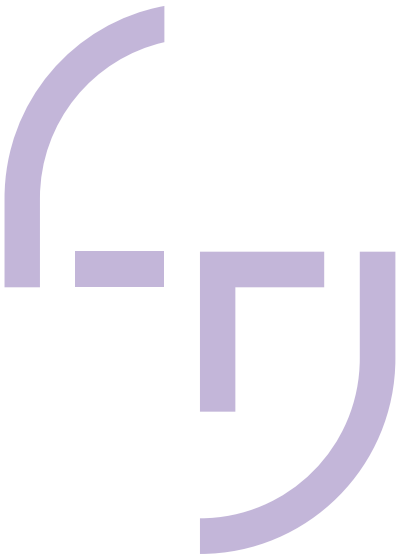
Tampereen ammattikorkeakoulu



Kirjastosovelluksen testilähtöinen kehitys

Artem Tolpa

OPINNÄYTETYÖ
Toukokuu 2023

Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

TOLPA, ARTEM:
Kirjastosovelluksen testilähtöinen kehitys

Opinnäytetyö 32 sivua, joista liitteitä 1 sivu
Toukokuu 2023

Opinnäytetyön tarkoituksena oli luoda kirjastosovellus testilähtöisen kehittämisen (TDD) avulla. Työssä tutustuttiin testilähtöisen kehittämisen perusperiaatteisiin ja niiden eroavaisuuksiin verrattuna perinteiseen sovelluskehittämiseen.

Työn tuloksena syntyi kirjastosovelluksena toimiva ohjelma. Sovellusta käytetään kirjojen lainaamisen hallinnointiin tilaajayrityksen eri toimipesteissä. Ohjelmalla on minimaalinen toimintokokonaisuus, jota voidaan tulevaisuudessa tarvittaessa laajentaa. Tällä hetkellä käyttäjät pystyvät varaamaan ja palauttamaan kirjoja sekä tarkistamaan kirjojen saatavuuden yrityksen eri toimipisteissä.

Sovelluksen kehittämisessä käytettiin Java-ohjelmointikieltä. Se on yksi maailman suosituimmista ohjelmointikielistä, koska se on universaalisti sovellettavissa. Java-kielen yksi tärkeimmistä eduista on sen riippumattomuus alustoista, minkä ansiosta sovelluksia voidaan ajaa eri käyttöjärjestelmissä eikä koodia tarvitse kirjoittaa uudelleen.

Kehittämällä sovellusta testilähtöisesti varmistettiin, että se toimii oikein jokaisessa kehitysvaiheessa. Lisäksi koodin laatua parannettiin testin selkeällä suunnittelulla ja TDD:n periaatteiden soveltamisella.

Asiasanat: testilähtöinen kehitys, Java-ohjelmointikieli, sovellus

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Software Development

TOLPA ARTEM:
Library Application's Test-driven Development

Bachelor's thesis 32 pages, appendices 1 page
May 2023

The purpose of this thesis was to create a library application through test-driven development (TDD). During the thesis, the basic principles of test-driven development and their differences from traditional application development were studied.

The result of the thesis is a functioning library application, which is used to manage borrowing books from different offices of the client company. The program has a minimal functionality that can be extended in the future if necessary. Currently, users can borrow and return books and check the availability of books at the company's offices in different cities.

The application was developed using the Java programming language. One of the main advantages of Java is its platform independence, which allows applications to run on different operating systems without rewriting code.

The test-driven development of the application ensured that the application works correctly at each stage of its development. A clear test design and the application of TDD principles also improved the quality of the code.

Key words: test-driven development, Java, application

SISÄLLYS

LYHENTEET JA TERMIT

TDD	Test-driven development, testivetoinen kehitys.
Siilibris	Toimeksiantajalle kehitettävä kirjastosovellus.
JVM	Java virtual machine. Virtuaalikone, joka suorittaa sille käännettyjä Java-ohjelmia.
API	Application programming interface. Rajapinta, joka mahdollistaa eri ohjelmistojen ja palveluiden välisen kommunikoinnin ja tiedonvaihdon.
JDK	Java Development Kit. Java-ohjelmointikielen kehitystyökalu, joka sisältää tarvittavat työkalut, kirjastot ja komponentit Java-sovellusten kehittämiseen ja suorittamiseen.
JPA	Java Persistence API. Java-ohjelmointikielen rajapinta tietokantakäsittelyyn, joka tarjoaa kehittäjille yksinkertaisen tavan hallita tietokantaa ja tallentaa tietoa Java-sovelluksissa.

1 JOHDANTO

Nykyään testaaminen on olennainen osa mitä tahansa ohjelmistokehitysprojektia. Testivetoinen kehitys (Test-Driven Development eli TDD) on lähestymistapa, jonka avulla voidaan luoda sovelluksia samalla jatkuvasti tarkistaen, että koodi toimii oikein. Toisin kuin perinteisessä lähestymistavassa ohjelmistokehitykseen, testivetoisessa kehityksessä testien kirjoittaminen tapahtuu ennen itse testattavan koodin kirjoittamista.

Tämän opinnäytetyön aiheena on tutustua testivetoisen kehittämisen periaatteisiin sekä soveltaa niitä käytännössä. Työn päämääränä on kehittää kirjastosovelluksen domain-osa Java-ohjelmointikielellä käyttäen TDD-lähestymistapaa.

2 TAUSTA

2.1 Toimeksiantajan esittely

Tämän työn toimeksiantaja on Siili Solutions Oyj, joka on vuonna 2005 perustettu ohjelmistoyritys. Yritys tarjoaa konsultointi- ja ohjelmistokehityspalveluita, ja sillä on toimistoja kahdeksassa Suomen kaupungissa sekä seitsemässä Euroopan ja Amerikan maassa.

2.2 Projektin esittely

Opinnäytetyön puitteissa suoritettavaksi toimeksiannoksi tilaaja ehdotti kirjastosovelluksen kehittämistä yrityksen sisäiseen käyttöön.

Yrityksen toimistoissa eri kaupungeissa on kirjahyllyjä, joista työntekijät voivat lainata kirjoja vapaasti. Hankalaksi lainaamisen tekee tällä hetkellä kuitenkin se, ettei siinä ole mitään seurantajärjestelmää. Tästä syntyikin idea sovelluksen kehittämisestä, joka toimisi slack-bottina ja antaisi työntekijöille mahdollisuuden tarkistaa kirjojen saatavuuden eri toimistoissa ja niiden kätevämmän lainaamisen.

2.3 Teknologiat ja menetelmät

2.3.1 Java

Projektissa käytettäväksi ohjelmointikieleksi valittiin Java.
Java on Sun Microsystemsin, – jonka on sittemmin ostanut Oracle, – vuonna 1995 julkaisema vahvasti tyypitetty oliopohjainen ohjelmointikieli.
Ohjelmointikielen syntaksi perustuu kieliin C ja C++. Yksi Javan suurimpia etuja on, että sen ohjelmistoalusta toimii eri laitteilla, esimerkiksi tietokoneella, mobiililaitteilla, pelikonsolilla ja niin edelleen. Näin ollen, kun koodi on kerran kirjoitettu, se on helppo siirtää toiselle laitetyypille.

Huolimatta uusien teknologioiden runsaudesta, Java on sen luomishetkestä lähtien pysynyt yli kahden vuosikymmenen ajan yhtenä maailman suosituimpana ohjelmointikielenä. (IBM n.d.)

Juuri oliopohjaisuutensa sekä laajan levinneisyytensä ansiosta tämä ohjelmointikieli sopii erinomaisesti projektissa asetettujen tavoitteiden ja tehtävien toteuttamiseksi, sillä projektin käsittämä aihealue edellyttää nimenomaan oliopohjaisen kehitysparadigman käyttöä.

2.3.2 Java Spring Framework

Java Spring Framework on suosittu viitekehys, jossa on avoin lähdekoodi autonomisten sovellusten kehittämiseen. Siihen sisältyy jo valmiita ratkaisuja tyypillisille tehtäville sovelluksissa, esimerkiksi tietojen muuntaminen, validointi, poikkeusten käsittely, resurssien hallinta ja niin edelleen. Tästä johtuen Spring Frameworkin avulla kehittäjät voivat kätevästi luoda monialustaisia sovelluksia, jotka toimivat käytännössä missä tahansa ympäristössä. (IBM n.d.)

2.3.3 Java Spring Boot

Java Spring Boot on työkalu verkkosovellusten, mikropalveluiden ym. luomiseen, joka tarjoaa suuren määrän valmiita työkaluja nopeaan autonomisten Spring-sovellusten kehittämiseen minimaalisella konfiguraatiolla ja säädöllä.

Spring Bootilla on seuraavat tärkeät ominaisuudet:

- Sovellusta alustettaessa siinä on jo valmiiksi asennettuja riippuvuuksia, joita ei tarvitse määrittää manuaalisesti.
- Spring Boot sisältää yli 50 Spring Starters -aloitusriippuvuutta, joihin kuuluu tarvittavat paketit riippuen projektin tarpeista. Käyttäjä määrittää itse projektinsa tarpeet alustusvaiheessa valitsemalla yhden Spring Starters -vaihtoehdon.
- Spring Bootin avulla voidaan luoda autonomisia sovelluksia, jotka toimivat itsenäisesti eivätkä vaadi ulkoista verkkopalvelinta. Tämä tapahtuu siten,

että Spring Boot upottaa oman verkkopalvelimensa sovellukseen alustusvaiheen aikana. Siten sovellus voidaan käynnistää millä tahansa alustalla antamalla Run-komento. (IBM n.d.)

2.3.4 IntelliJ IDEA

IntelliJ IDEA on JetBrains-yhtiön JVM-kielille kehittämä integroitu ohjelmointiympäristö (IDE, integrated development environment). Se sopii erinomaisesti seuraavilla kielillä kehittämiseen: Java, Kotlin, Scala ja Groovy. Lisäksi ohjelmointiympäristö tukee monia muitakin kieliä edellyttäen laajennusten asentamista. IntelliJ IDEA auttaa suorittamaan rutiininomaisia toistuvia toimintoja, muun muassa koodin älykkään täydennyksen sekä sen analyysin ja refaktoroinnin. (JetBrains n.d.)
Nämä ominaisuudet nopeuttavat huomattavasti koodin kirjoittamista, jolloin ohjelmoija voi paremmin keskittyä tehtävän varsinaiseen suorittamiseen suuntaamatta huomiotaan huolimattomuus- ja kirjoitusvirheisiin.

2.3.5 Testivetoinen kehitys

Sovellusta kehitettäessä on käytetty TDD-periaatetta (testivetoinen kehitys), jonka olemusta tarkastelemme seuraavaksi.

3 TESTIVETOINEN KEHITYS

3.1 Mikä on TDD?

TDD on lyhenne sanoista Test-Driven Development (testivetoinen kehittäminen), joka on ohjelmistokehitysmenetelmä, jossa testit kirjoitetaan ennen varsinaisen koodin kirjoittamista.

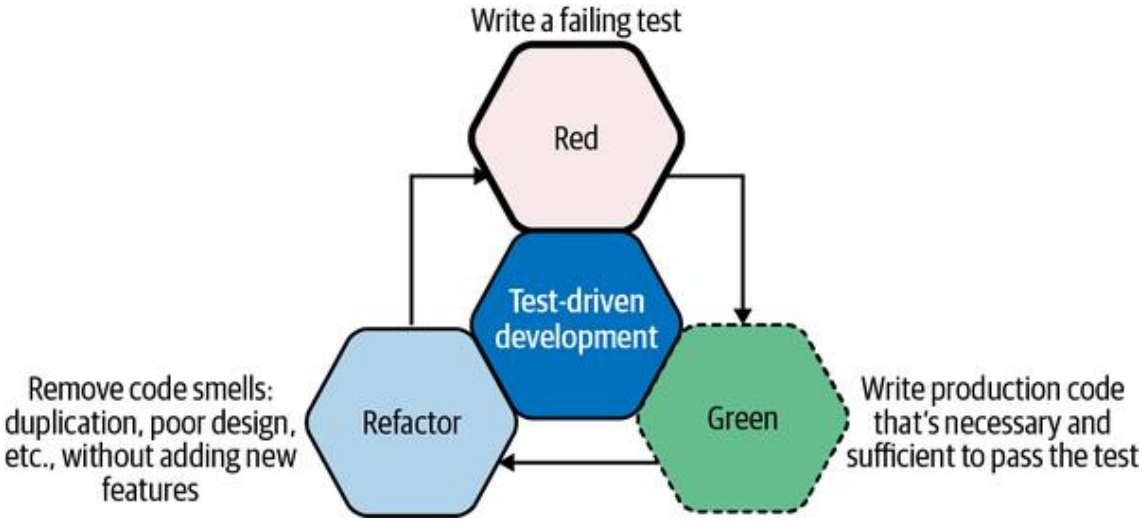
Testaaminen ei ole kuitenkaan TDD:n päätarkoitus. Sen päätarkoituksena on koodin suunnittelun ja rakenteen parantaminen testaamisen kautta. Jos TDD:n tarkoituksena olisi vain testata, olisi vaikeaa perustella testien kirjoittamisen välttämättömyys ennen varsinaisen testaamisen kohteena olevan koodin kirjoittamista. Siksi TDD on ainoastaan paremman ohjelmiston kehittämisen keino. (Siddiqui 2021.)

Millainen on hyvä testi? Hyvä testi testaa pienen atomisoidun palan haluttua ohjelman käyttäytymistä. Hyvä testi on eristettävä muista testeistä siten, että testi ei esimerkiksi oleta mitään aiemmin suoritetuista testeistä. (Koskela 2008.)

3.2 TDD:n vaiheet

Testilähtöinen kehitysprosessi koostuu kolmesta päävaiheesta:
1. Testin kirjoittaminen
2. Koodin kirjoittaminen
3. Refaktorointi

Tämä prosessi tunnetaan myös nimellä Red-Green-Refactor (RGF). Nimitys tulee niistä tiloista, jotka koodi käy läpi kehitysprosessinsa aikana. Punainen (Red) tila merkitsee toimimatonta koodia. Vihreä (Green) tila tarkoittaa, että koodi läpäisee testit onnistuneesti. Refaktoroinnin aikana käsitellään jo toimivaa koodia, joka läpäisee kaikki testit, joten sitä voidaan edelleen optimoida ja parantaa. (Garcia & Farcic 2018.)



KUVA 1. Red-Green-Refactor sykli (Siddiqui 2021.)

3.2.1 Testin suunnittelu ja toteutus

Testin kirjoittaminen ei tarkoita pelkästään sen kirjoittamista. Se sisältää myös koodin suunnittelun. Testin kirjoittaminen ennen itse koodin kirjoittamista pakottaa miettimään, miten koodin oikeasti pitäisi toimia.

On syytä huomata, ettei tulisi kirjoittaa massiivisia testejä, vaan riittää, että testi saadaan toimimaan oikein. Aluksi testin pitäisi epäonnistua, koska sen testaamaa koodia ei ole vielä olemassa. Pääsääntöisesti yhden testin kirjoittaminen ei pitäisi viedä enempää aikaa kuin pari minuuttia. Testien ansiosta voidaan ymmärtää, mitä toiminnallisuutta sovelluksessamme pitäisi olla juuri nyt. (Koskela 2008.)

3.2.2 Testattavan koodin kirjoittaminen

Testaamisen kautta tapahtuvaan kehitysprosessin seuraava vaihe on testattavan koodin kirjoittaminen. Kuten testien kirjoittamisenkin kohdalla, tässäkään vaiheessa ei kannata kirjoittaa suurikokoista koodia. Riittää, että kirjoittaa välttämättömän koodin siihen tarkoitukseen, että testit läpäistään onnistuneesti.

Yksi testilähtöisen kehittämisen pääideoista on, että testit osoittavat, missä järjestyksessä sovellusta tulisi kehittää. Koodin jokaisen osan on suunnattava siihen, että testit toteutuisivat onnistuneesti. Läpäistyjen testien ansiosta voidaan myös helposti seurata etenemistä työssä.

Toisinaan käy niin, että vaikka on kirjoitettu testin läpäisemiseen riittävä koodi, ratkaisu ei ehkä ole kaikkein optimaalisin. Tämä on kuitenkin normaalia, ja asia voidaan hoitaa seuraavassa refaktorointivaiheessa. Tärkeintä on, että oikeanlaiset testit auttavat olemaan menemättä harhaan kehitysprosessin aikana. (Koskela 2008.)

3.2.3 Refaktorointi

TDD-syklin viimeinen vaihe pitää sisällään sen, että mennään askel taaksepäin ja yritetään optimoida kirjoitettu koodi. Käytännössä melkein aina ensimmäistä versiota voidaan vielä parantaa. (Koskela 2008.)

Toisin kuin kaksi edellistä vaihetta tämä vaihe ei ole pakollinen, mutta erittäin suotava. Koska ei ole mitään tiukkoja sääntöjä, joiden mukaan refaktorointi tulisi tehdä jossain tietyssä kohdassa, kehittäjä päättää itse, milloin se on tehtävä. (Garcia & Farcic 2018.)

Yleensä refaktorointia tarvitaan seuraavissa tapauksissa:

- koodin osat ovat epäloogisessa järjestyksessä,
- koodin osia on kahdentunut,
- metodit tai luokat ovat liian pitkiä,
- muuttujat, metodit, luokat ym. on nimetty epäselkeästi. (Garcia & Farcic 2018.)

3.3 Miksi TDD?

Testivetoinen kehittäminen pakottaa miettimään jo ennen koodin kirjoittamista, miten koodia tullaan käyttämään, ja muutokset voidaan tehdä siinä vaiheessa,

jolloin se on helpointa. Lisäksi se auttaa refaktoroimaan koodi ja muuttamaan sen rakennetta siten, että samalla säilyy varmuus siitä, ettei sen toiminnallisuutta olla vahingoitettu. (Mellor 2023.)

TDD mahdollistaa sen, että kirjoitetaan vain välttämätön koodin minimimäärä ja ettei lisätä turhaa koodia sinne, missä sitä ei tarvita. Tätä mallia noudattamalla kehittäjä vapautuu mahdollisista ylimääräisistä vaikeuksista, joita voi syntyä kehitysprosessin eri vaiheissa. (Siddiqui 2021.)

Ajan mittaan muodostuva testipaketti suojaa kehittäjää ohjelman kaatumisilta, jotka aiheutuvat uusista toiminnallisuuksista, sillä testit kirjoitetaan ennen itse toiminnallisuuksien toteutusta. Tämän ansiosta koodin koon kasvaessa paranee myös sen laatu ja kehittäjän varmuus siitä. (Siddiqui 2021.)

4 SIILIBRIS

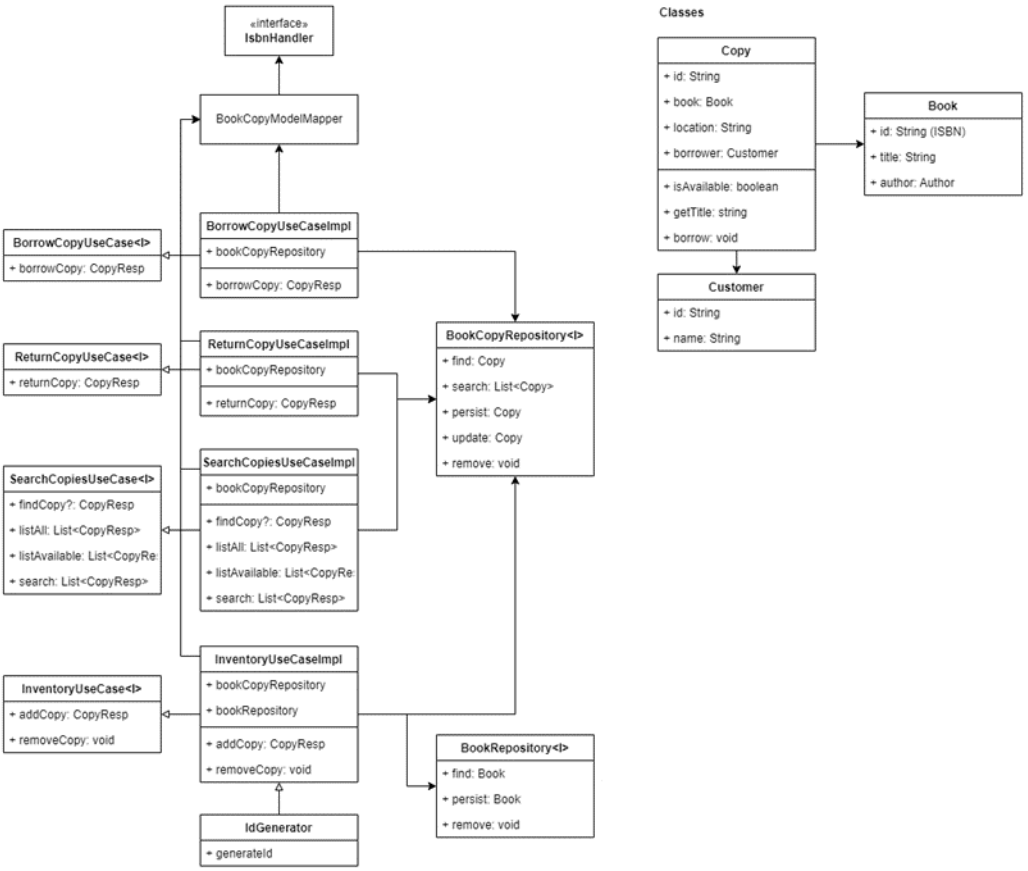
Keskusteltuamme toimeksiantajan kanssa sovelluksen yksityiskohdista päädyimme siihen, että minun tehtäväkseni tulisi domain-osan eli sovelluksen päälogiikan kirjoittaminen. Toimeksiantajan tehtäväksi jäisi loppuosan kehittäminen eli sovelluksen käyttöönotto AWS-pilvipalveluissa sekä slack-botin toteuttaminen. Jako perustuu osittain siihen, että harjoittelijoille on rajoitettu pääsy eräisiin yrityksen sisäisiin palveluihin.

4.1 Sovelluksen arkkitehtuuri

Koko aihealue jaettiin luokkiin, rajapintoihin (interfaceihin) ja niiden implementaatioihin.

Sovelluksen arkkitehtuuri on suunniteltu siten, että sovelluksen liiketoimintalogiikka, tietokanta sekä tietokantayhteydet on sijoitettu eri tasoille (Liite 1). Toisin sanoen, ne on erotettu toisistaan. Tämä mahdollistaa sen, että sovelluksen loogisia osia voidaan kehittää erillään toisistaan, eli voidaan kirjoittaa itsenäisiä koodilohkoja ja muuttaa niitä siten, ettei se vaikuta muihin osiin.

Alla on kaavio sovelluksen domain-osan arkkitehtuurista.



KUVA 2. Sovelluksen domain-osan rakenne

Kaikkiaan ohjelma sisältää seuraavat luokat ja rajapinnat (interfacet).

4.1.1 Luokat

- **Book**

  Luokka sisältää kentät id: String (id:nä käytetään isbn-kirjannumeroa), authors: String, title: String sekä get-metodit niille.

- **Copy**

  Luokka sisältää kentät id: String, book: Book, location: Location, borrowerId: String sekä set- ja get-metodit niille.
  Location kentälle tyyppinä käytetään Location enumia, joka sisältää listan kaupungeista, joissa sijaitsee Siilin toimistoja.

- **CopyRequest**

  Luokka sisältää kentät id: String, isbn: String, location: Location sekä set- ja get-metodit niille.
  Kyseistä luokkaa käytetään request-pyynnön luomisessa kirjakopiota varattaessa.

- **CopyResponse**

  Luokka sisältää kentät id: String, location: Location, title: String, authors: String, bookId: String, borrowerId: String, status: Status (kopion tila: vapaa vai lainassa), responseStatus: responseStatus (fail tai success) sekä set- ja get-metodit niille.
  Kyseistä luokkaa käytetään palvelimen muodostaessa vastausta kirjakopiota varattaessa ja palautettaessa.

- **CopyMapper**

  Luokka sisältää kentän isbnHandler: isbnHandler sekä mapToCopyResponse- ja mapToCopy-metodit.
  MapToCopyResponse-metodi ottaa parametriksi Copy-tyyppisen objektin ja sen pohjalta luo ja palauttaa uuden CopyResponse-tyyppisen objektin.
  MapToCopy-metodi ottaa parametriksi CopyRequest-tyyppisen objektin ja sen pohjalta luo ja palauttaa uuden Copy-tyyppisen objektin.
  isbnHandlerin tarkoitusta käsitellään myöhemmin.

- **LibraryInventoryController**

  Luokka sisältää POST- ja GET-metodit, joita tarvitaan routingissa eli sovelluksen sisällä tapahtuvassa reitityksessä.

4.1.2 Rajapinnat (interfacet) ja niiden implementaatiot

- **BookRepository**

  Rajapinta sisältää persist- ja findByIsbn-metodit.
  Persist-metodin avulla lisätään uusi kirja arkistoon.
  FindByIsbn-metodi löytää kirjan arkistosta annetulla isbn-numerolla.

- **BookCopyRepository**

  Rajapinta sisältää metodit persist, update, removeCopy, findById, findByLocation ja search.
  search-metodin avulla voidaan löytää kirjakopio tietokannasta sen nimellä tai sen tekijän nimellä.
  Muiden metodien funktiot (käyttötarkoitukset) voidaan helposti päätellä niiden nimityksistä.

- **InventoryUseCase**

  Rajapinta sisältää metodit addCopy, removeCopy, listCopies, listAvailable ja search.
  listCopies ja listAvailable palauttavat listat kaikista kopioista sekä lainattavissa olevista kopioista.
  Muiden metodien funktiot voidaan helposti päätellä niiden nimityksistä.

- **IsbnHandler**

  Rajapinta sisältää getBookByIsbn-metodin, joka ottaa parametriksi ISBN-numeron, minkä jälkeen lähetetään pyyntö palvelimelle. Mikäli syötetty ISBN-numero on validi, luodaan ja palautetaan Book-tyyppinen objekti, jossa on sekä kirjan että tekijän nimi. Tätä varten ohjelmointirajapintana (API) käytetään Google Books API -rajapintaa.

- **BorrowCopyUseCase**

  Rajapinta sisältää borrow-metodin, joka ottaa parametreiksi userId:n ja copyId:n. Borrow-metodi palauttaa CopyResponse-tyyppisen objektin, jossa

kyseiselle kopiolle annetaan käyttäjätunniste (userId) borrowerId-kenttänä.

- **ReturnCopyUseCase**

  Rajapinta sisältää returnCopy-metodin, joka ottaa parametreiksi userId:n ja copyId:n. ReturnCopy-metodi palauttaa CopyResponse-tyyppisen objektin, jossa kyseinen kopio merkitään vapaaksi ja borrowerId-kentälle annetaan null-arvo.

- **IdGenerator**

  Luokka sisältää generateId-metodin, joka luo ja palauttaa satunnaisen aakkosnumeerisen merkkijonon.

On syytä korostaa, että rajapinnoilla BookRepository ja BookCopyRepository on kaksi eri implementaatiota: BookRepositoryInMemory ja BookRepositoryJPA sekä vastaavasti BookCopyRepositoryInMemory ja BookCopyRepositoryJPA.

In-memory-implementaatiot tallentavat tietoja muistiin sovelluksen ollessa käynnissä ja niitä tarvitaan ennen kaikkea yksikkötestausta varten ja sovelluksen kehittämisen helpottamiseksi. JPA-implementaatioita (Java Persistence API) tarvitaan suoraan yhteydessä tietokantaan. Tämänkaltainen jako perustuu useaan tekijään:

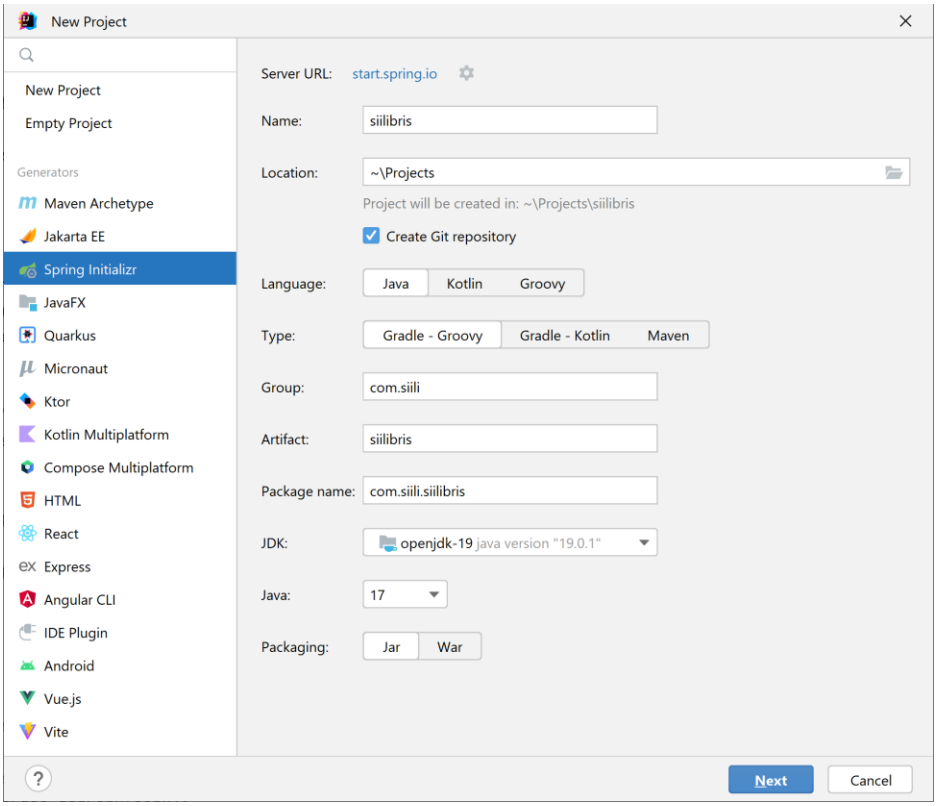
- Riippuvuudet
  Mikäli testit ovat riippuvaisia tietokannasta, niin kaavaa tai tietokannan asetusta muutettaessa voi syntyä ongelmia ja testit voivat lakata toimimasta.
- Testien nopeus
  Tietokantapyyntöjen suorittaminen voi hidastaa testien suorittamisnopeutta, etenkin jos pyyntöjä on paljon.
- Ennalta-arvaamattomuus

Tietokantapyyntöjen tulokset voivat muuttua riippuen tietokannan tilasta kyseisellä ajan hetkellä, ja tästä johtuen testeistä tulee vähemmän ennakoitavia.

Koska sovelluksesta tuli melko yksinkertainen ja tavoitteena oli kehittää nimenomaan sovelluksen domain-osaa, testauksessa käytettiin vain yksikkötestejä.

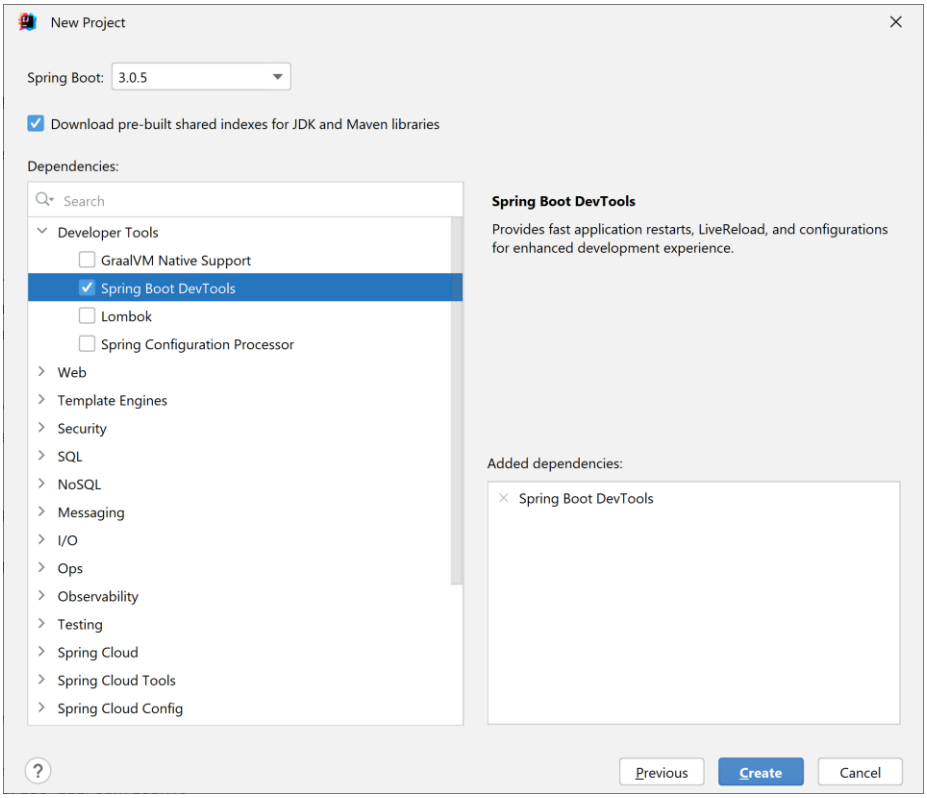
4.2 Uuden projektin luominen

On monta tapaa luoda uusi Java Spring -projekti. Käytännöllisin niistä lienee luoda se suoraan IntelliJIDEA:ssa.
Uuden projektin luomisikkunassa valittiin Spring Initializr. Seuraavaksi täytettiin projektin nimi, sijainti sekä tarvittavat JDK- ja Java-versiot.



KUVA 3. Uuden projektin luominen IntelliJ IDEA:ssa

Seuraavassa vaiheessa täytyi valita välttämättömät riippuvuudet. Tässä tapauksessa Spring Boot DevTools oli riittävä.



KUVA 4. Uuden projektin luominen IntelliJ IDEA:ssa

Tämän jälkeen IDE luo itse projektin, jolla on Spring-sovelluksille tyypillinen rakenne. Samalla luodaan erilliset paketit pääkoodille ja testeille.
Seuraavaksi tarkastellaan sovelluksen kehitysprosessia konkreettisen esimerkin kautta.

4.3 Esimerkki kehitysprosessista: uuden kirjakopion lisääminen

4.3.1 Testin kirjoittaminen

On loogista aloittaa ohjelman testaaminen perustoimintojen tarkistuksesta. Uuden kirjan lisäämisen logiikka suunniteltiin siten, että uusi kirja lisätään tietokantaan sillä hetkellä, kun järjestelmään lisätään kyseisen kirjan ensimmäinen kopio. Tässä tapauksessa on hyvä tarkistaa, että arkistoon voidaan ylipäänsä lisätä uusi kirjakopio.

Tältä näyttää testi, jossa halutaan lisätä uusi kirjakopio.

```
@Test
void testAddCopy() {
    var inventoryUseCase = new InventoryUseCaseImpl(new BookCopyRepositoryInMemory(),
            new BookRepositoryInMemory(),
            new CopyMapper(new IsbnServiceInMemory()),
            new IdGeneratorImpl());

    var copy = new CopyRequest( isbn: "9780132350884", Location.TAMPERE);
    var copyResponse = inventoryUseCase.addCopy(copy);

    Assertions.assertEquals( expected: "Clean Code", copyResponse.getTitle());
    Assertions.assertNotNull(copyResponse.getId());
    Assertions.assertEquals( expected: "Robert C. Martin", copyResponse.getAuthors());
    Assertions.assertNotNull(copyResponse.getBookId());
}
```

KUVA 4. testAddCopy testi, jossa tarkistetaan, että voidaan lisätä uusi kirjakopio

Testissä halutaan tarkistaa, että sekä itse kirja että sen kopio ovat tallentuneet. Samalla tarkistetaan, että kopio on tallentunut oikeine teoksen ja tekijän nimineen.

Testin ensimmäisellä käynnistyksellä se epäonnistuu, koska sillä hetkellä sovelluksen testattavana oleva toiminnallisuus ei ollut vielä toteutettu.
Jotta testi onnistuisi, täytyy toteuttaa seuraavat luokat.

4.3.2 Copy

```
public class Copy {

    private String id;
    private Book book;
    private Location location;
    private String borrowerId;

    public Copy(String title, String authors, String isbn, Location location) {
        this.book = new Book(title, authors, isbn);
        this.location = location;
    }

    public Copy(String id, Copy copy) {
        this.id = id;
        this.book = copy.getBook();
        this.location = copy.getLocation();
    }
```

KUVA 5. Copy-luokka

Tämä luokka edustaa kirjakopiomallia. Jokaisella kopio-objektilla on oltava kirja (book), jonka kopio se on, tunniste (id) ja sijainti (location), jossa kyseinen kopio sijaitsee. Luokalla on kaksi konstruktoria, joiden avulla voidaan luoda Copy-tyyppisiä objekteja erilaisine parametreineen. Lisäksi tämä luokka, kuten kaikki seuraavatkin, sisältää näyttökuvasta piiloon jäävät get- ja set-metodit kaikille luokan kentille.

4.3.3 Book

```java
public class Book {

    private String id;
    private String title;
    private String authors;

    public Book(String title, String authors, String id) {
        this.title = title;
        this.authors = authors;
        this.id = id;
    }

    public Book(Book book) {
        this.id = book.getId();
        this.title = book.getTitle();
        this.authors = book.getAuthors();
    }
```

KUVA 6. Book-luokka

Tämä luokka on kirjaobjektin luomista varten. Se sisältää kaksi konstruktoria, joiden avulla voidaan luoda tämän luokan kappaleita erilaisine parametreineen. Yhtä konstruktoreista käytetään jo olemassa olevan Book-objektin kopioimiseen.

4.3.4 BookCopyRepositoryInMemory

```java
public class BookCopyRepositoryInMemory implements BookCopyRepository {

    private final Map<String, Copy> copyMap = new HashMap<>();

    @Override
    public Copy persist(Copy copy) {
        if (copyMap.containsKey(copy.getId())) {
            throw new SaveException("Id " + copy.getId() + " is already exists!");
        }
        var savedCopy = new Copy(copy.getId(), copy);
        copyMap.put(savedCopy.getId(), savedCopy);
        return savedCopy;
    }
```

KUVA 6. BookCopyRepositoryInMemory-luokka

Tämä luokka on arkisto, jota käytetään kirjakopioiden tallentamiseen ja hakemiseen muistista sovelluksen ollessa käynnissä. Luokkaa tarvitaan vain testaamiseen. Se sisältää kaikki samat metodit, kuin luokka BookCopyJPA, jossa tapahtuu yhteys tietokantaan.

Luokassa luodaan copyMap-kenttä, jota tarvitaan objektien tallentamiseen. persist-metodin avulla voidaan tallentaa uusia kopioita arkistoon. Metodi tarkistaa, onko kyseistä objektia jo olemassa taulukossa samoine id-tunnisteineen. Jos kopio on jo olemassa, metodi luo poikkeuksen (exception).

4.3.5 BookRepositoryInMemory

```
public class BookRepositoryInMemory implements BookRepository {

    private final Map<String, Book> bookMap = new HashMap<>();

    @Override
    public Optional<Book> findByIsbn(String isbn) {
        return Optional.ofNullable(bookMap.get(isbn));
    }

    @Override
    public Book persist(Book book) {
        book = new Book(book);
        bookMap.put(book.getId(), book);

        return book;
    }
}
```

KUVA 7. BookCopyRepository-luokka

Tämä luokka toimii samalla periaatteella kuin BookCopyRepositoryInMemory sillä poikkeuksella, että se sisältää findByIsbn-metodin. Metodin avulla voidaan löytää kirja arkistosta annetulla ISBN-numerolla.

4.3.6 CopyRequest

```
public class CopyRequest {
    private String isbn;
    private Location location;

    public CopyRequest(String isbn, Location location) {
        this.isbn = isbn;
        this.location = location;
    }
```

KUVA 8. CopyRequest-luokka

Tätä luokkaa tarvitaan pyyntö- eli request-objektin luomiseen, kun yritetään lisätä uutta kirjakopiota.

4.3.7 CopyResponse

```
public class CopyResponse {
    private final String id;
    private Location location;
    private String title;
    private String authors;
    private String bookId;
    private String borrowerId;
    private Status status;
    private ResponseStatus responseStatus = ResponseStatus.SUCCESS;

    public CopyResponse(String id) {
        this.id = id;
    }
```

KUVA 9. CopyResponse-luokka

Tätä luokkaa käytetään vastaus- eli response-objektin luomisessa, kun yritetään lisätä uutta kirjakopiota. Se sisältää muun muassa kentät status (kirjan tila: lainassa vai vapaa) sekä responseStatus (vastauksen tila: onnistunut vai epäonnistunut).

4.3.8 CopyMapper

```
public class CopyMapper {
    final IsbnHandler isbnHandler;

    public CopyMapper(IsbnHandler isbnHandler) {
        this.isbnHandler = isbnHandler;
    }

    public CopyResponse mapToCopyResponse(Copy copy, ResponseStatus responseStatus) {
        var response = mapToCopyResponse(copy);
        response.setResponseStatus(responseStatus);
        return response;
    }

    public CopyResponse mapToCopyResponse(Copy copy) {
        var response = new CopyResponse(copy.getId());
        response.setTitle(copy.getBook().getTitle());
        response.setLocation(copy.getLocation());
        response.setAuthors(copy.getBook().getAuthors());
        response.setBookId(copy.getBook().getId());
        response.setBorrowerId(copy.getBorrowerId());
        response.setStatus(copy.getStatus());
        return response;
    }

    public Copy mapToCopy(CopyRequest copyRequest) {
        var book = isbnHandler.getBookByIsbn(copyRequest.getIsbn());
        return new Copy(book.getTitle(), book.getAuthors(), book.getIsbn(), copyRequest.getLocation());
    }
}
```

KUVA 10. CopyMapper-luokka

Kyseistä luokkaa tarvitaan yhden tyypin objektien muuntamiseksi toisen tyyppisiksi, esim. Copy → CopyResponse ja CopyRequest → Copy.
Luokkaa käytetään lisättäessä uutta kopiota arkistoon sekä vastaanotettaessa vastausta tämän toiminnon päätteeksi.

4.3.9 IsbnServiceInMemory

```
public class IsbnServiceInMemory implements IsbnHandler {

    private final Map<String, Book> bookList = new HashMap<>();


    private void createList() {
        bookList.put("9780132350884", new Book( title: "Clean Code",  authors: "Robert C. Martin",  id: "9780132350884"));
        bookList.put("9780134757599", new Book( title: "Refactoring ",  authors: "Martin Fowler",  id: "9780134757599"));
        bookList.put("9780201633610", new Book( title: "Design Patterns",  authors: "Richard Helm, Ralph Johnson, John Vlissides",  id: "9780201633610"));
        bookList.put("9780134494166", new Book( title: "Clean Architecture",  authors: "Robert C. Martin",  id: "9780134494166"));
    }

    @Override
    public Book getBookByIsbn(String isbn) {
        createList();
        return bookList.get(isbn);
    }
}
```

KUVA 11. IsbnServiceInMemory-luokka

Tämä on isbnHandler-rajapinnan implementaatio, joka säilyttää muistissa kirjataulukkoa, josta voidaan hakea kirjoja niiden ISBN-numerolla.
CreateList-metodi luo neljästä kirjasta koostuvan listan, ja getBookByIsbn-metodi palauttaa tästä listasta tietyn ISBN-numeron omaavan kirjan.
Luokkaa käytetään yksikkötesteissä. Lisäksi on isbnServiceGoogleAPI-luokka, joka suorittaa palvelinpyyntöjä ja käyttää Google Books API:a kirjojen hakemiseen.

4.3.10 IdGenerator

```
public class IdGeneratorImpl implements IdGenerator {

    @Override
    public String generateId() {
        return RandomStringUtils.randomAlphanumeric( count: 4).toUpperCase();
    }
}
```

KUVA 12. IdGenerator-luokka

Tämä on IdGenerator-rajapinnan implementaatio, jota tarvitaan uniikin id:n luomiseksi kirjakopiolle. Luokka sisältää generateId-metodin, joka palauttaa satunnaisen aakkosnumeerisen merkkijonon.

4.3.11 InventoryUseCaseImpl

```
public class InventoryUseCaseImpl implements InventoryUseCase {

    private final BookCopyRepository bookCopyRepository;
    private final BookRepository bookRepository;
    private final CopyMapper mapper;
    private final IdGenerator idGenerator;

    public InventoryUseCaseImpl(BookCopyRepository bookCopyRepository,
                                BookRepository bookRepository,
                                CopyMapper copyMapper,
                                IdGenerator idGenerator) {
        this.bookCopyRepository = bookCopyRepository;
        this.bookRepository = bookRepository;
        this.mapper = copyMapper;
        this.idGenerator = idGenerator;
    }

    @Override
    public CopyResponse addCopy(CopyRequest copyRequest) {
        var copy = mapper.mapToCopy(copyRequest);
        this.bookRepository.findByIsbn(copy.getBook().getIsbn())
                .ifPresentOrElse(copy::setBook, () -> copy.setBook(this.bookRepository.persist(copy.getBook())));
        copy.setId(this.idGenerator.generateId());
        return mapper.mapToCopyResponse(this.bookCopyRepository.persist(copy));
    }
```

KUVA 13. InventoryUseCaseImpl-luokka

Tätä luokkaa tarvitaan välineistön eli kirjojen, kirjakopioiden ja niiden arkistojen hallintaan. Luokka sisältää konstruktorin ja addCopy-metodin uuden kopion lisäämistä varten.

Alussa addCopy-metodi muuntaa copyRequest-objektin copy-objektiksi. Sen jälkeen metodi yrittää hakea kirjaa ISBN-numerolla bookRepository:sta. Jos kirja on jo arkistossa, metodi asettaa löydetyn kirjan copy-objektiin kutsuen setBook-metodin. Muuten metodi tallentaa uuden kirjan arkistoon, asettaa sen copy-objektiin ja kutsuu persist-metodin.
Kun kopio-objekti on määritetty, metodi asentaa sille uuden id:n ja tallentaa bookCopyRepository-arkistoon.
Lopuksi metodi muuntaa copy-objektin copyResponseksi ja palauttaa sen.

4.3.12 Tulos

Kun kaikki ylempänä luetellut luokat ja välttämättömät metodit on suoritettu, testAddCopy-testi onnistuu.

Ohjelman muut osat kehitettiin samalla periaatteella. Sovelluksen lopullisessa versiossa käyttäjä voi käyttää seuraavia toimintoja:

- Nähdä kaikki tietyn toimiston järjestelmässä olevat kirjat,
- Selata lainattavissa olevia kirjoja sekä niiden määriä tietyssä toimistossa,
- Hakea kirjoja teoksen tai tekijän nimellä,
- Lisätä uuden kirjan järjestelmään,
- Varata kirjan,
- Palauttaa kirjan.

5 POHDINTA

Työn yhteenvetona voidaan tehdä johtopäätös, että yleisesti ottaen testivetoinen ohjelmistokehittäminen on hyödyllinen ja tehokas menetelmä, joka mahdollistaa laadukkaampien ja luotettavampien sovellusten luomisen. Kyseistä lähestymistapaa käytettäessä on mahdollista testata koodia heti sen kirjoittamisen aikana, minkä ansiosta virheet voidaan löytää ja korjata heti niiden ilmestymisvaiheessa. Ohjelmistokehittäminen testivetoisesti vaatii kuitenkin kokemusta ja tiettyjä taitoja testien kirjoittamisessa, minkä vuoksi aloitteleville kehittäjille se voi olla hankalaa. Joka tapauksessa valinta TDD:n ja perinteisen lähestymistavan välillä riippuu projektin konkreettisista vaatimuksista ja tarpeista.

Tehtyäni tämän opinnäytetyön ja luotuani sovelluksen TDD-lähestymistapaa käyttäen voin todeta, että minulle ohjelmoijana tämän lähestymistavan soveltaminen tuntui aluksi vaikealta ja perusteettoman työläältä. Tämä kokemus johtui varmaankin kokemuksen puutteesta testien kirjoittamisessa. Välillä testien suunnitteluun ja kirjoittamiseen kului enemmän aikaa, kuin itse testattavan koodin kirjoittamiseen. Ajan myötä aloin kuitenkin paremmin ymmärtää tämän lähestymistavan periaatteita. Seurauksena projektin loppuun mennessä minulle tuli luonnolliseksi se, että testit pitäisi kirjoittaa ennen koodin kirjoittamista ja että niiden avulla tulisi tarkistaa koodin oikeanlainen toimivuus erilaisissa ohjelman käyttöskenaarioissa.

Työn tuloksena syntyi kirjastosovellus. On syytä huomata, ettei kyseessä ole vielä lopullinen versio ja ajan myötä siihen tullaan lisäämään uusia toimintoja. Valitettavasti sovelluksen koodi on Siili Solutionsin GitLabissa ja siihen on rajoitettu pääsy, minkä vuoksi sitä ei voida julkaista kokonaisena.

LÄHTEET

Farcic, V., & Garcia, A. 2018. Test-Driven Java Development, Second Edition: Invoke TDD Principles for End-To-end Application Development, 2nd Edition. Packt Publishing, Limited. Viitattu 30.3.2023. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/test-driven-java-development/9781788836111/

Koskela, L. 2008. Test driven: practical TDD and acceptance TDD for Java developers (1st edition). Manning Publications. Viitattu 30.3.2023. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/test-driven-practical/9781932394856/

Mellor, A. 2023. Test-driven development with Java: create higher-quality software by writing tests first with solid and hexagonal architecture (1st ed.). Packt Publishing, Limited. Viitattu 30.3.2023. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/test-driven-development-with/9781803236230/

Siddiqui, S. 2021. Learning Test-Driven Development. O'Reilly Media, Inc. Viitattu 30.3.2023. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/learning-test-driven-development/9781098106461/

IntelliJ IDEA overview. 2023. JetBrains. Verkkosivu. Viitattu 20.2.2023 https://www.jetbrains.com/help/idea/discover-intellij-idea.html

What is Java? n.d. IBM. Verkkosivu. Viitattu 20.2.2023 https://www.ibm.com/topics/java

What is Java Spring Boot? n.d. IBM. Verkkosivu. Viitattu 20.2.2023 https://www.ibm.com/topics/java-spring-boot

LIITTEET

Liite 1. Sovelluksen rakenne