Opinnäytetyö (AMK)

Tietojenkäsittelyn koulutus

2024

Johannes Natunen

Korkeakouluhakijoiden SAT-pistetietojen välittäminen College Boardista Opintopolkuun ohjelmointirajapintoja käyttäen

– sovelluksen suunnittelutyö

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tietojenkäsittelyn koulutus

2024 | 40 sivua

Johannes Natunen

Korkeakouluhakijoiden SAT-pistetietojen välittäminen College Boardista Opintopolkuun ohjelmointirajapintoja käyttäen

- sovelluksen suunnittelutyö

Suomen korkeakoulujen yhteishaussa on useita tapoja hakea korkeakoulusta opiskelupaikkaa. Perinteisten todistusvalinnan tai valintakokeen perusteella tehtävän haun lisäksi joihinkin koulutuksiin voi hakea SAT-kokeiden avulla. SAT on College Boardin järjestämä standardisoitu tapa mitata hakijoiden kelpoisuutta korkeakouluopintoihin.

Turun ammattikorkeakoulun hakijapalveluilla oli tarve saada SAT-pisteiden vienti Opintopolkuun automatisoitua, sillä aiemmin tämä prosessi hoidettiin käsin. Tämän opinnäytetyön tavoitteena oli suunnitella hakijapalveluille sovellus, joka hakee College Boardin järjestelmästä korkeakouluhakijan SAT-pistetiedot, jotka välitetään Opintopolun järjestelmään ohjelmointirajapintoja käyttäen.

Työ alkaa vaatimusmäärittelystä, jonka jälkeen itse sovellusta lähdetään suunnittelemaan. Sovelluksen prosessien kulku käydään läpi vaiheittain. Tämän jälkeen käydään läpi suunnitelma sovelluksen toteuttamiseen, missä pohditaan teknisiä seikkoja sovelluksen kehittämisen suhteen. Sovellukselle luodaan myös testitapauksia vaatimusmäärittelyn perusteella. Opinnäytetyön lopputuloksena on sovelluksen valmis suunnitelma, jonka pohjalta sovellusta aletaan kehittämään.

Asiasanat:

Opintopolku, College Board, rajapinta, sovelluskehitys, integraatiokehitys

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Business Information Technology

2024 | 40 pages

Johannes Natunen

Transferring Applicants’ SAT Scores from College Board to Studyinfo by Utilizing Application Programming Interfaces

- An application design process

There are several ways to apply for a place at a higher education institution in the Finnish joint application procedure. In addition to the traditional applications based on matriculation examinations or an entrance examination, one can also apply for some degrees with SAT test results. The SAT is a standardized way to measure applicants’ eligibility for higher education that is organized by the College Board.

The Admission Services of Turku University of Applied Sciences had a need for the automatization of transferring applicants’ SAT scores to Studyinfo (Opintopolku) as the process was previously handled manually. Studyinfo is the official portal for education-related information and services in Finland. The objective of this thesis was to design an application that retrieves the SAT scores of an applicant from the College Board and then submits the scores to Opintopolku system by utilizing APIs.

The methodology of this thesis began with the creation of a requirements specification, after which the design process began. A plan for the implementation of the application was created after the the design process. In addition, test cases for the application were created based on the requirements specification. The result of the thesis was the complete design of the application.

Keywords:

Opintopolku, College Board, API, software development, integration development

Sisältö

Liitteet

Kuvat

Taulukot

Käytetyt lyhenteet

API	Application Programming Interface
BSON	Binary JSON
CAS	Central Authentication Service
CSRF	Cross-site Request Forgery
CSV	Comma-separated Values
GDPR	General Data Protection Regulation
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
JSX	JavaScript Syntax Extension
REST	Representational State Transfer
TGT	Ticket Granting Ticket

1 Johdanto

Tämän opinnäytetyön tavoitteena on suunnitella sovellus, joka hakee arvosanoja SAT-kokeita järjestävästä College Boardista korkeakouluhakijoiden koetuloksia, ja vie ne Suomen yhteishakujärjestelmä Opintopolkuun.

Opinnäytetyön toimeksiantaja on Turun ammattikorkeakoulun hakijapalvelut. Hakijapalveluiden töihin kuuluvat kaikki tutkintoon johtavan koulutuksen hakeutumiseen, hakuprosessiin ja valintaperusteisiin liittyvät asiat. Hakijapalvelut tarjoaa myös tietoa ja tukea hakijoille, vastaa kysymyksiin ja auttaa hakijoita hakemisen eri vaiheissa. Lisäksi hakijapalvelut huolehtii siitä, että ammattikorkeakoulu noudattaa opiskelijavalintaan liittyvää lainsäädäntöä.

Toimeksiantajalla on tarve järjestelmälle, sillä entuudestaan toimeksiantaja on tehnyt tätä prosessia manuaalisesti. Hakijoiden alkutiedot kerätään Excel-taulukkoon, johon lisätään hakijoiden pistetiedot College Boardista yksi kerrallaan, minkä jälkeen pisteet syötetään manuaalisesti Opintopolun järjestelmään. Opinnäytetyössä suunniteltava sovellus nopeuttaisi prosessia merkittävästi. Ihanteellisessa tapauksessa käyttäjän tarvitsisi vain syöttää sovellukselle lista hakijoista, minkä jälkeen sovellus hakee automaattisesti College Boardista hakijan SAT-arvosanan ja syöttää sen Opintopolkuun.

Opinnäytetyön tavoitteena on tehdä sovelluksesta kattava suunnitelma. Ensin tehdään vaatimusmäärittely, johon on lueteltu toimeksiantajan ennalta antamat vaatimukset. Tämän lisäksi vaatimukset priorisoidaan. Vaatimusmäärittelyn jälkeen lähdetään suunnittelemaan sovellusta. Suunnitteluvaiheessa tutustutaan College Boardin ja Opintopolun järjestelmiin ja ohjelmointirajapintoihin sekä käydään läpi sovelluksen toiminta vaihe vaiheelta. Lisäksi luvussa 3.3 pohditaan, miten yhden hakijan College Board- ja Opintopolkuprofiilit saadaan yhdistettyä toisiinsa. Suunnitteluvaiheen jälkeen käydään läpi pikaisesti, miten sovelluksen käytännön toteutus tulee toimimaan. Tässä luvussa määritetään sovelluksen kehitykseen käytetyt teknologiat, suunnitellaan toimintojen kehitysjärjestys ja selvitetään sovelluksen julkaisutapa sen valmistuessa. Lopuksi sovellukselle luodaan testitapauksia aiemmin laaditun vaatimusmäärittelyn perusteella.

2 Vaatimusmäärittely

Tässä luvussa käydään läpi työn kannalta keskeiset vaatimukset. Toimeksiantajalta saatujen alustavien tietojen mukaan on hyvä tehdä vaatimusmäärittely, jossa sovelluksen vaatimukset ovat selkeästi listattuna. Vaatimusmäärittely auttaa havainnollistamaan sovelluksen tärkeimmät tehtävät ja samalla erittelee ne, tehden jokaisen osan suunnittelun helpommaksi. Vaatimusmäärittelyssä on myös mahdollista priorisoida eri ominaisuuksia. Sovelluksen kehittäjä voi täten nähdä vaatimusmäärittelystä, mitkä vaiheet kehityksestä ovat tärkeimpiä. Hyvän vaatimusmäärittelyn tekeminen vähentää mahdollisia väärinkäsityksiä toimeksiantajan ja kehittäjän välillä. Vaatimukset prioriteetteineen taulukoitiin. (Taulukko 1.)

Taulukko 1. Vaatimukset priorisoituna.

VAATIMUS	**PRIORISAATIO**
Sovelluksen täytyy pystyä noutamaan College Boardin rajapinnoista hakijan pistetiedot.	**Must have**
Sovelluksen täytyy pystyä yhdistämään hakijan College Board-tunnus oikeaan Opintopolkutunnukseen.	**Must have**
Sovelluksen täytyy pystyä lähettämään Opintopolkuun hakijan pistetiedot, jotka saadaan College Boardista.	**Must have**
Sovelluksen täytyy pystyä tallentamaan jokaisen hakijan status tietokantaan, jotta tiedetään, onko tietojen haku ja lähetys onnistunut tietyn hakijan kohdalla.	**Must have**
Sovelluksen täytyy pystyä tekemään loppuraportti (esim. txt-tiedostomuodossa) tuloksista, kuten onnistumisista ja epäonnistumisista, ja antaa se käyttäjän tarkasteltavaksi.	**Should have**
Sovellukseen täytyy pystyä tekemään sujuva sisäänkirjautumisjärjestelmä, jotta ulkopuoliset tahot eivät pääse sovellukseen käsiksi.	**Must have**
Sovelluksessa täytyy olla helposti ymmärrettävä käyttöliittymä.	**Should have**
Sovellusta täytyy pystyä käyttämään sekä suomeksi että englanniksi.	**Could have**

Sovelluksen täytyy pystyä tekemään raportteja onnistuneista tulosten vienneistä (missä näkyisi haun nimi, hakukohteen nimi, hakijoiden nimet, Opintopolkutunnukset ja heidän pisteensä).	**Could have**

Vaatimusmäärittelyä lähdettiin tekemään sekä toimeksiantajan tekemän opinnäytetyön aiheen kuvailun pohjalta että toimeksiantajan kanssa käytyjen keskustelujen pohjalta. Määrittelyssä esille tulleiden kohtien priorisaatio tehtiin MoSCoW-menetelmällä, jossa luokittelukategoriat ovat *must have, should have, could have* ja *will not have* (suomeksi käännettynä karkeasti *täytyy olla, pitäisi olla, voisi olla, ei tule olemaan*). MoSCoW-menetelmä on Oraclella työskennelleen Dai Cleggin kehittämä tapa luokitella kehitettävän tuotteen vaatimuksia toimeksiantajan, osakkeenomistajien ja kehittäjien välillä. *Must have* -luokituksen saaneet ominaisuudet ovat sellaisia, joiden olemassaolo on pakollista tuotteen (tässä tapauksessa sovelluksen) oikeanlaisen toiminnan kannalta. *Should have* -luokituksen saavat ominaisuudet, jotka ovat tärkeitä sovelluksen toiminnan kannalta, mutta joita ilman sovellus kuitenkin on käyttökelpoinen. *Could have* -luokituksen saaneet ominaisuudet olisivat hyödyllisiä, mutta valmistuessaan niillä ei olisi merkittävän suurta vaikutusta sovelluksen toimintaan. Nämä ominaisuudet toteutetaan tyypillisesti vain silloin, jos kaikki *must have-* ja *should have* -luokituksen saaneet ominaisuudet on saatu tehdyiksi. *Will not have* -luokituksen saavat sellaiset ominaisuudet, joiden kehittämiseen ei määritetä ollenkaan aikaa. Nämä ovat kuitenkin sellaisia ominaisuuksia, joihin saatetaan palata, kun sovelluksen seuraavaa versiota lähdetään kehittämään. Tämän takia monet kutsuvatkin luokitusta ennemmin nimellä *wish (toive)* tai *will not have this time (ei tule olemaan tällä kertaa)*. [1]

2.1 Priorisaatio ”must have”

Sovelluksen täytyy pystyä noutamaan College Boardin rajapinnoista hakijan pistetiedot. Yksi sovelluksen päävaatimuksista. Yksittäisen hakijan pistetiedot on haettu aiemmin College Boardista käyttöliittymän kautta. Uuden toteutuksen mukaan pistetiedot saataisiin suoraan sovellukseen College Boardin rajapinnoista, jonka takia käyttäjän ei tarvitsisi käydä College Boardissa hakemassa pistetietoja.

Sovelluksen täytyy pystyä yhdistämään hakijan College Board-tunnus oikeaan Opintopolkutunnukseen. Tämä on sovelluksen alkuperäinen idea ja sen suurin ongelmakohta: Opintopolun ja College Boardin välillä yksittäisellä hakijalla ei ole

mitään yhdistävää henkilökohtaista id:tä tai muuta samankaltaista, jolla hakijan pistetiedot saataisiin mutkattomasti vietyä. Tähän ongelmaan palataan sovelluksen suunnitteluvaiheessa.

Sovelluksen täytyy pystyä lähettämään Opintopolkuun hakijan pistetiedot, jotka saadaan College Boardista. Pistetietojen lähettäminen opintopolkuun on jälleen kerran olennainen osa sovelluksen kunnollista toimintaa. Yksittäisten hakijoiden pistetiedot on aiemmin viety Opintopolkuun käsin, hakija kerrallaan. Toimiva sovelluksen toteutus mahdollistaisi sen, ettei käyttäjän tarvitsisi enää käydä Opintopolun puolella lisäämässä pistetietoja käsin.

Sovelluksen täytyy pystyä tallentamaan jokaisen hakijan status tietokantaan, jotta tiedetään, onko tietojen haku ja lähetys onnistunut tietyn hakijan kohdalla. Tietokannan olemassaolo ei aluksi kuulunut suunnitelmiin, mutta toimeksiantajan kanssa keskustellessa ilmeni, että sen olemassaolo olisi lähes pakollista. Tietokannan avulla vältytään viemästä yhden hakijan tietoja Opintopolun järjestelmään useaan kertaan sekä saadaan tietää, onko hakijan tietojen haku tai vienti onnistunut.

Sovellukseen täytyy pystyä tekemään sujuva sisäänkirjautumisjärjestelmä, jotta ulkopuoliset tahot eivät pääse sovellukseen käsiksi. Sujuvan sisäänkirjautumisjärjestelmän avulla käyttäjän ei välttämättä tarvitse syöttää Opintopolun ja College Boardin tunnuksiaan jokaisella sovelluksen käyttökerralla, jos tunnukset olisivat tallennettuna yhdelle käyttäjälle. Tarpeeksi turvallisen sisäänkirjautumisjärjestelmän luominen saattaa kuitenkin osoittautua hankalaksi.

2.2 Priorisaatio ”should have”

Sovelluksen pitäisi pystyä tekemään loppuraportti (esim. txt-tiedostomuodossa) tuloksista, kuten onnistumisista ja epäonnistumisista, ja antaa se käyttäjän tarkasteltavaksi. Tämä oli sovelluksen toimeksiannossa yksi niistä ideoista, joka voisi olla toteutuksessa. Loppuraportin avulla käyttäjä saisi helposti tietää, kuinka moni haku- ja vientiyritys onnistui tai epäonnistui. Käyttäjä voisi tämän jälkeen käydä manuaalisesti hakemassa ja viemässä tiedot niiden hakijoiden kohdalta, joiden tulosten haku tai vienti epäonnistui.

Sovelluksessa pitäisi olla helposti ymmärrettävä käyttöliittymä. Sovellusta tulee mahdollisesti käyttämään henkilöitä, jotka eivät ymmärrä teknologiasta paljoa.

Mukavan käyttökokemuksen takaamiseksi helposti ymmärrettävä käyttöliittymä on lähes välttämätön ominaisuus toimivassa sovelluksessa.

2.3 Priorisaatio ”could have”

Sovellusta olisi hyvä pystyä käyttämään sekä suomeksi että englanniksi. On mahdollista, etteivät kaikki sovelluksen käyttäjät ole suomea pääasiallisena kielenään puhuvia. Tämän takia olisi hyvä, jos sovellusta voisi käyttää myös englanniksi. Ruotsin kielellekin sovelluksen voisi kääntää. Siinä tapauksessa olisi kuitenkin turvauduttava ulkopuoliseen apuun käännöksissä. Toimeksiantajalla on kuitenkin lähes varmasti suomea pääasiallisena kielenään puhuva henkilö saatavilla käyttämään sovellusta, joten sen kääntäminen muille kielille ei ole korkealla prioriteettilistalla.

Sovelluksen olisi hyvä pystyä tekemään raportteja onnistuneista tulosten vienneistä (missä näkyisi haun nimi, hakukohteen nimi, hakijoiden nimet, Opintopolkutunnukset ja heidän pisteensä). Käytännössä tämä on jatkovaatimus ”should have” -kohdan vaatimuksesta *sovelluksen täytyy pystyä tekemään loppuraportti (esim. txt-tiedostomuodossa) tuloksista, kuten onnistumisista ja epäonnistumisista, ja antaa se käyttäjän tarkasteltavaksi.* Erona vaatimuksissa on se, että tässä vaatimuksessa sovellus antaisi enemmän tietoa suorituskerran lopuksi. Siinä missä aiempi vaatimus raportoi vain onnistumisista ja epäonnistumisista, tämän vaatimuksen mukainen toteutus antaisi käyttäjälle laajemmin tietoja hakijasta.

2.4 Priorisaatio ”will not have”

Vaikka vaatimusmäärittelyyn annettiin vaihtoehdoksi ”will not have” -luokitus, yksikään määrittelyssä mainittu ominaisuus ei saanut sitä. Tämä saattaa johtua laajalti siitä, ettei sovelluksen ideoinnissa ja vaatimusmäärittelyä tehdessä juurikaan lähdetty tekemään vaatimuksia sovelluksen pääasiallisen toiminnan ulkopuolelta. ”Will not have” -luokituksen saavia vaatimuksia saattaa kuitenkin ilmentyä esimerkiksi sovelluksen suunnittelu- tai kehitysvaiheessa.

3 Sovelluksen suunnittelu

Sovelluksen suunnittelussa on kolme olennaista osaa, joita täytyy tarkastella etukäteen, ennen kuin sovellusta alkaa kehittämään: College Boardin API:n (Application Programming Interface, sovellusohjelmointirajapinta) opiskelu, Opintopolun API:n opiskelu sekä sovelluksen suuripiirteinen toimintatapa.

Lisäksi suunnitteluvaiheessa on pohdittava sitä, miten hakijoiden profiilit yhdistetään College Boardin ja Opintopolun välillä. Yhdellä hakijalla ei ole valmiiksi kummankaan palvelun puolelta sellaista tekijää, joka mahdollistaisi profiilin mutkattoman yhdistämisen palveluiden välillä. Tämä tulee olemaan tärkeä osa suunnitteluprosessia, sillä profiilien on oltava 100 % varmuudella samalle hakijalle kuuluvat.

3.1 College Board API

College Boardin tarjoamaa API:ta käytetään hakijoiden SAT-pistetietojen hakuun. Verkkosivujensa mukaan College Board tarjoaa API:n, jossa pystyy kirjautumaan palveluun sekä hakemaan käyttäjätunnusten käyttöoikeuksien mukaan käyttäjän organisaatiota (tässä tapauksessa Turun ammattikorkeakoulua) koskevia tiedostoja, joissa on listattuna organisaatioon hakeneet hakijat sekä heidän pistetietonsa [2]. Nämä tiedostot ovat toimeksiantajan mukaan ladattavissa sekä CSV-muodossa että tekstitiedostoina. Toimeksiantajan kanssa käydyn keskustelun perusteella College Boardin www-käyttöliittymässä on myös mahdollista hakea yksittäisen hakijan tietoja College Board ID:n perusteella, mutta API:n dokumentaatiossa tällaista mahdollisuutta ei esitetä.

Autentikaatio College Boardin järjestelmään on helposti toteutettu. API:n dokumentaation mukaan autentikointitapoja on kaksi:

- Kirjautuminen ennen tietojen hakemista. Käyttäjä laittaa kirjautumisrajapintaan pyynnön, jossa on mukana käyttäjänimi ja salasana. Kirjautumisen onnistuessa rajapinta palauttaa ticketin, jota käyttäen rajapinnasta voi hakea tietoja 15 sekunnin ajan.
- Autentikaatio samalla, kun tietoja haetaan. Käyttäjä tekee johonkin API-päätepisteeseen (esimerkiksi tiedostojen listaukseen) pyynnön, jonka mukaan laittaa käyttäjätunnuksen ja salasanan. College Boardin järjestelmä tarkistaa,

onko käyttäjätunnukset oikeat sekä onko niillä oikeuksia tehdä kyseistä toimintoa, jonka jälkeen se tekee pyynnössä toivotun toiminnon.

Sovelluksessa tullaan käyttämään ensimmäisenä mainittua vaihtoehtoa, kirjautumista ennen tietojen hakemista. Ensimmäinen vaihtoehto on College Boardin mukaan tarkoitukseemme parempi, sillä kirjautumisen yhteydessä saatavan ticketin avulla sovellus voi ladata useamman tiedoston College Boardista peräkkäin. Tämä tulee tarpeeseen, koska toimeksiantajan mukaan ladattavia tiedostoja on yleensä useita. Näin sovelluksen ei tarvitse lähettää käyttäjätunnuksia College Boardin järjestelmään kuin kerran. [3]

Suureksi ongelmakohdaksi College Boardin API:n testaamisessa muodostuu testausympäristön puute. College Board ei tarjoa API:lle minkäänlaista testausympäristöä mikä tarkoittaa, että API:n tutkimisessa olisi käytettävä sekä oikeita API-päätepisteitä että oikeiden hakijoiden henkilötietoja. Ongelmana on myös se, että toimeksiantaja ei voi antaa testauskäyttöön oikeita College Board -käyttäjätunnuksia, sillä kehittäjällä täytyisi olla työsuhde Turun ammattikorkeakouluun. Tämän takia osa College Boardin API:n tutkimisesta täytyy tehdä vasta sovelluksen kehitysvaiheessa.

Vaikka College Boardin API:lla ei ole testausympäristöä, sieltä saatavaa tietoa on silti mahdollista käsitellä toimeksiantajan toimittaman CSV-tiedoston avulla. Kyseisessä tiedostossa on kuitenkin oikeiden ihmisten henkilötietoja, joten tietoja on käsiteltävä varoen. Esittelyä varten luotiin tiedoston muiden, oikeiden hakijoiden lisäksi yksi testihakija, jonka avulla tuloksia voi esitellä. Tiedostoa on hyvä käyttää ohjelman toiminnan testaamiseen, sillä sen data on samassa muodossa kuin College Boardin API:sta tulevissa tiedostoissa. College Boardista tulevien tietojen käsittelyyn luotiin testikoodi (Kuva 1).

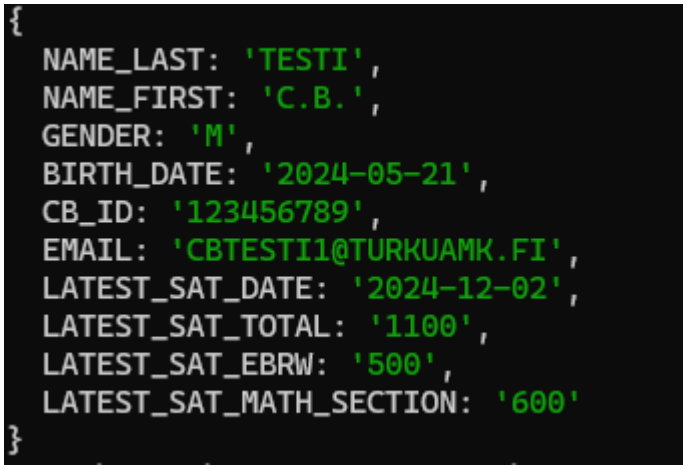
```
import parser from "csv-parser";
import fs from "fs";
const results = [];

fs.createReadStream("6867_20231227_073353.csv")
  .pipe(parser({}))
  .on("data", (data) => results.push(data))
  .on("end", () => {
    console.log(results[results.length - 1]);
    const list_properties = [
      "NAME_LAST",
      "NAME_FIRST",
      "GENDER",
      "BIRTH_DATE",
      "EMAIL",
      "CB_ID",
      "LATEST_SAT_DATE",
      "LATEST_SAT_TOTAL",
      "LATEST_SAT_EBRW",
      "LATEST_SAT_MATH_SECTION",
    ];

    for (const entry of results) {
      for (const property in entry) {
        if (!list_properties.includes(property)) {
          delete entry[property];
        }
      }
    }

    console.log(results[results.length - 1]);
  });
```

Kuva 1. College Board -datan käsittelyn testaukseen käytetty koodi.

College Boardista saatavia tietoja on käsiteltävä jonkin verran, sillä yhdelle hakijalle on tiedostossa listattu 682 eri kohtaa, joista suurinta osaa ei tarvita. Tarvittavia tietoja on toimeksiantajan mukaan kaikkien SAT-kokeiden tulokset sekä tunnistusta varten College Board ID, nimi ja sähköposti. Kuvan 1 koodissa eliminoidaan jokaisen käyttäjän kohdalla kaikki kohdat, jotka eivät ole listattuna *list_properties* -listassa. Tämä toteutetaan käyttämällä CSV-tiedoston käsittelemiseen luotua kirjastoa *csv-parser*. Csv-parser luo CSV-tiedoston jokaisesta rivistä JavaScript-objektin. Objektit ovat tämän ansiosta helposti käsiteltävässä listassa. Tämän jälkeen listan jokaisen objektin

jokainen kohta käydään läpi: jos kohta ei ole *list_properties* -listassa, se poistetaan objektista.

Koodin lopputulos on saanut yhden henkilön tiedot 682 rivistä 10 riviin (Kuva 2).



Kuva 2. 682 kohdan listasta poimitut sovellukselle tarpeelliset tiedot.

Testikäyttöön ei kuitenkaan otettu aivan kaikkia tietoja, joita lopulliseen sovellukseen tarvitaan. Yhdellä hakijalla voi olla esimerkiksi useampi SAT-koe tehtynä ajan mittaan. Näistä kokeista ei voi valita vain uusinta, kuten kuvassaKuva 2, vaan jokaista suorituskertaa täytyy tarkastella, ja niistä valita se, jossa hakija oli suoriutunut parhaiten. Kokeiden tuloksia ei saa myöskään yhdistellä esimerkiksi niin, että uusimmasta kokeesta otetaan EBRW eli *Evidence-based Reading and Writing* -pisteet, ja toiseksi uusimmasta matematiikkaosan pisteet. Tälle täytyy tehdä sovellusta kehittäessä oma osionsa, jossa tarkastellaan, onko hakijalla aiempia SAT-suorituskertoja, joissa hän on pärjännyt paremmin kuin uusimmassa.

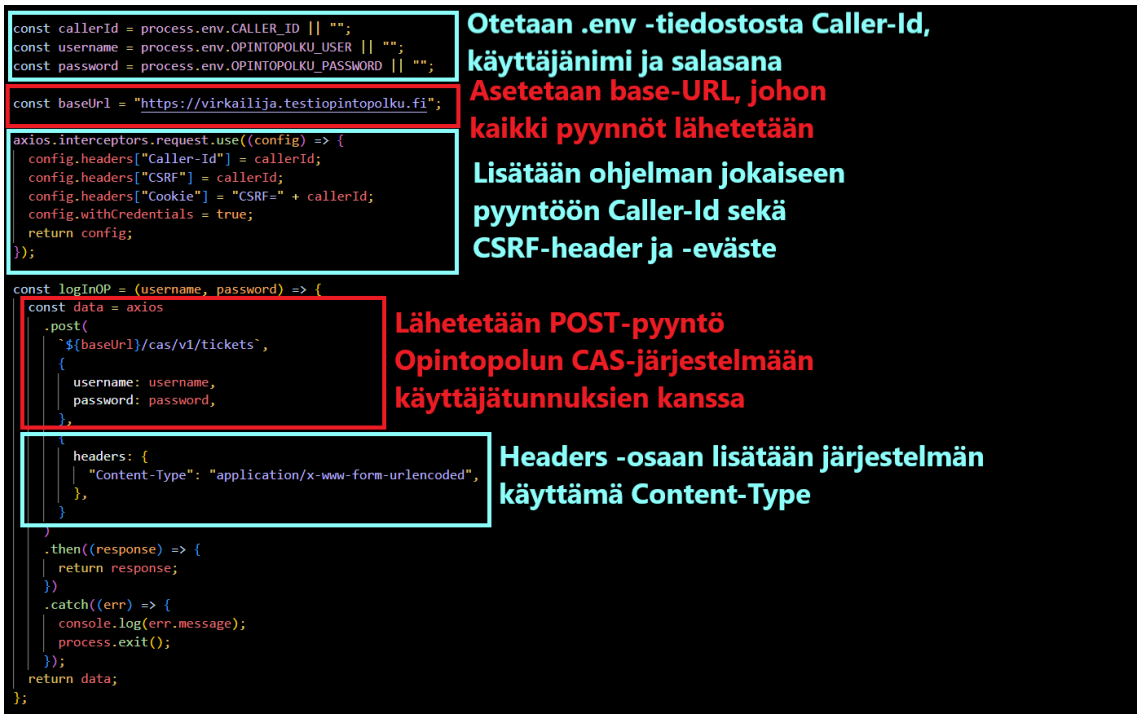
3.2 Opintopolun API

Opintopolun API:ta käytetään sovelluksessa parissa eri kohdassa. Aluksi käyttäjä syöttää hakijoiden Opintopolku ID:t sovellukselle, jonka jälkeen sovellus hakee Opintopolun API:sta ID:n perusteella hakemuksen. Hakemuksesta selviää ID:n lisäksi esimerkiksi hakijan nimi, sähköposti, ja mahdollisesti College Board ID. Näitä tietoja käytetään College Boardin puolella SAT-pisteiden hakuun. Toisen kerran Opintopolun API:ta tarvitaan, kun College Boardista haettuja SAT-tietoja halutaan syöttää Opintopolun järjestelmään.

Opintopolun autentikaatiotapa on hieman monimutkaisempi College Boardiin verrattuna. Opintopolku käyttää autentikaatioonsa myös käyttäjätunnusta ja salasanaa, mutta autentikaatio tapahtuu Opintopolussa käytetyn CAS:n (Central Authentication Service) kautta. CAS on Apereo Foundationin kehittämä kertakirjautumisjärjestelmä (single sign-on, SSO). CAS:n avulla käyttäjän ei tarvitse lähettää session aikana kirjautumistietojaan palveluun kuin kerran. [4]

Opintopolun dokumentaatiossa Eduuni-sivustolla on ohjeet oikeaoppiseen Opintopolun API:n autentikaatioon [5]. Dokumentaation mukainen, testimuotoinen autentikaatio ja sen toimivuuden varmistus toteutetaan JavaScript (Node.js) -koodilla. Toteutusprosessi selitetään tässä kuvien avulla.

Ensimmäisenä esitellään aluksi tehtävää tietojen alustusta sekä CAS:iin sisäänkirjautumista (Kuva 3).

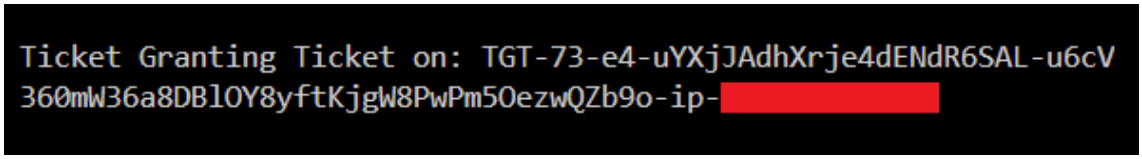


Kuva 3. Ohjelman alustus sekä kirjautuminen Opintopolun CAS-järjestelmään.

Opintopolun järjestelmiin kirjautuminen vaatii koodin puolelta hieman alustusta. Ensiksi haetaan ympäristömuuttujiin säilötty ”Caller-Id”, joka on laitettava jokaiseen Opintopolun API:hin lähtevään kutsuun. Caller-Id:n avulla Opintopolun puolella tiedetään, kuka palvelua käyttää. [6] Tälle sovellukselle valittu Caller-Id on muotoa *{organisaation Opintopolku-id}.sovellukselle-annettu-nimi*.

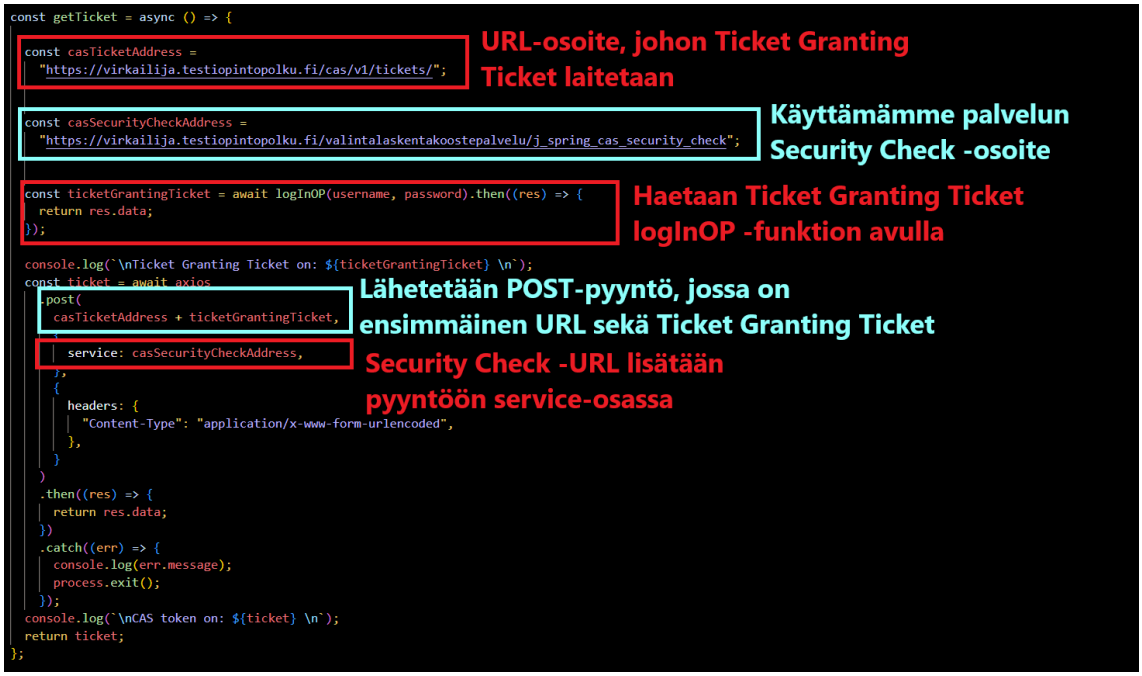
Caller-Id:n lisäksi jokaiseen kutsuun lisätään CSRF-otsake (engl. header) ja -eväste. CSRF eli Cross-site Request Forgery on hyökkäys, jossa hyökkääjä suorittaa käyttäjän tunnuksilla ei-haluttuja komentoja, esimerkiksi pääsemällä käsiksi selaimeen tallennettuihin käyttäjätunnuksiin [7]. CSRF-hyökkäyksien estämiseen sovelluksessa käytetään CSRF-otsaketta ja -evästettä, joilla on kummallakin sama arvo. Tätä kutsutaan Cookie Double Submit -tekniikaksi. Eduuni-sivun mukaan CSRF-suojauksen tehokkuuden varmistamiseksi ”sen on vaikutettava kaikkiin tilaa muuttaviin pyyntöihin, mikä aiheuttaa muutostarpeen sovelluksiin, jotka sellaisia pyyntöjä tekevät”. [8] Sovelluksen tapauksessa CSRF-otsakkeeksi ja -evästeeksi laitetaan Caller-Id, Eduuni-sivun ohjeiden mukaisesti.

Koodin alustuksen jälkeen on aika siirtyä itse pyyntöjen tekemiseen. Autentikaation ensimmäinen vaihe on lähettää Opintopolun CAS-palveluun POST-pyyntö, johon on laitettu form-parametreissa mukaan käyttäjänimi ja salasana. Tärkeää on muistaa asettaa Eduuni-sivun autentikaatio-ohjeiden mukainen Content-Type, jotta pyynnön vastaanottava palvelin osaa lukea sovelluksesta lähetettyä dataa. Seuraavaksi nähdään palvelimen vastaus pyyntöön (Kuva 4).



Kuva 4. CAS-järjestelmän palauttama Ticket Granting Ticket (TGT), josta käyttäjän IP-osoite sensuroituna.

Huomataan, että palvelin palauttaa Ticket Granting Ticketin (TGT). Ticket Granting Ticket on autentikaation jälkeen käytettävä tapa tunnistautua Opintopolun palveluihin. Jokaiseen Opintopolun eri palveluun tarvitsee luoda erillinen sessio. TGT mahdollistaa sen, ettei käyttäjätunnuksia tarvitse lähettää aina, kun eri Opintopolun palveluja aiotaan käyttää. Tämän sijaan session luomiseen voidaan käyttää TGT:tä, joka saatiin autentikaation yhteydessä. Kun TGT on haettu, sitä käytetään ”service ticketin” hakemiseen seuraavasti (Kuva 5).

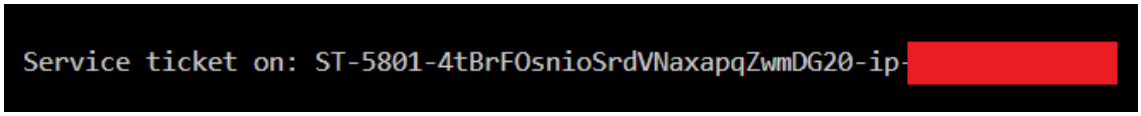


Kuva 5. Service ticketin hakeminen TGT:n avulla.

Opintopolun REST-rajapinnat on jaettu useisiin eri palveluihin, joista jokainen vaatii erillisen TGT:n avulla tehtävän autentikaation [9]. Sovelluksessa tullaan käyttämään useita näistä palveluista. Service ticket on Opintopolun järjestelmän palvelukohtainen autentikaatiotapa. Se haetaan Opintopolun järjestelmästä POST-pyynnöllä TGT:n avulla kahta URL-osoitetta käyttäen. Ensimmäinen osoitteista on se, johon itse pyyntö laitetaan, ja tämä osoite pysyy samana riippumatta siitä, mihin Opintopolun palveluun autentikaatioyritys kohdistuu. Toinen osoitteista laitetaan pyynnön mukana ”service”-parametrinä. Tämä osoite on esimerkiksi muotoa *https://service.xyz/example-service/j_spring_cas_security_check*. [5] Esimerkkiosoitteen kohta *service.xyz* korvataan sovelluksen tapauksessa osoitteella *virkailija.opintopolku.fi* tai *virkailija.testiopintopolku.fi* riippuen siitä, onko kyseessä testaus- vai tuotantoympäristö. Kohta *example-service* korvataan sillä Opintopolun palvelulla, johon service ticket aiotaan hakea. Kuvassa 5 esitetyn esimerkin mukaan osoitteeksi tulee siis *https://virkailija.testiopintopolku.fi/valintalaskentakoostepalvelu/j_spring_cas_security_c*

heck, koska käytössä on Opintopolun testiympäristö ja service ticket halutaan valintalaskentakoostepalveluun.

Näillä tiedoilla järjestelmästä saadaan vastauksena service ticket seuraavasti (Kuva 6).



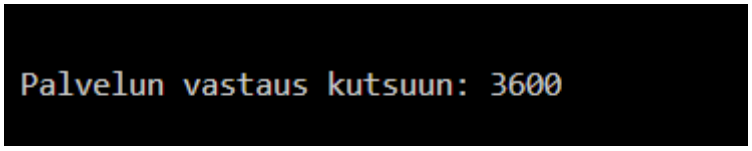
Kuva 6. Service ticket, jossa käyttäjän IP-osoite sensuroituna.

Service ticketillä päästään viimeinkin hakemaan oikeaa dataa Opintopolun rajapinnoista. Jokaisen kutsun yhteyteen on kuitenkin hyvä luoda uusi service ticket, sillä yksi service ticket on voimassa vain 10 sekuntia [5]. Seuraavaksi testaamme service ticketin toimintaa (Kuva 7).

Kuva 7. Service ticketin avulla tehtävä pyyntö Opintopolun API-päätepisteeseen.

Testikäyttöön valitaan tässä tapauksessa valintalaskentakoostepalvelun maxinactiveinterval-päätepiste, joka palauttaa session erääntymisen aikarajan sekunteina [10]. Service ticket lisätään pyyntöön *ticket*-URL-parametrinä. Käytettävällä päätepisteellä ei testikäytössä ole kuitenkaan merkitystä, sillä ensimmäisenä tärkeintä on varmistaa, että service ticket toimii. On kuitenkin erittäin tärkeä muistaa, että service ticket toimii *vain* sen palvelun kohdalla, mihin se on alun perin haettu. Esimerkiksi *seuranta-service*-palveluun haettu service ticket ei toimi valintalaskentakoostepalveluun kohdistuvissa pyynnöissä, ja palvelu palauttaa tässä tapauksessa virhekoodin.

Kun edellä mainittua rajapintaa kutsutaan, palvelusta tulee seuraava vastaus (Kuva 8).



Kuva 8. Päätepisteen palauttama vastaus.

Näin on varmistettu, että valintalaskentakoostepalveluun luotu service ticket toimii kunnolla. Esimerkin mukaista koodia voi jatkossa käyttää service ticketin luomiseen jokaisen Opintopolun palvelun kohdalla, mitä sovelluksen tarvitsee käyttää saadakseen halutun tuloksen.

3.3 Hakijoiden profiilien yhdistäminen toisiinsa College Boardin ja Opintopolun välillä

Kun opinnäytetyötä lähdettiin alustamaan, yksi olennaisimmista ratkottavista ongelmista oli se, kuinka yhden hakijan Opintopolkuprofiilin ja College Board -profiilin saa yhdistettyä toisiinsa. Palvelujen välillä ei ole minkäänlaista yhteistä tunnistetapaa, joten tunnistautuminen on tehtävä jollain muulla tavalla. Tähän yksi selvä ratkaisu olisi sähköpostiosoitteen käyttäminen yhdistävänä tekijänä, sillä sähköpostiosoitteet ovat aina ainutlaatuisia. Toimeksiantajan mukaan hakijoiden ilmoittamat sähköpostiosoitteet kuitenkin eriävät valitettavan usein Opintopolun ja College Boardin välillä, joten sähköpostiosoite ei voi toimia yksinään identifioivana tekijänä.

Toimeksiantaja mainitsi, että he pystyvät kysymään Opintopolussa College Board ID:tä hakemuksen tekemisen yhteydessä. Hakija voi kuitenkin ilmoittaa College Board ID:n väärin haun yhteydessä, joten tämäkään ei voi toimia 100 %:n varmuudella yhdistävänä tekijänä. Tämän takia sovellus tulee käyttämään sekä sähköpostiosoitetta että College Board ID:tä hakijan tunnuksien yhdistämiseen Opintopolun ja College Boardin välillä. Näin profiilit voidaan mahdollisesti yhdistää, vaikka joko sähköpostiosoite tai College Board ID olisi väärin ilmoitettu. Pitää kuitenkin ottaa huomioon, ettei tälläkään tavalla yhteyden löytyminen ole sataprosenttisen varmaa. On siis mahdollista, että sovelluksen käyttäjä joutuu joidenkin hakijoiden kohdalla käydä palveluissa käsin hoitamassa kyseisten hakijoiden pistetietojen viemisen.

3.4 Sovelluksen toiminta

Sovellus tulee muodostumaan kolmesta komponentista. Ensimmäiseksi sovellus tarvitsee toimivan palvelinpuolen eli back-endin, josta lähetetään REST-pyyntöjä sekä Opintopolun että College Boardin API:hin. Palvelinpuoli tulee toimimaan ikään kuin koko sovelluksen selkärankana, joka suorittaa kaikista tärkeimmät tehtävät.

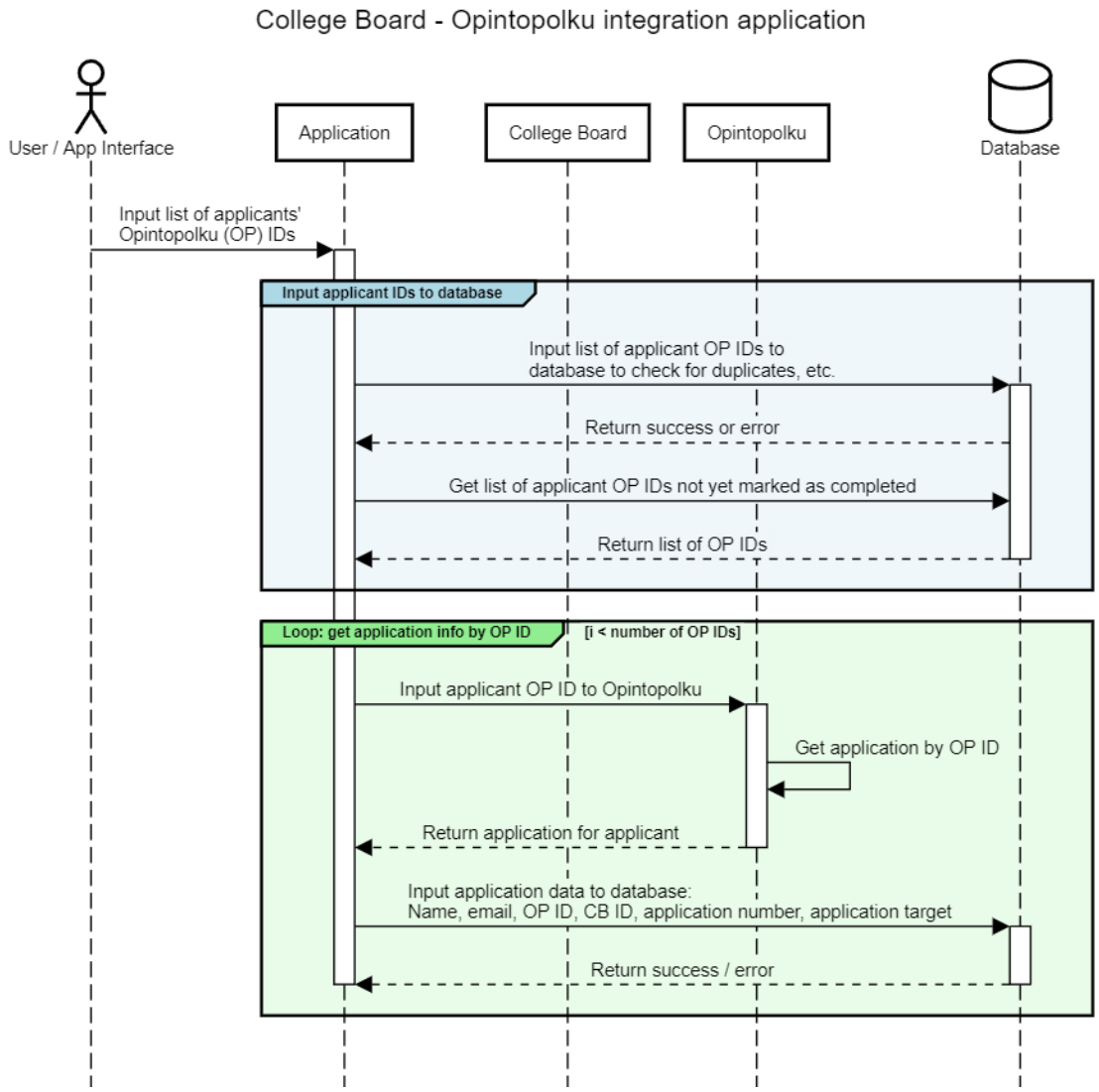
Toiseksi sovellus tarvitsee sopivan käyttöliittymän eli front-endin. Käyttöliittymästä ei tulla lähettämään REST-pyyntöjä muualle kuin sovelluksen palvelinpuolelle, joka sitten toteuttaa käyttöliittymästä tulleet käskyt. Tämä tarkoittaa sitä, että palvelinpuolelle täytyy myös luoda API-päätepisteitä, johon käyttöliittymästä voi lähettää pyyntöjä.

Kolmantena komponenttina sovelluksessa tulee toimimaan tietokanta, johon tallennetaan sekä käyttäjän käyttäjätunnukset että tiedot prosessoiduista hakijoista, kuten hakukohde, SAT-pisteet sekä status siitä, onko hakijan tietojen välittäminen Opintopolun ja College Boardin välillä onnistunut. Tietokannan suunnittelussa kehittäjän on oltava erityisen tarkkana, sillä henkilötietoja täytyy säilyttää Euroopan unionin GDPR-tietosuoja-asetusten mukaisesti. On esimerkiksi hyvä asettaa jokin aikaraja, jonka jälkeen hakijoiden henkilötiedot poistetaan tietokannasta. Kehitysvaiheessa sovelluksen tietojenkäsittelyä on muutenkin pohdittava Turun ammattikorkeakoulun tietosuojavastaavan kanssa, jotta GDPR-asetusta ei rikottaisi.

Ennen kuin sovelluksen päätoimintoja, eli API-kutsuja ja tietojen käsittelemistä, voi lähteä suorittamaan, on sovellukseen luotava turvallinen ympäristö näiden toteuttamiseen. Käytännössä tämä tarkoittaa riittävien turvallisuustoimenpiteiden toteuttamista: sovellukseen on luotava vedenpitävä sisäänkirjautumisjärjestelmä sekä on varmistettava, että esimerkiksi tietokannasta ei pysty vuotamaan tietoja ulkopuolisille tahoille. Pyyntöjä sovellukseen on hyvä tehdä vain HTTPS:n kautta, sillä HTTP-kutsujen tiedot eivät ole salattuja, mikä luo tietoturvariskin henkilötietoja tai käyttäjätunnuksia siirrettäessä. [11] Tietokannassa säilytettävät salasanat sekä Opintopolun ja College Boardin autentikaation yhteydessä saadut ticketit on hyvä salata esimerkiksi hajautusalgoritmien avulla. Ticketejä käyttämällä hyökkääjä voisi pahimmillaan saada täydet käyttöoikeudet Opintopolun tai College Boardin palveluihin.

Sovelluksen pääasiallinen toiminta, eli ne prosessit, joissa hakijoiden tietoja käsitellään, voidaan jakaa viiteen eri vaiheeseen. Jokaisella vaiheella on oma tavoitteensa, jonka täytyy onnistua ennen kuin seuraavaan osaan voidaan siirtyä. Alla nähdään

sovelluksen toiminnan kaksi ensimmäistä vaihetta graafisesti kuvattuna UML-sekvenssikaavion avulla (Kuva 9). Sovelluksen vaiheeksi ei ole laskettu esimerkiksi sisäänkirjautumisia sovellukseen, Opintopolkuun tai College Boardiin. Mainittakoon myös, että sovelluksen toiminnan vaiheet kaaviossa ovat laajalti yksinkertaistetut.

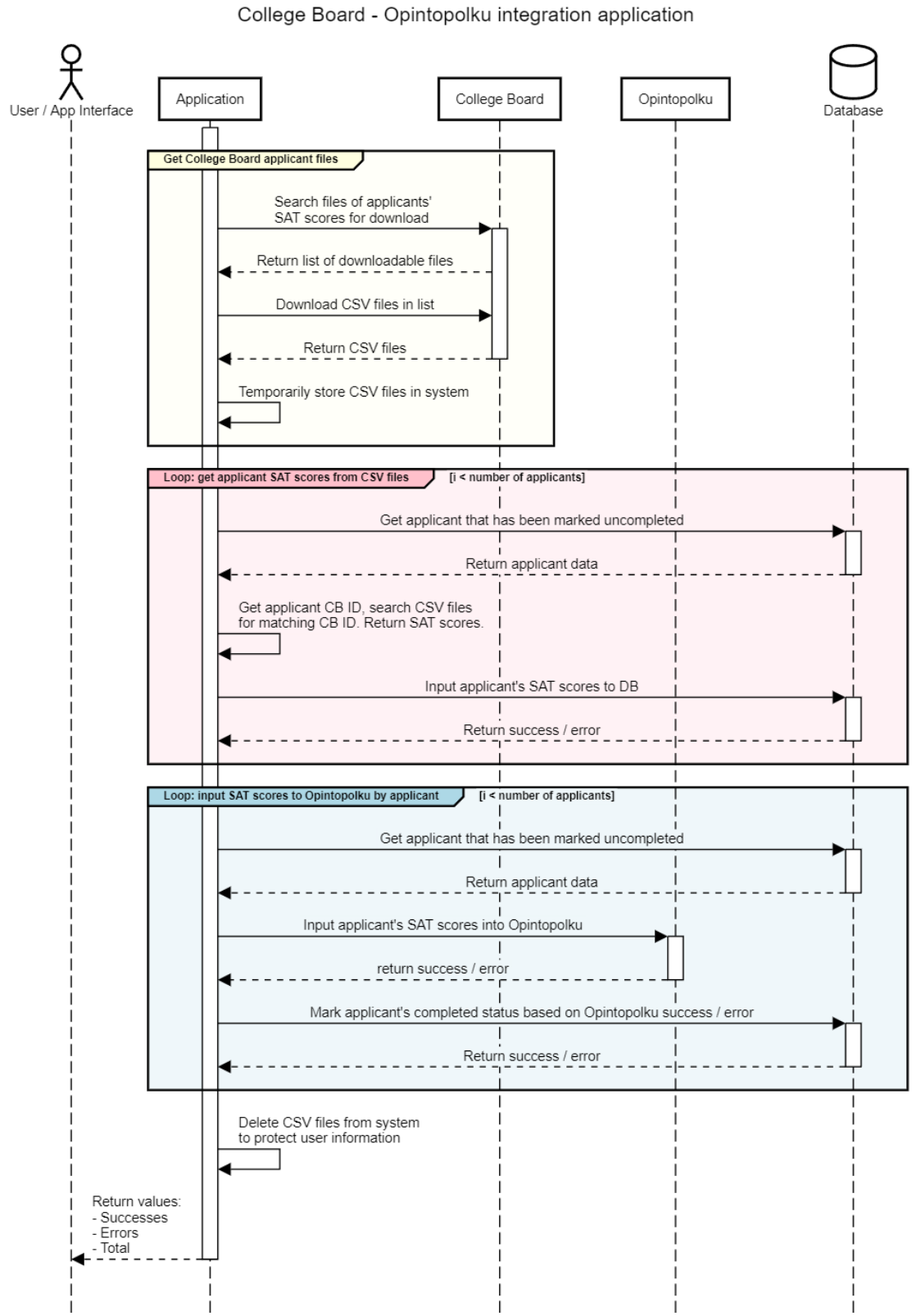


Kuva 9. Sovelluksen toiminnan kaksi ensimmäistä vaihetta.

Sovelluksen toiminnan ensimmäisessä vaiheessa syötetään sovelluksen käyttäjän antamia Opintopolku id:itä tietokantaan. Jokainen id:istä saa oman kohtansa tietokannassa, johon tallennetaan myöhemmin sen avulla etsittyjä tietoja hakijasta, kuten nimi, sähköposti ja SAT-pistetiedot. On hyvä tarkistaa käyttäjän syöttämät Opintopolku id:t, sillä virheitä saattaa tapahtua. Esimerkiksi saman id:n voi vahingossa syöttää järjestelmään kaksi kertaa. Jos duplikaatti-id:itä ei tarkistettaisi, järjestelmä

saattaisi myöhemmissä vaiheissa hakea turhaan hakijan hakemusta, jolle on jo aiemmin syötetty pistetiedot. id:iden syöttämisen jälkeen ensimmäisessä vaiheessa tietokannasta haetaan ne id:t, joilla ei ole vielä merkintää siitä, että pistetietojen välittäminen on onnistunut. Näin saadaan haettua seuraavaa vaihetta varten niiden hakijoiden id:t, joiden tiedot ovat vielä käsittelemättä.

Toisessa vaiheessa tavoitteena on etsiä hakijan hakemus tai hakemukset id:n perusteella. Tässä tullaan käyttämään silmukkaa (engl. loop), joka etsii jokaisen id:n kohdalla Opintopolun järjestelmästä ne hakemukset, jotka on laitettu haluttuun organisaatioon, sovelluksen kohdalla Turun ammattikorkeakouluun. Opintopolun järjestelmään tunnistaudutaan luvun 3.2 mukaisella tavalla. Koska sovelluksen päämäärä on syöttää SAT-pistetietoja hakemuksiin, voidaan suodattaa sellaiset hakemukset pois, joissa ei ole SAT-pistetiedoille erillistä kohtaa merkittynä. Hakemuksen löytämisen jälkeen hakemuksesta syötetään relevantit tiedot, kuten nimi, College Board id ja hakukohde, tietokantaan seuraavia vaiheita varten. Jos id:llä ei löydy hakijaa, voi id:n tietojen haun merkitä tietokannassa epäonnistuneeksi. Tämän tiedon voi välittää sovelluksen käyttäjälle, vaatimusmäärittelyn kohdan *Sovelluksen täytyy pystyä tekemään loppuraportti (esim. txt-tiedostomuodossa) tuloksista, kuten onnistumisista ja epäonnistumisista, ja antaa se käyttäjän tarkasteltavaksi.* mukaisesti. Kun jokaisen id:n tiedot on haettu Opintopolun järjestelmästä, voidaan siirtyä kolmanteen vaiheeseen. Vaiheet 3–5 ovat kuvattuna seuraavaksi (Kuva 10).



Kuva 10. Sovelluksen toiminnan kolme viimeistä vaihetta.

Prosessin kolmannessa vaiheessa haetaan opiskelijoiden pistetietoja College Boardista. College Boardiin tunnistaudutaan luvun 3.1 mukaisella tavalla. Ensin College Boardin API:hin tehdään pyyntö listasta, jossa on listattuna käyttäjän ladattavat tiedostot. Tämän jälkeen ladataan nämä tiedostot CSV-muodossa, ja säilötään ne palvelimelle väliaikaisesti. Tiedostoja ei kannata säilyttää palvelimella pysyvästi, sillä ne sisältävät hakijoiden henkilötietoja.

Kun sovellus on ladannut CSV-tiedostot palvelimelle, voidaan aloittaa neljäs vaihe, jossa yritetään yhdistää hakijan Opintopolku- ja College Board -profiilit toisiinsa. Tähän tullaan käyttämään silmukkaa. Silmukassa haetaan tietokannasta hakija kerrallaan ne hakijat, joilla ei vielä ole merkintää siitä, että pistetietojen välittäminen on onnistunut. Tämän jälkeen käydään läpi CSV-tiedostojen rivejä, esimerkiksi luvun 3.1 näytteen mukaisesti niin kauan, kunnes tiedostoista löydetään hakemuksessa ilmoitettua College Board id:tä vastaava id. Jos ilmoitettua College Board id:tä ei löydy tiedostoista, voidaan yrittää hakea tietoja sähköpostiosoitteen perusteella. Onnistuneen pistetietojen haun jälkeen hakijan pistetiedot viedään tietokantaan. Jos sähköpostiosoitteen perusteella ei myöskään löydy tietoja, voidaan pistetietojen haku merkitä tietokantaan epäonnistuneeksi. Sovelluksen käyttäjä saa tiedon jokaisesta epäonnistumisesta aiemmin mainitun vaatimusmäärittelyn mukaisen loppuraportin yhteydessä.

Viimeiseen vaiheeseen siirrytään niiden hakijoiden osalta, joiden Opintopolku- ja College Board -profiilit on onnistuneesti yhdistetty. Vaihe toteutetaan jälleen kerran silmukan avulla. Ensin haetaan tietokannasta hakija, jolle ei ole vielä merkintää siitä, että pistetietojen välittäminen on onnistunut. Tämän jälkeen syötetään College Boardista saadut hakijan SAT-pistetiedot opintopolkuun, hakemuksen ID:tä käyttäen. Jos hakija on hakenut yhdessä haussa useampaan hakukohteeseen, pistetiedot on täytettävä vain yhteen näistä hakukohteista. Kun Opintopolun järjestelmästä tulee onnistumiseen viittaava vastaus, hakijan kohdalle merkitään tietokantaan, että pistetietojen vienti on onnistunut. Tämä prosessi toteutetaan silmukassa jokaisen tähän vaiheeseen päässeen hakijan kohdalla.

Kun kaikki viisi vaihetta on käyty läpi, on aika poistaa CSV-tiedostot järjestelmästä. Tiedostoista on tähän mennessä tallennettu tietokantaan kaikki sovellukselle ja käyttäjälle tarpeellinen tieto, joten CSV-tiedostojen pitäminen palvelimella olisi merkityksetöntä, minkä lisäksi se saattaisi luoda tietoturvariskin. Tiedostojen poistamisen jälkeen palautetaan lopputulos käyttäjälle esimerkiksi aiemmin mainitun

loppuraportin muodossa. Käyttäjälle voidaan välittää tietoa prosessin kulusta myös sen suorituksen aikana, vaikka sitä ei ole mainittukaan UML-sekvenssikaaviossa. Tämä johtuu siitä, ettei prosessi todennäköisesti tule olemaan valmis sekunneissa, varsinkaan siinä tapauksessa, kun hakijoita on useita.

4 Kehitysvaiheeseen siirtyminen

Sovelluksen suunnitteluvaiheen jälkeen sovellusta aletaan kehittää suunnitelman mukaisesti. Sovelluksen kehitysvaihe jää kuitenkin opinnäytetyön ulkopuolelle, sillä opinnäytetyönä tehdään vain sovelluksen suunnitelma. Sovelluksen kehityksen suunnittelu on kuitenkin mahdollista. Tähän sisältyy toimintojen kehittämisjärjestys, sovelluksen kehittämiseen käytettävät teknologiat sekä sovelluksen julkaisusuunnitelma kehittämisen päätyttyä.

4.1 Sovelluksen kehittämiseen käytettävät teknologiat

Tässä luvussa suunnitellaan luvussa 3.4 mainittujen sovelluksen osien kehittämiseen käytettävät teknologiat.

4.1.1 Palvelinpuoli eli back-end

Palvelinpuolen kehittämiseen käytetään TypeScriptiä, Node.js:ää sekä Express.js-kirjastoa. TypeScript on JavaScriptin päälle rakennettu ohjelmointikieli, joka lisää sääntöjä JavaScriptiin, tehden kehityksestä sekä turvallisempaa että sujuvampaa. TypeScript ”pakottaa” ohjelmoijan määrittämään muuttujien tyypit (merkkijono, numero, jne.). Tämä tekee ohjelmoinnista selkeämpää kehittäjälle, sillä heidän ei enää itse tarvitse muistaa, mitä tyyppiä jokin tietty muuttuja on, toisin kuin JavaScriptissä. [12] Node.js on avoimen lähdekoodin JavaScript- ja TypeScript-ajoympäristö. Node.js mahdollistaa JavaScript- ja TypeScript-koodin suorittamisen palvelinpuolella. Ilman Node.js:ää koodia olisi mahdollista suorittaa vain selaimissa. [13] Express.js on Node.js:lle luotu kirjasto, joka mahdollistaa erilaisten web-sovellusten helpon kehityksen [14]. Sovelluksessa tarvittavat API-päätepisteet ja -kutsut sekä reititys ovat Expressin avulla helppoja toteuttaa.

4.1.2 Tietokanta

Sovelluksen tietokannassa tulee olemaan kaksi eri taulukkoa, joihin tallennetaan tietoa. Ensimmäinen taulukoista on sovelluksen käyttäjän tiedot: käyttäjänimi, sähköpostiosoite sekä salasana. Toisessa taulukoista säilytetään tietoa hakijoista,

kuten sovelluksen suunnitteluvaiheessa todettiin. Tämän takia on hyvä käyttää yksinkertaista tietokantaa. Päävaatimukset tietokannalle ovat turvallisuus ja helppokäyttöisyys. Luotettavana ja helposti käytettävänä tunnettu MongoDB täyttää nämä vaatimukset. MongoDB on dokumenttitietokanta, joka tallentaa tietoja JSON-tiedostomuotoon pohjautuvassa BSON-muodossa [15]. Koska tietoja käsitellään sovelluksessa muutenkin JSON-tiedostomuodossa, tietojen tallennus tietokantaan on yksinkertaista ja nopeaa.

4.1.3 Käyttöliittymä eli front-end

Käyttöliittymän luomiseen sovelluksessa käytetään Reactia. React on Metan luoma ja ylläpitämä JavaScriptin kirjasto, jonka avulla on helppo toteuttaa interaktiivisia käyttöliittymiä. Tästä esimerkkinä on esimerkiksi komponentit, joiden tilaa voi muuttaa sen mukaan, mitä dataa sovelluksen palvelinpuoli lähettää käyttöliittymää. React-komponenttien luomiseen käytetään usein JSX:ää, joka laajentaa JavaScriptin syntaksia niin, että sillä on mahdollista luoda HTML:n kaltaista koodia. [16]

4.2 Sovelluksen toimintojen kehittäminen

Toimintojen kehittämiselle on hyvä olla jokin järjestys. Järjestyksen luominen tekee kehittämisprosessista suoraviivaisempaa. Näin saadaan myös kaikista tärkeimmät ominaisuudet kehitettyä ensimmäisenä.

1. Palvelinpuolella toimivan sisäänkirjautumisjärjestelmän kehittäminen. Tämä luo pohjan sovellukselle ja sen toiminnoille, sillä jokaista muuta ominaisuutta käyttääkseen käyttäjän täytyy autentikoitua sovellukseen. Sisäänkirjautumisjärjestelmä tulee myös tarvitsemaan tietokantaa, joten sen implementointi ensimmäisten joukossa on tarpeen.
2. Palvelinpuolella opintopolkuun liittyvien toimintojen kehittäminen. Vaikka Opintopolun rajapintoihin täytyy tehdä kutsuja sekä ennen että jälkeen College Boardin järjestelmien kutsumista, on nämä toiminnot silti hyvä tehdä yhdellä kertaa. Opintopolusta saadaan valmiiksi tieto, missä muodossa hakijan SAT-pistetiedot otetaan vastaan. Täten on hyvä selvittää jo valmiiksi, mitä rajapintoja pisteiden syöttämiseen tullaan tarvitsemaan, vaikka tätä ominaisuutta tarvitaan vasta College Boardin rajapintojen käytön jälkeen. Opintopolusta on myös

tärkeää saada hakemuksien tiedot, joissa on College Board id tai sähköpostiosoite profiilien yhdistämistä varten.

3. Palvelinpuolella College Boardiin liittyvien toimintojen kehittäminen. College Boardista ladataan vain organisaatiolle kuuluvia tiedostoja, joten tämän vaiheen voisi periaatteessa toteuttaa myös Opintopolun järjestelmiä koskevia toimintoja ennen. Tässä vaiheessa voidaan kuitenkin suoraan ruveta yhdistämään hakijan Opintopolku- ja College Board -profiileja toisiinsa, kun College Boardista saatavia tietoja aletaan käymään läpi.
4. Palvelinpuolella raportin kehittäminen. Tätä toimintoa varten joutuu todennäköisesti muokkaamaan aiempien toimintojen koodia hieman. Raporttitoiminto on kuitenkin sellainen, joka vaatii jokaisen muun toiminnon oikeanlaisen toiminnan. Ilman sovelluksen oikeanlaista toimintaa raportin luominen ei ole mielekästä.
5. Käyttöliittymän kehittäminen. Kun kaikki sovelluksen ominaisuudet on saatu palvelinpuolella toimimaan, on aika kehittää käyttöliittymä. Käyttöliittymässä tullaan säilyttämään hyvin minimaalinen määrä sovelluksen tarvitsemasta tiedosta, joten sen kehittäminen voi hyvin alkaa vasta silloin, kun palvelinpuolen ominaisuudet on saatu valmiiksi.

4.3 Sovelluksen julkaisu

Koska sovellus tulee Turun ammattikorkeakoulun hakijapalveluiden käyttöön, täytyy sen julkaisussa noudattaa Turun ammattikorkeakoulun ohjeistusta palveluiden hankkimisen suhteen. Paras vaihtoehto olisi saada suoraan ammattikorkeakoululta esimerkiksi Linux-palvelin, johon sovelluksen voisi pystyttää. Näin palvelu olisi varmasti turvallisissa käsissä, ja sen ylläpidosta ei aiheutuisi lisäkuluja. Jos ammattikorkeakoulu ei kuitenkaan voi tarjota palvelinta sovelluksen käyttöön, on muitakin vaihtoehtoja saatavilla. Turun ammattikorkeakoulun service desk kommentoi asiaa näin: *”Alustan tulisi olla meidän hallinnassamme, mielellään eurooppalaisilla palvelimilla, ja palveluntarjoajan kanssa pitää olla selkeä sopimus tehtynä. Suomalainen hosting-palvelu olisi hyvä, jos sitä ei pystytä hostaamaan koulun sisäisesti.”*. Tässä tapauksessa paras vaihtoehto olisi luoda kartoitus suomalaisista verkkoisännöintipalveluista, joista hakijapalvelut valitsisivat mieleisimmän.

Turun ammattikorkeakoulun hakijapalveluista tulisi julkaisun ohella palvelun ylläpitäjä. Tämä tarkoittaa, että hakijapalveluiden täytyy kiinnittää huomiota ammattikorkeakoulun määrittämiin tietojärjestelmän hankintasääntöihin ja omistajan velvollisuuksiin. Palvelun ylläpitäjän tulisi esimerkiksi täyttää liuta erilaisia lomakkeita ennen järjestelmän hankkimista. Lisäksi ylläpitäjän täytyy huolehtia, että henkilötietoja käsitellään lain mukaisesti. Tämä edellyttää muun muassa tietosuojailmoituksen sekä käsittelytoimien selosteen luomisen.

5 Sovelluksen testaus

Sovelluksen kehityksen jälkeen sitä on testattava. Ilman testauksia sovelluksen julkaiseminen olisi vastuutonta, sillä kunnolliset testit takaavat sen, että se toimii kunnolla ja ettei se anna virhetilanteiden tapahtua ilman, että niitä huomataan. Sovelluksessa käytettävälle Node.js:lle on monta erilaista testauskirjastoa, joilla testit voidaan automatisoida, kuten Jest ja Mocha. Luvun ajatus ei kuitenkaan ole paneutua testaukseen käytettäviin teknologioihin, vaan enemminkin tapauksiin, joissa sovellusta on tarpeellista testata, toisin sanoen testitapauksiin.

Ennen sovelluksen kehitystä testitapauksia voidaan tehdä vaatimusmäärittelyn avulla. Testitapauksia voi vaatimusmäärittelyn lisäksi syntyä myös käyttäjäkertomusten kautta sekä sovellusta kehittäessä esille tulleista seikoista. On myös hyvä testata ei-haluttuja toimintoja; yksi testitapauksista voisi varmistaa, ettei käyttäjä pääse kirjautumaan tunnuksilleen käyttämällä väärää salasanaa.

Lähdetään muodostamaan vaatimusmäärittelyn perusteella testitapauksia. Testitapaukset on listattu vaatimus kerrallaan. Yhdestä vaatimuksesta voi muodostua useita testitapauksia. Vaatimukset käydään sovelluksen vaiheiden kulkujärjestyksessä, eikä siinä järjestyksessä, jossa vaatimukset ovat alun perin listattu. Vaatimuksista käydään läpi ainoastaan *must have*- ja *should have* -kohdat. Myös käyttöliittymää koskevat vaatimukset jätetään pois, sillä esimerkiksi käyttöliittymän selkeyttä koskevat seikat ovat laajalti mielipidekysymyksiä.

1. Sovellukseen täytyy pystyä tekemään sujuva sisäänkirjautumisjärjestelmä, jotta ulkopuoliset tahot eivät pääse sovellukseen käsiksi.

ID	TC1.1
Nimi	Onnistunut sisäänkirjautuminen
Ennakko-vaatimukset	Käyttäjä on päässyt sovelluksen kirjautumissivulle.
Kuvaus	Varmistetaan, että sisäänkirjautuminen toimii oikein ja käyttäjä pääsee sovellukseen.
Toivottu tulos	Käyttäjä kirjautuu onnistuneesti sisään.

ID	TC1.2
Nimi	Virheellinen sisäänkirjautuminen
Ennakko-vaatimukset	Käyttäjä on päässyt sovelluksen kirjautumissivulle.
Kuvaus	Testataan virheellisiä kirjautumistietoja.
Toivottu tulos	Sovellus ilmoittaa virheellisistä kirjautumistiedoista, ja ei päästä käyttäjää kirjautumaan sisään.

2. Sovelluksen täytyy pystyä tallentamaan jokaisen hakijan status tietokantaan, jotta tiedetään, onko tietojen haku ja lähetys onnistunut tietyn hakijan kohdalla.

ID	TC2.1
Nimi	Onnistunut hakijan tallennus
Ennakko-vaatimukset	Käyttäjä on kirjautunut sisään sovellukseen.
Kuvaus	Käyttäjä antaa sovellukselle hakijan id:n. Varmistetaan, että hakijan tallennus tietokantaan onnistuu.
Toivottu tulos	Hakija on tallennettu tietokantaan.

ID	TC2.2
Nimi	Onnistunut statuksen muutos
Ennakko-vaatimukset	Käyttäjä on kirjautunut sisään sovellukseen Tietokantaan on tallennettu hakija
Kuvaus	Yritetään muuttaa hakijan statusta tietokannassa.
Toivottu tulos	Statuksen muutos onnistuu.

ID	TC2.3
Nimi	Statuksen muuttamisen epäonnistuminen
Ennakko-vaatimukset	Käyttäjä on kirjautunut sisään sovellukseen Tietokantaan on tallennettu hakija
Kuvaus	Yritetään asettaa virheellinen arvo hakijan statukseen tietokannassa.
Toivottu tulos	Sovellus ilmoittaa virheellisestä arvosta, ja ei muuta hakijan statusta.

3. Sovelluksen täytyy pystyä yhdistämään hakijan College Board -tunnus oikeaan Opintopolkutunnukseen. Sovelluksen täytyy pystyä noutamaan College Boardin rajapinnoista hakijan pistetiedot.

ID	TC3.1
Nimi	Onnistunut hakijan tietojen hakeminen
Ennakko-vaatimukset	Käyttäjä on kirjautunut sisään sovellukseen Käyttäjä on tallentanut hakijan Opintopolku id:n tietokantaan
Kuvaus	Haetaan onnistuneesti hakijan tiedot Opintopolun järjestelmästä.
Toivottu tulos	Opintopolun järjestelmä palauttaa hakijan tiedot, joissa on mukana esimerkiksi hakukohde sekä College Board id.

ID	TC3.2
Nimi	Epäonnistunut hakijan tietojen hakeminen
Ennakko-vaatimukset	Käyttäjä on kirjautunut sisään sovellukseen Käyttäjä on tallentanut **virheellisen** Opintopolku id:n tietokantaan
Kuvaus	Haetaan virheellisellä id:llä tietoja Opintopolun järjestelmästä.
Toivottu tulos	Opintopolun järjestelmä palauttaa virheen, josta sovellus ilmoittaa käyttäjälle.

ID	TC3.3
Nimi	Hakijan tietojen tallentaminen tietokantaan
Ennakko-vaatimukset	Käyttäjä on kirjautunut sisään sovellukseen Käyttäjä on tallentanut hakijan Opintopolku id:n tietokantaan Sovellus on hakenut id:llä hakijan tiedot onnistuneesti
Kuvaus	Hakijan tiedot on haettu Opintopolusta, minkä jälkeen ne tallennetaan hakijan id:n kohdalle tietokantaan.
Toivottu tulos	Sovellus tallentaa hakijan tiedot tietokantaan.

ID	TC3.4
Nimi	Onnistunut pistetietojen nouto
Ennakko-vaatimukset	Käyttäjä on kirjautunut sisään sovellukseen Käyttäjä on syöttänyt järjestelmään College Board -tunnukset
Kuvaus	Haetaan hakijoiden pistetietoja CSV-muodossa College Boardin järjestelmästä.
Toivottu tulos	College Boardista saadaan CSV-muodossa hakijoiden pistetietoja.

ID	TC3.5
Nimi	Virheellinen College Board -tunnus
Ennakko-vaatimukset	Käyttäjä on kirjautunut järjestelmään
Kuvaus	Syötetään virheellinen College Board -käyttäjätunnus ja tarkistetaan, että sovellus käsittelee virheen oikein.
Toivottu tulos	Sovellus ilmoittaa virheellisestä tunnuksesta.

ID	TC3.6
Nimi	Hakijan Opintopolku- ja College Board -profiilien yhdistäminen
Ennakko-vaatimukset	Opintopolusta on haettu hakijan tiedot College Boardista on haettu hakijoiden SAT-pistetietoja
Kuvaus	Yritetään yhdistää hakijan Opintopolku- ja College Board -profiilit käyttäen tunnisteena College Board id:tä tai sähköpostiosoitetta, jotta SAT-pistetiedot saataisiin vietyä Opintopolkuun.
Toivottu tulos	Hakijan profiilit saadaan onnistuneesti yhdistettyä. Tietokantaan saadaan merkittyä hakijan SAT-pistetiedot.

4. Sovelluksen täytyy pystyä lähettämään Opintopolkuun hakijan pistetiedot, jotka saadaan CollegeBoardista

ID	TC4.1
Nimi	Hakijan pistetietojen onnistunut vieminen Opintopolkuun
Ennakko-vaatimukset	Hakijan pistetiedot on saatu haettua College Boardin järjestelmästä
Kuvaus	Viedään Opintopolun järjestelmään hakijan tietojen perusteella oikealle hakemukselle SAT-pistetiedot.
Toivottu tulos	Hakijan SAT-pistetiedot on onnistuneesti viety Opintopolkuun.

5. Sovelluksen täytyy pystyä tekemään loppuraportti (esim. txt-tiedostomuodossa) tuloksista, kuten onnistumisista ja epäonnistumisista, ja antaa se käyttäjän tarkasteltavaksi.

ID	TC5.1
Nimi	Onnistunut loppuraportin luominen
Ennakko-vaatimukset	Sovellus on suorittanut prosessit loppuun, kuten pistetietojen hakemisen ja viemisen.
Kuvaus	Odotetaan, että sovellus muodostaa raportin prosessin onnistumisista ja epäonnistumisista.
Toivottu tulos	Sovellus antaa käyttäjän luettavaksi raportin, jossa on listattuna jokaisen hakijan kohdalla status, ja tarvittaessa lisätietoja.

College Boardia koskevissa testitapauksissa yhdistettiin kaksi vaatimusta niiden samankaltaisuuksien vuoksi. Joitain Opintopolkua koskevia testitapauksia myös lisättiin, sillä Opintopolun järjestelmille ei tehty konkreettisia vaatimuksia. On otettava huomioon, että oikeilla käyttäjätiedoilla testaaminen on hyvin riskialtista, minkä takia esimerkiksi College Boardista saaduilla tiedoilla ei ole hyvä lähteä testaamaan sovelluksen toimintaa. Opintopolun järjestelmien puolella on testattava sovellusta Opintopolun testiympäristössä, ettei oikeita käyttäjätietoja vahingossakaan mennä muokkaamaan testaamisen aikana.

6 Lopuksi

Opinnäytetyön alkuperäisenä tavoitteena oli suunnitella, kehittää ja julkaista sovellus, joka noutaa korkeakouluhakijoiden SAT-pistetietoja, yhdistää hakijan Opintopolku- ja College Board -profiilit toisiinsa sekä vie hakijoiden pistetiedot Opintopolkuun sen jälkeen, kun profiilit on saatu yhdistettyä. Aikarajoitteiden vuoksi kuitenkin sovittiin, että opinnäytetyönä tehtäisiin vain tämän sovelluksen suunnitelma. Tämä mahdollisti laajan ja laadukkaan suunnitelman luomisen, minkä pohjalta sovellusta on helppo lähteä kehittämään jatkossa.

Aivan ensimmäisenä, ennen suunnittelun alkua, luotiin vaatimusmäärittely toimeksiantajalta saadun tehtävänannon perusteella. Suunnitteluvaiheen päättymisen jälkeen vaatimusmäärittelyssä huomattiin, ettei se täysin kattanut kaikkia suunnittelun vaiheita, varsinkaan Opintopolun kohdalla. Vaatimusmääritelmää luotaessa ei ollut paneuduttu College Boardin tai Opintopolun järjestelmiin ollenkaan, ja vaatimukset luotiin ainoastaan tehtävänannon perusteella. Tämän takia ei osattu arvioida Opintopolun järjestelmien laajuutta mikä johti siihen, ettei sille annettu tarpeeksi huomiota vaatimusmääritelmää luotaessa. Kun testitapauksia luotiin vaatimuksien perustella, Opintopolun järjestelmiä testaavalle testitapaukselle ei ollut sopivaa paikkaa. Jatkossa vaatimusmäärittelyä luotaessa olisi hyvä tutustua aluksi niihin järjestelmiin, joiden kanssa sovellus tulee kommunikoimaan. Näin vaatimuksista saataisiin tarkempia, ja ne käsittäisivät sovelluksen prosessit paremmin. Kaikesta huolimatta vaatimusmäärittely kuitenkin antoi hyväksyttävät puitteet sovelluksen suunnitteluun.

Sovelluksen suunnitteluvaiheessa lähdettiin aluksi tutkimaan sovelluksessa käytettäviä järjestelmiä, eli College Boardin ja Opintopolun ohjelmointirajapintoja. College Boardin rajapintaa tutkittaessa kävi ilmi, ettei sen puolelta tarjottu minkäänlaista testausympäristöä. Näin ollen rajapinnan testaaminen ei ollut mahdollista, jättäen kaikenlaiset API-kutsut tehtäväksi toteutusvaiheessa, oikeilla henkilötiedoilla. Toimeksiantajan toimittaman CSV-tiedoston perusteella College Boardista tulevia tietoja pystyttiin kuitenkin käsittelemään jonkin verran. Opintopolun API osoittautui odottamattoman suureksi. Hankaluuksia sen kanssa loi etenkin API:n dokumentaation epäselvyys, sillä monet päätepisteet tarjosivat testikäytössä vain vanhaa tietoa. Jotkin päätepisteistä eivät toimineet ollenkaan. Suunnitteluvaiheesta huolimatta on edelleen hieman epäselvää, mikä olisi kaikista tehokkain tapa toteuttaa tarvittavien tietojen haku

Opintopolun järjestelmästä. Asiaa hankaloitti myös se, että Eduuni-sivuston ohjeistukset oli ripoteltu eri puolille sivustoa, ja osa tiedosta oli vanhentunutta. Opintopolun API-päätepisteisiin olisi voinut paneutua hieman enemmän, sillä suunnitelmassa käydään läpi ainoastaan autentikaatioprosessi ja yksi esimerkkipäätepiste.

Suunnitteluvaiheessa pohdittiin myös oikeanlaista tapaa yhdistää hakijan Opintopolku- ja College Board -profiilit toisiinsa. Päädyttiin tulokseen, että yhdistämiseen on parasta käyttää sekä College Board id:n että sähköpostiosoitteen avulla tehtävää identifikaatiota. Tämän jälkeen käytiin läpi viisi eri vaihetta, joihin sovelluksen toiminnan pystyy jakaa. Nämä osat suunnittelusta onnistuivat hyvin, ja niitä on helppo seurata sovellusta kehitettäessä.

Suunnitteluvaiheen jälkeen selvitettiin sitä, miten sovellusta lähdetään suunnittelun jälkeen kehittämään. Tähän selvitykseen kuului sovellukseen käytettävät työkalut ja teknologiat, ominaisuuksien kehitysjärjestys sekä pohdinta sovelluksen julkaisusta kehityksen jälkeen. Käytettävien työkalujen ja teknologioiden kanssa olisi voinut käyttää enemmän kriittistä ajattelua: miksi kyseiset teknologiat valittiin ja olisiko sopivampia tai tehokkaampia teknologioita ollut tarjolla. teknologiat valittiin oman mieltymykseni ja kokemukseni perusteella. Valinnat eivät ole huonot, mutta parempia ratkaisuja olisi voinut olla tarjolla.

Lopuksi käytiin läpi sovelluksen testausta vaatimusmäärittelyn perusteella. Testitapauksia luotiin niistä vaatimuksista, jotka olivat saaneet määrittelyssä luokituksen *must have* tai *should have*. Käyttöliittymiä koskevia vaatimuksia ei otettu huomioon. Testitapauksia tehtiin sekä onnistumisien että epäonnistumisien kannalta, mutta tässäkin kohdassa olisi voinut ajatella ”laatikon ulkopuolelta”, sillä on varmasti lisää tapauksia, joissa ohjelman suoritus voisi esimerkiksi epäonnistua tai joissa voi ilmentyä virheitä.

Jatkossa sovellusta lähdetään kehittämään tässä opinnäytetyössä luodun suunnitelman pohjalta Turun ammattikorkeakoulun hakijapalveluille. Tavoitteena on luoda sovellus, joka toimii luotettavasti, turvallisesti ja tehokkaasti niin, ettei hakijapalveluiden henkilökunnan tarvitsisi enää käydä käsin muokkaamassa hakijoiden SAT-pistetietoja Opintopolun järjestelmässä.

Lähteet

[1] ProductPlan, ”What is MoSCoW Prioritization? | Overview of the MoSCoW Method,” n.d.. [Online]. Available: https://www.productplan.com/glossary/moscow-prioritization/. [Haettu 14.05.2024].

[2] College Board, ”API References - Higher Ed Reporting Portal Help,” n.d.. [Online]. Available: https://satsuite.collegeboard.org/help-center/higher-ed-reporting-portal/web-service/api-reference. [Haettu 21.05.2024].

[3] College Board, ”Downloading Files Via Web Service - Higher Ed Reporting,” n.d.. [Online]. Available: https://satsuite.collegeboard.org/help-center/higher-ed-reporting-portal/web-service/download-files. [Haettu 21.05.2024].

[4] Apereo Foundation, ”About CAS | Apereo,” n.d.. [Online]. Available: https://www.apereo.org/projects/cas/about-cas. [Haettu 22.05.2024].

[5] antsierla ja L. von der Hagen, ”Rajapintojen autentikaatio - Opintopolun palvelukokonaisuus - Eduuni-wiki,” Eduuni, 17.08.2023. [Online]. Available: https://wiki.eduuni.fi/display/ophpolku/Rajapintojen+autentikaatio. [Haettu 27.05.2024].

[6] M. Väistö, ”Kutsujan tunnisteen (caller-id) lisääminen rajapintakutsuihin - Opintopolun palvelukokonaisuus - Eduuni-wiki,” Eduuni, 12.11.2020. [Online]. Available: https://wiki.eduuni.fi/pages/viewpage.action?pageId=176867280. [Haettu 27.05.2024].

[7] KirstenS, ”Cross Site Request Forgery (CSRF) | OWASP Foundation,” OWASP foundation, n.d.. [Online]. Available: https://owasp.org/www-community/attacks/csrf. [Haettu 28.05.2024].

[8] P. Sutelainen ja O. Kivipelto, ”CSRF-tokenin lisääminen HTTP-kutsuihin - Opintopolun palvelukokonaisuus - Eduuni-wiki,” Eduuni, 26.11.2021. [Online]. Available: https://wiki.eduuni.fi/pages/viewpage.action?pageId=195149584. [Haettu 28.05.2024].

[9] M. Bärlund, ”REST-rajapintojen kuvaukset - Opetushallituksen palvelukokonaisuus - Eduuni-wiki,” Eduuni, 12.10.2022. [Online]. Available: https://wiki.eduuni.fi/display/OPHPALV/REST-rajapintojen+kuvaukset. [Haettu 28.05.2024].

[10] Opintopolku, ”Swagger UI,” n.d.. [Online]. Available: https://virkailija.opintopolku.fi/valintalaskentakoostepalvelu/swagger-ui/index.html#/%2Fsession. [Haettu 28.05.2024].

[11] Amazon Web Services, ”HTTP vs HTTPS - Difference Between Transfer Protocols - AWS,” n.d.. [Online]. Available: https://aws.amazon.com/compare/the-difference-between-https-and-http/. [Haettu 30.05.2024].

[12] W3Schools, ”TypeScript Introduction,” n.d.. [Online]. Available: https://www.w3schools.com/typescript/typescript_intro.php. [Haettu 03.06.2024].

[13] The Node.js Project, ”Node.js — About Node.js®,” n.d.. [Online]. Available: https://nodejs.org/en/about. [Haettu 03.06.2024].

[14] GeeksForGeeks, ”Express.js Tutorial - GeeksForGeeks,” 15.04.2024. [Online]. Available: https://www.geeksforgeeks.org/express-js/. [Haettu 03.06.2024].

[15] IBM, ”What Is MongoDB | IBM,” n.d.. [Online]. Available: https://www.ibm.com/topics/mongodb. [Haettu 03.06.2024].

[16] Wikipedia, ”React (JavaScript library) - Wikipedia,” n.d.. [Online]. Available: https://en.wikipedia.org/wiki/React_(JavaScript_library). [Haettu 03.06.2024].

UML-sekvenssikaavio