät DAO-luokista ja tarkistamalla, että näitä kutsutaan oikeilla parametreilla. Metodeja tässä luokassa on useita, ja niiden kutsumiseen on paljon eri tapauksia, joten testejäkin tarvittiin yksitoista.

FileListParser-luokan kirjoitus- ja lukumetodeja testattiin luomalla väliaikaisia tiedostoja *TemporaryFolder*-kirjastoluokalla ja tarkistamalla, että kirjoitettava tai luettava data on oikeaa.

PasswordGenerator-luokan testit ovat varsin suoraviivaisia: metodeille syötetään merkkijonolistoja, ja tarkistetaan, että niistä luodaan oikeanlaisia salasanoja, tai mahdolliset virhetilanteet antavat oikeanlaisen tuloksen.

ShowList-luokassa on vain yksi metodi, ja sitä varten vain yksi testi. Testistä tuli varsin pitkä, sillä tämä metodi vaatii paljon parametreja, ja palauttaa varsin pitkiä merkkijonoja.

4.2 Integraatiotestaus

Ohjelmassa on yksi ylemmän tason integraatiotesti, pakkauksessa *passwordapplication.integration*. Tässä testissä luodaan tekoilmentymä *jdbcTemplate*-oliosta, joka jäljittelee tietokanta-yhteyden toimintaa – kaikki muu toiminnallisuus on jäljittelemätöntä.

Testissä luodaan lista *Pair*-olioita, joissa on kussakin sana ja aktiivisuustieto, ja teko-*jdbcTemplate* palauttaa nämä kun tietokannasta haetaan sanoja. Kun *PasswordGenerator*-luokan *getPasswords*-metodia kutsutaan erikseen määritellyillä parametreilla, tulee sen palauttaa lista salasanoja jotka on luotu näistä sanoista. Jokaiselle palautetulle salasanalle tehdään joukko testejä, ja koko integraatiotesti hylätään, mikäli yksikin salasana on esimerkiksi liian lyhyt merkkijono tai *null.*

4.3 Järjestelmätestaus

Ohjelman koko toiminnallisuus testattiin järjestelmätestausvaiheessa.

4.3.1 Graafisen käyttöliittymän ikkunat

Käyttöliittymän ikkunoiden toimintaa testattiin siirtymällä nopeasti ikkunasta toiseen satunnaisessa järjestyksessä. Ohjelma toimi nopeasti ja virheettä, ja jokaisessa ikkunassa oli oikea sisältö jokaisen siirtymisen jälkeen.

4.3.2 Sanalistan lisäys ja poistaminen

Sanalistan lisäämistä ohjeiden mukaan testattiin. Sanalista ilmestyi listalle, ja sen tiedot (pituus, asetettu blacklist-status sekä esimerkkinä annetut sanat listalta) olivat oikein. Saman listan lisääminen blacklist-statuksella meni myös läpi onnistuneesti.

Sanalistojen poistaminen listalta onnistui – sekä sanalistat, että niiden sanat poistettiin tietokannasta oikein.

4.3.3 Sanalistan lataaminen tiedostosta

Tekstimuotoisen sanalistan lataaminen ohjelman sanalistakenttään onnistui.

Myös virheellisessä muodossa olevan listan lataamista kokeiltiin: jar-tiedosto kopioitiin txt-muotoon, ja sen lataamista ohjelmaan sanalistana kokeiltiin. Ohjelma antoi tästä varoituksen, eikä ladannut sanalistaa.

4.3.4 Sanalistojen ilmoitettujen tietojen oikeellisuus

Latasin ohjelman tietokantaan viisi erilaista sanalistaa, ja sen jälkeen vertasin ohjelman kertomia sanalistojen tietoja siihen, mitä tietokantaan oli talletettu. Tutkin tietokantaa linuxin komentorivillä org.h2.tools.Shell -ohjelmalla. Ohjelma ilmoitti kaikki tiedot oikein.

4.3.5 Rasitustesti

Eräänlaisena ohjelman rasitustestinä koitettiin lisätä tietokantaan lista, jossa väitetään olevan jokainen englannin kielen sana, yhteensä vähän päälle 370 000 riviä:

https://raw.githubusercontent.com/dwyl/english-words/master/words_alpha.txt

Ohjelmalla kesti useita minuutteja prosessoida tämä lista, mutta se teki sen kuitenkin onnistuneesti. “Wordlists” -ikkunaan siirtymisessä oli tämän jälkeen havaittavissa hienoinen hidastuminen, oletettavasti johtuen siitä, että sanalistojen tiedot haetaan joka kerta tähän ikkunaan siirryttäessä, mikä tarkoittaa pitkien sanalistojen kohdalla raskaita tietokantahakuja. Yksi ohjelman tuleva kehityskohde voisi olla tämän tarpeettoman prosessoinnin poistaminen – sanalistojen tiedot tarvitsee hakea uudestaan vain kun sanalistoihin on tehty muutoksia.

4.3.6 Sanalistojen rajoitteiden testi

Sanalistan lisäyksen merkki- ja toistuvuusrajoitteita testattiin lisäämällä sanalista, jossa olivat seuraavat rivit:

```
test2
test-
/test
AVeryLongStringToTestTheMaximumLenghtLimitOfAWord
test'); DROP TABLE WORDLIST; --
しけん
Test test
test,     test
test      test
test;test
test
test
```

Ohjelma toimi kuten oli tarkoituskin, ja ainoastaan viimeinen rivi lisättiin tietokantaan

4.3.7 Sanalistan nimen rajoitteiden testi

Sanalistan nimen pituusrajoite on 50 merkkiä, mutta muita rajoitteita tälle ei ole asetettu. Sanalistan lisäämistä testattiin seuraavalla otsikolla:

```
Erittäin pitkä nimi joka ylittää nimen pituusrajoitteen
```

Tämä aiheutti varoituksen liian pitkästä nimestä, eikä sanalistaa lisätty tietokantaan. Kun nimeä lyhensi alle 50 merkkiin, sanalistan lisäys onnistui.

4.3.8 Salasanojen generointi ja tallennus tiedostoon

Salasanojen generointi onnistui ohjeiden mukaan, ja niiden talletus tiedostoon onnistui.

Salasanoja koitettiin myös luoda tietokannan ollessa tyhjä. Tällöin ohjelma antoi varoituksen riittämättömästä sanamäärästä.

4.3.9 Salasanojen generointi, kun uniikkeja sanoja ei ole tarpeeksi

Kuten ohjelman logiikan kuvauksessa todetaan on olemassa rajatapaus, jossa tietokannassa on näennäisesti riittävästi aktiivisia sanoja salasanojen generointiin, mutta osa niistä on duplikaatteja eri sanalistoilla, jolloin todellisuudessa uniikkeja aktiivisia sanoja ei ole riittävästi. Tällöin ohjelma katkaisee hakuprosessin kesken tietyn iteraatiomäärän jälkeen, ja ohjelma ilmoittaa sanoja olevan liian vähän.

Testasin tätä lisäämällä tietokantaan kaksi kertaa saman 12 sanan listan, ja tämän jälkeen lisäämällä kiellettyjen sanojen listan, jossa on kolme näistä sanoista. Tällöin tietokannassa on näennäisesti 18 aktiivista sanaa, mutta todellisuudessa uniikkeja aktiivisia sanoja on vain yhdeksän. Koitin tämän jälkeen luoda viittä kahden sanan salasanaa, mikä vaatii 10 uniikkia aktiivista sanaa. Tämän pitäisi johtaa virheilmoitukseen liian vähäisestä sanamäärästä.

Testissä ohjelma toimi kuten kuuluikin, ja virheilmoitus tuli erittäin nopeasti.

4.3.10 Salasanojen välimerkkien rajoitteiden testaus

Salasanojen generointia testattiin seuraavilla välimerkeillä:

```
-h- & ggg ####
```

Kuten kuuluukin, vain kolmea ensimmäistä välimerkkiä käytettiin salasanoissa, ja neljättä ei.

4.3.11 Tietokannan alustus

Tietokannan pystyi alustamaan onnistuneesti – kaikki sanalistat katosivat. Myös alustamisen keskeytys (“Cancel” -napin painaminen) toimi kuten kuului, eikä sanalistoja tällöin kadonnut.

4.3.12 Exit-nappi

Exit-nappi toimii kuten kuuluukin – ohjelman prosessi loppuu kokonaan.

5 PÄIVÄKIRJA JA ITSEREFLEKTIO

Opin kurssin aikana paljon. Olen aiemminkin kuullut iteratiivisesta ohjelmakehityksestä, testaamisen tärkeydestä sekä dokumentaation kirjoittamisen merkityksestä, mutta tällä kurssilla opin arvostamaan näitä aivan uudella tavalla. Löysin ohjelmistosta jopa joitain rajatapauksiin liittyviä bugeja okumentaatiota kirjoittaessani – katsomatta koodia ollenkaan. Myös esimerkiksi *Mockito*-kirjasto on sellainen asia, jonka opettelusta on tulevaisuudessa varmasti hyötyä – jos ei suoranaisesti juuri tästä kirjastosta, niin vastaavia tekoilmentymiä luovia kirjastoja on muitakin.

Työskentely sujui hyvin. Tein useimpina päivinä ainakin tunnin verran töitä kurssin eteen, mutta pääasiassa työ painottui viikonloppuihin. Pystyin seuraamaan iteraatiolle asettamiani vaatimuksia aika pitkälti – joitain pieniä osa-alueita jouduin siirtämään eteenpäin, kun en ehtinyt tehdä niitä suunnitellussa aikataulussa. Esimerkiksi alunperin olin suunnitellut sanalistan poiston olevan ensimmäisessä iteraatiossa toteutettava asia, mutta toteutin sen vasta toisessa.

Lopputulokseen olen tyytyväinen. Ohjelma tuottaa salasanoja juuri siten, kuin toivonkin, ja sen käyttö on suoraviivaista ja nopeaa. Puutteet ovat varsin minimaalisia – esimerkiksi pitäisi selvittää miten luotuja salasanoja voisi kätevästi kopioida leikepöydälle ohjelman ikkunasta.

Olin aluksi hyvin epävarma siitä, kuinka kauan aikaa graafisen käyttöliittymän kirjoittaminen veisi, joten olin varannut sille yhden iteraation verran aikaa. Tämä arvio osuikin varsin nappiin, mutta näin jälkikäteen ajatellen käytin ehkä turhan paljon aikaa ensimmäisen kahden iteraation aikana tekstipohjaisen käyttöliittymän muokkaamiseen. Tämä käyttöliittymä jäi kokonaan pois kun graafinen käyttöliittymä tuli käyttöön.

Jos nyt aloittaisin tälläisen ohjelman kirjoittamisen, niin en tekisi ollenkaan sellaista tekstipohjaista dialogiin perustuvaa käyttöliittymää kuin aluksi tein. Sensijaan aluksi ohjelmaa käytetäisiin – kuten linux-ohjelmia usein käytetään – terminaalissa, antaen muuttujat parametreina ohjelmakomennolle. Nyt ohjelmaa ei voi ollenkaan käyttää tällä tavoin, mikä on harmi. Mikäli ohjelman käyttäminen tällä tavoin olisi mahdollista, voisi sen generoimia salasanoja esimerkiksi putkittaa toisen ohjelman käyttöön.

Tämän terminaalipohjaisen käyttöliittymän lisäksi on myös monia muita tapoja, joilla ohjelman toiminnallisuutta voisi laajentaa. Esimerkiksi sanojen väliin tulevia välimerkkejä tulisi voida käyttää helpommin – käyttäjä voisi esimerkiksi valita erilaisista välimerkkilistoista hänelle parhaiten sopivat. Yksi merkitsevä tekijä tässä on se, kuinka helppo kukin merkki on kirjoittaa erilaisten puhelinten käyttöjärjestelminen näppäimistöillä. Myös sallittujen merkkien listaa salasanoissa voisi mahdollisesti laajentaa; nyt ne on rajoitettu vain perus-ASCII-merkkeihin, mutta todennäköisesti myös joitain muita merkkejä sallitaan yleisesti salasanoissa.

Alla työn edistyminen viikoittain:

5.1 Ensimmäinen viikko

Aloitin kirjoittamalla luokat sekä data access objectit ohjelman käsitteille: sanoille ja sanalistoille Aluksi nämä olivat vain yksinkertaisia pohjia – toiminnallisuus oli vain sanojen ja sanalistojen kirjoittaminen tietokantaan, ja niiden lukeminen sieltä. Kirjoitin myös yksinkertaisen tekstikäyttöliittymän sekä service-luokan joka muuntaa käyttäjän antaman listan sopivaan muotoon ja kirjoittaa sen tietokantaan *DAO*-luokkaa käyttäen

Ohjelman vaatima tietorakenne aiheutti jonkin verran päänvaivaa. Sanoja tietokannassa on kahdenlaisia: tavallisia sanoja, joita käytetään salasanoissa sekä blacklist-sanoja, joita ei saa käyttää salasanoissa. Päätin järjestää tämän vaatiman logiikan tietokantaan, eli jokaisen tavallisen sanan kohdalle merkitään,

löytyykö se myös blacklist-listalta. Tämä tuotti jonkin verran työtä viikolla kaksi, sillä sanojen lisääminen vaati monenlaisia iteraatiokierroksia sanojen lisäämisen kohdalla.

- Tiistaina 26.2. - kaksi tuntia suunnittelua
- Keskiviikkona 27.2. - tunti suunnittelua ja kolme tuntia koodausta
- Torstaina 28.2. - tunti suunnittelua ja tunti koodausta
- Perjantaina 29.2. - tunti suunnittelua ja tunti koodausta
- Lauantaina 2.3. - kaksi tuntia suunnittelua ja tunti koodausta
- Sunnuntaina 3.3. - kolme tuntia koodausta

5.2 Toinen viikko

Viikolla kaksi kirjoitin koodin jonka avulla sanalistan tiedot näytetään käyttäjälle, sekä toteutin mahdollisuuden sanalistojen poistamiseen. Toteutin myös ohjelman varsinaisen toiminnallisuuden, eli salasanan generoinnin tekevän luokan. Tämä tosin oli vielä yksinkertainen versio, ja suuri osa valmiin ohjelman toiminnallisuudesta puuttui vielä – esimerkiksi salasanan pituus (käytetty sanojen määrä) sekä käytetyt olivat suoraan koodissa määritelty, kun tavoitteena on antaa käyttäjän päättää nämä asiat.

Näiden muutosten takia myös edellisellä viikolla kirjoitettuihin *DAO*-luokkiin täytyi luoda uusia laajoja metodeja sanalistoihin liittyvien tietojen hakemiseksi. Myös käyttöliittymää laajennettiin. Viikon lopulla hoidin ohjelman kommentoinnin kuntoon ja kirjoitin testejä sekä readme-tiedoston

- Tiistaina 5.3. - kolme tuntia koodausta
- Keskiviikkona 6.3. - kolme tuntia koodausta
- Perjantaina 8.3 - kaksi tuntia suunnittelua ja kaksi tuntia koodausta
- Lauantaina 9.3. - kaksi tuntia testausta
- Sunnuntaina 10.3. - kaksi tuntia koodausta, kaksi tuntia testausta ja kaksi tuntia dokumentointia

5.3 Kolmas viikko

Ohjelman muuttuessa yhä monimutkaisemmaksi päätin uudistaa sen pakkausrakennetta paremmin tarpeita vastaavaksi. Siirsin luokat niiden tarkoitusta vastaaviin pakkauksiin ja päivitin importit vastaavasti. Järkevöitin myös sanaluokkien rakennetta, ja otin käyttöön perinnän niiden välillä.

Lisäsin sanalistaan attribuutiksi sen luontiajan. Tämä on ollut minulle aikamoinen murheenkryyni aikaisemmilla ohjelmointikursseilla, johtuen Javan useista erilaisista kirjastoista ajan ja päivämäärän kuvaamiseen. Tästä syystä päätin käyttää unix-timestamppia ajan tallentamiseen, ja vasta kun päivämäärä esitetään käyttäjälle, se muutetaan ihmisen luettavaan muotoon.

Laajensin salasanan generointiluokkaa, ja lisäsin käyttäjälle sekä mahdollisuuden määrittää salasanan pituus (käytettävien sanojen määrän) sekä käytettävät välimerkit. Laajensin käyttöliittymää vastaavasti.

Kirjoitin myös jonkin verran lisää testejä, mutta näiden kanssa minulla oli ongelmia, etenkin *DAO*-luokkien testaamisessa.

- Tiistaina 12.3. - kaksi tuntia suunnittelua ja tunti koodausta
- Torstaina 14.3. - tunti koodausta
- Perjantaina 15.3. - kaksi tuntia testausta
- Lauantaina 16.2. - kaksi tuntia koodausta, kaksi tuntia testausta

5.4 Neljäs viikko

Kirjoitin tällä viikolla paljon testejä. Opettelin käyttämään *Mockito*-kirjastoa tietokantayhteyden "väärentämiseen" testaamista varten. Kirjaston avulla sain myös "injektoitua" tämän yhteyden DAO-luokkiin, jolloin pystyin kirjoittamaan järkeviä testejä niille.

Opiskelin tällä viikolla myös paljon UML:ää, opettelin käyttämään ArgoUML:ää ja tein pakkaus- sekä sekvenssidiagrammit ohjelmasta.

- Maanantaina18.3. - tunti dokumentointia
- Keskiviikkona 20.3. - tunti testausta
- Torstaina 21.3. - kaksi tuntia dokumentointia
- Perjantaina 22.3. - kaksi tuntia testausta
- Lauantaina 23.3. - kolme tuntia testausta
- Sunnuntaina 24.3. - kaksi tuntia testausta, kolme tuntia dokumentointia

5.5 Viides viikko

Aloin tällä viikolla – suunnitelman mukaisesti – toteuttaa graafista käyttöliittymää ohjelmalle. Tämän opettelu vei paljon aikaa, sillä en ollut aiemmin tehnyt gtaafisia käyttöliittymiä Javalla. Sain tällä viikolla toteutettua itse käyttöliittymän, mutta vain rajoitetulla toiminnallisuudella.

- Maanantaina 24.3. - tunti suunnitelua
- Keskiviikkona 27.3. - kaksi tuntia suunnittelua ja tunti koodausta
- Torstaina 28.3. - tunti koodausta

- Lauantaina 30.3. - kolme tuntia koodausta
- Sunnuntaina 31.3. - neljä tuntia koodausta

5.6 Kuudes viikko

Jatkoin graafisen käyttöliittymän toteuttamista. Toteutin sanalistojen näyttämisen sekä salasanan generoinnin gtaafisessa käyttöliittymässä. Tämä vaati myös joitain muutoksia palveluluokkien koodiin.

- Maanantaina 1.4. - tunti suunnittelua ja tunti dokumentointia
- Keskiviikkona 3.4. - tunti suunnittelua ja tunti koodausta
- Torstaina 4.4. - kolme tuntia koodausta
- Perjantaina 5.4. - tunti koodausta ja tunti testausta
- Lauantaina 6.4. - neljä tuntia koodausta
- Sunnuntaina 7.4. - viisi tuntia koodausta

5.7 Seitsemäs viikko

Jostain selittämättömästä syystä ohjelmani graafinen käyttöliittymä ei suostunut enää käynnistymään tällä viikolla. Pitkällisen tutkimisen jälkeen sain sen toimimaan, kun sain Spring-ohjelman kontekstin määriteltyä oikein. Loin tätä varten uuden palveluluokan.

Jatkoin tämän jälkeen gtaafisen käyttöliittymän toteuttamista. Lisäsin mahdollisuuden sanalistojen lisäämiseen sekä tietokannan alustamiseen.

- Maanantaina 8.4. - tunti dokumentointia
- Tiistaina 9.4. - tunti koodausta
- Keskiviikkona 10.4. - tunti koodausta
- Torstaina 11.4. - tunti dokumentointia
- Perjantaina 12.4. - kaksi tuntia koodausta
- Lauantaina 13.4. - kolme tuntia koodausta
- Sunnuntaina 14.4. - neljä tuntia koodausta

5.8 Kahdeksas viikko

Lisäsin tällä viikolla ohjelmaan mahdollisuuden ladata sanalistan tiedostosta. Tätä varten opettelin käyttämään *FileChooser*-kirjastoa. Paransin ohjelman erinäisiä merkkijonojen validointimetodeja, sillä niistä paljastui lukuisia rajatapauksia, joissa vääränlaisia merkkejä hyväksyttiin. Paransin ohjelman testausta merkittävästi, kun opin käyttämään *Mockiton Argumentcaptor*-työkalua. Kirjoitin myös laajan integraatiotestin, joka testaa koko salasanan generointiprosessin.

Lisäsin ohjelmaan exit-napin sekä erillisen Help-sivun, jossa on ohjeet ohjelman käyttöön, ja päivitin myös projektin readme-tiedoston.

- Tiistaina 16.4. - kaksi tuntia dokumentointia
- Torstaina 18.4. - tunti koodausta ja tunti testausta
- Perjantaina 19.4. - kaksi tuntia dokumentointia
- Lauantaina 20.4. - neljä tuntia koodausta
- Sunnuntaina 21.4. - kolme tuntia testausta

5.9 Yhdeksäs viikko

Yhdeksännellä viikolla lisäsin ohjelmaan salasanojen vahvuuden laskennan. Tämä vaati varsin syvällistä tutustumista informaatioteoriaan ja entropian käsitteeseen sen kontekstissa. Itse laskentalogiikan lisäksi tämä vaati myös jonkin verran laajempia muutoksia ohjelman koodiin: muutin salasanojen esittämisen ikkunassa tekstikentästä tauluun, johon tuli omat sarakkeet salasanalle sekä sen entropialle. Vaihdoin salasanojen generointiluokan palauttamaan merkkijonolistan sijaan listan uusia *Password*-olioita, jolle luonnollisesti loin myös oman luokan.

Toteutin tällä viikolla myös sanalistan talletuksen tiedostoon.

- Maanantaina 22.4. - tunti dokumentointia
- Tiistaina 23.4. - tunti dokumentointia
- Keskiviikkona 24.4. - kaksi tuntia testausta ja tunti dokumentointia
- Torstaina 25.4. - tunti koodausta ja tunti testausta
- Perjantaina 26.4. - kaksi tuntia dokumentointia
- Lauantaina 27.4. - neljä tuntia koodausta
- Sunnuntaina 281.4. - kaksi tuntia koodausta, tunti testausta ja kolme tuntia dokumentointia

5.10 Kymmenes viikko

Projektin viimeinen viikko oli varsin lyhyt. Tein ohjelmaan joitain korjauksia, yhtenä esimerkkinä testatessa löytämäni rajatapaus, jossa ohjelma saattaa mennä katkeamattomaan looppiin tietokannassa olevista sanoista tehtyjen väärien oletusten vuoksi. Laajensin testausta kattamaan joitain rajatapauksia. Pääasiassa aika tällä viikolla meni dokumentaatioon.

- Maanantaina 29.4. - tunti koodausta ja tunti dokumentointia
- Tiistaina 30.4. - tunti koodausta ja kaksi tuntia testausta
- Keskiviikkona 1.5 - neljä tuntia dokumentointia

6 LIITTEET

1. Javadoc löytyy ohjelmakoodin yhteydestä, kansiosta netbeans/target/apidocs
2. Readme.MD löytyy ohjelmakoodin yhteydestä kansiosta netbeans/README.md
3. Ajankäyttö-kirjanpito löytyy liitteet-kansiosta
4. Versionhallintatehtävästä vaaditut kuvankaappaukset löytyvät kansiosta liitteet/versionhallintaharjoitus
5. Testausharjoituksesta vaaditut kuvankaappaukset sekä koodi löytyvät kansiosta liitteet/testausharjoitus
6. Kirjoittamani vertaisarviot löytyvät liitteet-kansiosta