Tampereen ammattikorkeakoulu

Tekoälyn käyttö rajapintatestauksen tukena.

Alina Kauppila

OPINNÄYTETYÖ
Kesäkuu 2025

Tietotekniikan tutkinto-ohjelma
Tietoliikennetekniikka ja tietoverkot

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma
Tietoliikennetekniikka ja tietoverkot

KAUPPILA, ALINA:
Tekoälyn käyttö rajapintatestauksen tukena

Opinnäytetyö 31 sivua, joista liitteitä 1 sivu
Kesäkuu 2025

Tämän opinnäytetyön tavoitteena oli kehittää olemassa olevaa rajapintojen testausmenetelmää hyödyntämällä tekoälypohjaisia ratkaisuja. Työssä tarkastellaan kahta erillistä tekoälytyökalua, joita vertaillaan sekä toisiinsa että projektissa käytettyyn testausmenetelmään. Vertailun avulla pyritään selvittämään, voiko tekoälyn käyttö tehostaa testausprosessia säilyttäen samalla testitulosten selkeyden ja luotettavuuden.

Opinnäytetyössä tarkasteltiin erilaisia rajapintatyyppejä, painottaen erityisesti sitä, miten rajapinnat on toteutettu ja rakennettu Broker-projektin yhteydessä. Työssä analysoitiin lisäksi niitä työkaluja ja testausmenetelmiä, joita kyseisessä projektissa on hyödynnetty. Osana kokonaisuutta perehdyttiin myös tekoälyn perusteisiin sekä sen tarjoamiin mahdollisuuksiin erilaisten ohjelmistotestauksen osa-alueiden tukena.

Koska tekoälyn hyödyntäminen ohjelmistotestauksessa on vielä melko uusi ilmiö, työssä kartoitettiin myös tekoälyn mahdollisia vaikutuksia testauksen tulevaisuuteen. Tekoälyn käyttö yleistyy nopeasti monilla eri aloilla, joten on tärkeää pohtia, mitä se voi tarkoittaa myös testausprosessien kannalta tulevaisuudessa. Työssä pohdittiin tekoälyn käytön hyötyjä, mahdollisia haasteita sekä siihen liittyviä tietoturvaan ja luotettavuuteen liittyviä kysymyksiä.

Asiasanat: rajapinta, tekoäly, testaus

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree programme in ICT engineering
Telecommunications and Networks

KAUPPILA, ALINA:
The Use of Artificial Intelligence to Support API Testing

Bachelor's thesis 31 pages, appendices 1 page
June 2025

The goal of this thesis was to improve an existing API testing method by using AI-based tools. The study looked at two different AI tools and compared them to each other and to the testing method used in the project. The goal was to find out if using AI can make the testing process more efficient while keeping the results clear and reliable.

The thesis also examined different types of interfaces, focusing especially on how they have been built in the Broker project. It also reviews the tools and testing methods used in that project. As a part of the study, basic information about artificial intelligence was introduced, especially how AI can support different parts of software testing.

Since AI is still quite a new tool in software testing, the thesis also looked at what kind of role AI might play in the future of testing. As AI becomes a more common tool in many areas, it’s important to think about how it could affect testing processes. The thesis also discussed the benefits and challenges of using AI, and also security and reliability concerns that may come with it.

Key words: artificial intelligence, api, testing

SISÄLLYS

LYHENTEET JA TERMIT

API	Application Programming Interface
CLI	A command line interface
CRUD	Create, read, update and delete
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
LLM	Large language model
Pseudodata	Keinotekoinen data, joka jäljittelee oikeaa dataa
REST	Representational state transfer
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
UI	User Interface
USB	Universal Serial Bus
XML	Extensible Markup Language

1 JOHDANTO

Opinnäytetyön tarkoituksena on tarkastella tekoälyn madollisuuksia rajapinta toteutusten testausprosesseissa, sekä vertailla sen käyttöä perinteisiin testausmenetelmiin. Työssä perehdytään aluksi erilaisiin rajapintoihin niiden ominaisuuksiin ja erityisesti niiden testausmenetelmiin. Rajapintojen osalta syvennytään erityisesti API-rajapintoihin, jotka mahdollistavat sovellusten ja palveluiden välisen vuorovaikutuksen ja tiedonsiirron painottaen erityisesti sitä, miten rajapinnat on toteutettu Broker-projektin yhteydessä.

Testauksen osalta tutustumme lyhyesti erilaisiin rajapintojen testausmenetelmiin ja pohdimme, miten testien luomisessa ja ajamisessa olisi mahdollista hyödyntää tekoälyä. Tarkoituksena on luoda kattava yleiskuva tekoälyn soveltamisesta rajapintatestaukseen, keskittyen erityisesti sen mahdollisuuksiin ja rajoituksiin. Tavoitteena ei kuitenkaan ole toteuttaa testausta täysin tekoälypohjaisesti, vaan selvittää, miten se voi täydentää perinteisiä menetelmiä. Tekoälyn hyödyntämisessä testauksen apuna kiinnitetään erityisesti huomiota testitapausten luomiseen, testien päivittämiseen ja virheiden kattavaan havaitsemiseen. Tekoälyn hyödyntämistä testausprosessissa käsitellään käytännön demon kautta, jossa tarkastellaan kahden erilaisen tekoälyratkaisun vaikutusta, vertaillaan tuloksia toisiinsa, sekä alkuperäiseen ei-tekoälypohjaiseen versioon.

Tekoälyä hyödynnetään yhä enemmän tietotekniikan alalla, niin ohjelmoinnissa, kuin ohjelmistotestauksessa, johon myös rajapintatestaus sisältyy. AI:n avulla on mahdollista testauksen osalta mm. tunnistaa ongelmia, kerätä ja tuottaa testidataa ja luoda automaattisesti uusia testitapauksia ja päivittää niitä. Näiden ominaisuuksien avulla pyritään nopeuttamaan testausprosessia ja parantamaan testauksen laatua. Tekoälytestaukseen sisältyy sekä etuja, että mahdollisia haasteita. Tekoälyavusteisen testauksen eduiksi voidaan katsoa muun muassa tehokkuus, kattavuus ja suurten testidata määrien käsittely lyhyessä ajassa. Tämän kaltaisen testauksen riskeiksi voi nostaa esimerkiksi tekoälyn tuottamien tulosten ymmärrettävyyden, tulosten kattavuuden ja laadun, sekä tietoturvakysymykset.

Lopuksi työssä pohditaan tekoälyn ja perinteisten testausmenetelmien yhdistämistä. Tavoitteena on arvioida, miten tekoäly voi tukea ohjelmistotestausta ja pohtia, miten se muuttaa testauksen käytäntöjä tulevaisuudessa. Rajapintojen vakaus ja laatu ovat kriittisiä monille sovelluksille, joten tekoälyn hyödyntäminen voisi tarjota mahdollisuuksia näiden testauksen tehostamiseen ja datan käsittelyyn. Samalla arvioidaan, miten mahdollisia rajoitteita ja haasteita voidaan hallita.

Opinnäytetyö on tehty toimeksiantona Netum Oy:lle. Netum Oy on IT-alan yritys, joka toteuttaa laajasti monipuolisia digitaalisia ratkaisuja ja erilaisia IT-palveluita sekä julkisen sektorin organisaatioille että yksityisille yrityksille. Netum panostaa vahvasti vastuullisuuteen ja asiakaslähtöisiin ratkaisuihin projekteissaan. Yrityksellä on useita toimipisteitä eri puolilla Suomea, ja se työllistää lähes 400 IT-alan asiantuntijaa.

2 RAJAPINNAT

Tietotekniikassa rajapinnat ovat tärkeä osa ohjelmistokehitystä. Niiden avulla eri järjestelmät ja sovellukset voivat keskustella keskenään, eli siirtää tietoa tai käynnistää toimintoja toistensa välillä. Rajapinnat ovat mukana melkein kaikessa digitaalisten palveluiden toiminnassa. Rajapinnat voi ajatella eräänlaisena käyttöliittymänä järjestelmien välillä. Esimerkiksi, jos verkkopalvelu haluaa hakea käyttäjätietoja taustajärjestelmästä, se lähettää pyynnön rajapinnan kautta. (Chen, M. 2025.)

2.1 Mitä on rajapinnat

Rajapinnat ovat keskeinen osa järjestelmien välistä yhteistyötä. Niiden tehtävänä on toimia linkkinä tai välittäjänä, ja varmistaa yhteensopivuus kahden kokonaisuuden välillä. Hyvin suunnitellut ja toteutetut rajapinnat tekevät ohjelmistojen yhdistämisestä ja kehittämisestä tehokkaampaa ja auttavat pitämään kokonaisuuden hallittavana, vaikka taustalla pyörisi monimutkaisempikin järjestelmä. Kun järjestelmien välinen viestintä on selkeästi määritelty, voidaan eri osia päivittää tai kehittää ilman, että koko järjestelmää tarvitsee rakentaa uudestaan.

Kuten yleisesti ohjelmistoja kehittäessä, myös rajapintojen tulee olla mahdollisimman yksinkertainen ja helposti ymmärrettävä. Monimutkaiset rajapinnat voivat monesti johtaa virheisiin ja huonoon käytettävyyteen. Käyttöön tulisi valita selkeät ja yhtenevät nimet, vakiintuneet tietotyypit ja loogiset rakenteet. (Bloch, J. 2006.)

Rajapintoja on useisiin erilaisiin käyttötarkoituksiin. Näitä ovat muun muassa käyttöliittymärajapinta eli UI, joka on ihmisen ja järjestelmien välinen rajapinta, jonka avulla käyttäjä voi olla vuorovaikutuksessa ohjelmiston kanssa. Toisena esimerkkinä voi mainita laiterajapinnan, joka määrittelee, miten eri laitteet kommunikoivat keskenään. Tämä sisältää protokollat kommunikaatiolle, jos tarkoituksena on esimerkiksi välittää tietoa eri laitteistojen, kuten USB-tikun ja tietokoneen välillä. Kolmantena esimerkkinä rajapinnoista on ohjelmistorajapinta, eli API, johon tässä työssä erityisesti keskitytään. Sen tarkoituksena on mahdollistaa datan

välittäminen useamman eri ohjelman tai sovelluksen kesken. Tämän tekniikan avulla voidaan käytännössä liittää eri palveluita yhteen.

2.1.1 Ohjelmistorajapintojen toiminta

Ohjelmistorajapinnat eli API:t ovat keskeisessä asemassa ohjelmistokehitystä, jossa erilaisten ohjelmistojen on tarkoitus kommunikoida keskenään. API:t määrittävät ohjelmistojen välisessä kommunikaatiossa sen, mitä tietoja tarvitaan ja mitä vastauksia odotetaan. API:n etuna on myös se, ettei kyseessä olevien järjestelmien sisäisiä rakenteita tarvitse välttämättä tuntea. (Postman Inc. 2025a.)

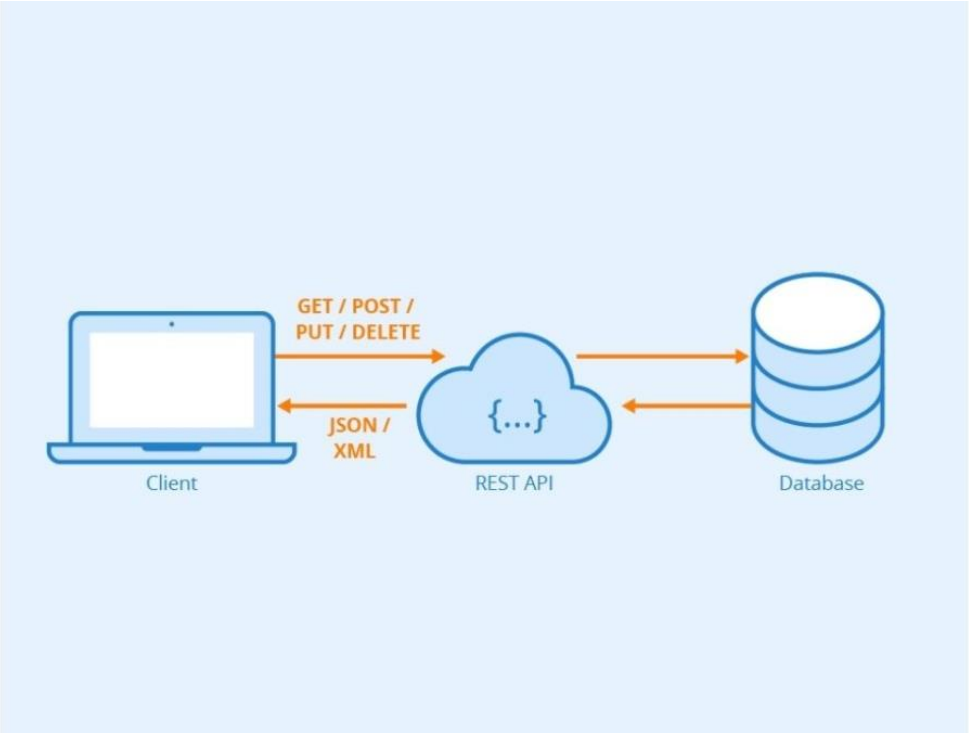
Järjestelmien kommunikointi API:n kautta tapahtuu pyyntö- ja vastaussyklin kautta. Käyttäjä lähettää pyynnön API-palvelimelle esimerkiksi painiketta napsauttamalla tai kirjoittamalla hakusanan, jonka jälkeen API-palvelin todentaa ja validoi syöttötiedot, sekä hakee ja palauttaa halutun datan käyttäjälle toiselta sovellukselta (kuva 1). Vastaus sisältää käyttäjän pyytämän sisällön, tai vaihtoehtoisesti virheviestin, mikäli jokin on prosessin aikana mennyt pieleen. (Postman Inc. 2025a.)

KUVA 1. Esimerkki API rajapinnan toiminnasta (Kuva: Nick, The Iron.io. 2020).

API arkkitehtuurityylejä on useampia, joista yleisimmin käytettyjä ovat muun muassa SOAP, WebSocket ja REST. SOAP on protokolla, joka käyttää XML:ää mahdollistamaan viestien siirron eri protokollien kuten HTTP tai TCP yli Sillä on verrattain hyvät suojausominaisuudet, mutta on monesti monimutkaisempi ja raskaampi kuin muut arkkitehtuurit. WebSocket on puolestaan protokolla, joka mahdollistaa kaksisuuntaisen reaaliaikaisen viestinnän asiakkaan ja palvelimen välillä yhden jatkuvan yhteyden kautta. Tällaista arkkitehtuurityyliä käyttävät esimerkiksi monet chat- ja pelisovellukset. (Amazon Web Services. n.d.)

REST on suosituin ja nykyään yleisimmin käytetty arkkitehtuuri resurssien ja palveluiden käyttämiseen verkossa. Se hyödyntää olemassa olevia web-standardeja, erityisesti http-protokollaa, resurssien hakemiseen, luomiseen, päivittämiseen ja poistamiseen. REST API:n tärkein ominaisuus on tilattomuus. Tilattomuus parantaa järjestelmän skaalautuvuutta ja luotettavuutta, sillä jokainen pyyntö on itsenäinen, eikä palvelin tallenna tietoa aiemmista pyynnöistä, kuten asiakastietoja, pyyntöjen välillä. (Richardson, L. & Amundsen, M. 2013.)

REST API:t käyttävät niin sanottuja CRUD-operaatioita. CRUD on lyhenne neljälle funktiotyypille, joita ovat Create, Read, Update ja Delete. Käyttäjä suorittaa näitä pyyntöjä palvelimelle HTTP metodien, kuten GET, POST, PUT ja DELETE avulla (kuva 2). GET-metodi hake olemassa olevan resurssin ilman, että sen tilaa muutetaan, kun taas POST luo uuden resurssin käyttäjän antaman datan pohjalta. PUT-metodi mahdollistaa jo olemassa olevan resurssin korvaamisen tai päivittämisen ja DELETE poistaa kyseessä olevan resurssin pysyvästi. HTTP-protokollien selkeän semantiikan ansiosta REST rajapinnat ovat helposti ymmärrettäviä, että laajasti hyödynnettävissä eri ohjelmistokielissä ja ympäristöissä. (Nick. 2020)



KUVA 2. Esimerkki REST API:n HTTP metodien toiminnasta (Kuva: Nick, The Iron.io. 2020).

2.1.2 Broker ja sen tarkoitus

Broker on Netumin rakentama pilvipohjainen integraatioratkaisu, jonka tarkoituksena on siirtää arvokasta liiketoimintadataa automaattisesti, reaaliaikaisesti sekä tietoturvallisesti uuden ja vanhan tietojärjestelmän välillä. Broker-ratkaisun suurimpana etuna on sen suoma mahdollisuus ottaa käyttöön uusia järjestelmiä vaiheistetusti siten, että tarvittuja tietoja vanhasta järjestelmästä voi myös hyödyntää tehokkaasti, mikä edesauttaa uusien järjestelmien hallittua kehitystä, sekä datan säilyvyyttä. Broker vastaanottaa kutsun järjestelmältä, validoi ja konvertoi datan kohdejärjestelmän mukaiseksi ja välittää sen eteenpäin. Tällaisen ratkaisun avulla integroitavan järjestelmän ei tarvitse tuntea teknistä vastapuoltaan, ja on voitu mm. minimoida inhimillisten virheiden ansiosta johtuvat virheet, sekä tehostaa tiedonsiirtoa, jotta voitaisiin vähentää manuaalisen työn tarvetta, eikä näin ollen käyttäjänkään tarvitse tuntea integroitavien järjestelmien yksityiskohtia. Broker noudattaa Netumin Legacy to Digi® -filosofiaa, jonka mukaan on tärkeä pyrkiä hyödyntämään jo olemassa olevia toimivia teknologioita ja menetelmiä, ja rakentamaan uutta toteutusta näiden ympärille, jotta voisi välttää hyvien elementtien hukkaan heittämistä sekä mahdollistamaan siirtymän järjestelmästä toiseen vaiheittain. (Netum. 2022.)

2.2 Rajapintojen testausmenetelmät

Rajapinnat mahdollistavat järjestelmien vuorovaikutuksen keskenään, joten niiden testaus on välttämätöntä, jotta voidaan taata ohjelmistojen toimivuus, suorituskyky ja turvallisuus. Testaus on tärkeää pitää mukana koko ohjelmiston kehityksen kaaren ajan, jotta virheet tunnistettaisiin jo varhaisessa vaiheessa, voitaisiin välttää riskejä tuotantoon vietäessä ja yleisesti nopeuttaa kehitysprosessia. Rajapintojen testaukseen on monenlaisia eri menetelmiä ja työkaluja erilaisiin tarpeisiin. Rajapintojen testaukseen kuuluu muun muassa toiminnallinen testaus, suorituskykytestaus, tietoturvatestaus ja regressiotestaus. (Global App Testing. 2024.)

2.2.1 Yleisiä testausmenetelmiä rajapinnoille

Rajapintojen testaaminen alkaa yleensä toiminnallisella testauksella, jonka tarkoituksena on varmistaa, että API toimii määriteltyjen vaatimusten mukaisesti. Testauksessa tarkistetaan, että oikeilla syötteillä saadaan oikeat vastaukset ja että virheelliset syötteet käsitellään asianmukaisella tavalla. Tämä on perustana kaikille muille testausmenetelmille ja varmistaa rajapinnan oikeanlaisen perustoiminnallisuuden. (Podutwar, P. 2025.)

Suorituskykytestauksessa puolestaan arvioidaan rajapinnan toimintaa erilaisissa kuormitustilanteissa, eli kykyä käsitellä suuria määriä samanaikaisia pyyntöjä. Suorituskykytestauksen alle sijoittuu sekä kuormitustestaus, että stressitestaus. Kuormitustestauksessa simuloidaan tyypillistä käyttöä ja arvioidaan rajapinnan suorituskykyä eri kysyntätasoilla. Stressitestauksessa tarkastellaan sitä, miten käyttöliittymä käyttäytyy äärimmäisissä olosuhteissa, kuten erityisen vilkkaan liikennöinnin tai simuloidun kyberhyökkäyksen aikana sekä järjestelmän kykyä palautua normaaliksi kuormituksen jälkeen. (Global App Testing. 2024.)

Tietoturvatestaus on tärkeä osa testausprosesseja, ja erityisesti API-testausta, sillä rajapinnat altistuvat herkästi ulkopuolisille hyökkäyksille. Rajapintojen tietoturvatestauksessa varmistetaan muun muassa autentikoinnin ja autorisoinnin oikeanlainen toiminta, sekä tarkastellaan sitä, miten rajapinta reagoi mahdollisiin tietoturvauhkiin. Parhaimmillaan tietoturvatestit paljastavat rajapintatoteutuksesta haavoittuvuuksia, jolloin voidaan ehkäistä näiden aiheuttamia haittoja ennen kuin ohjelmisto päätyy tuotantoon. (Global App Testing. 2024.)

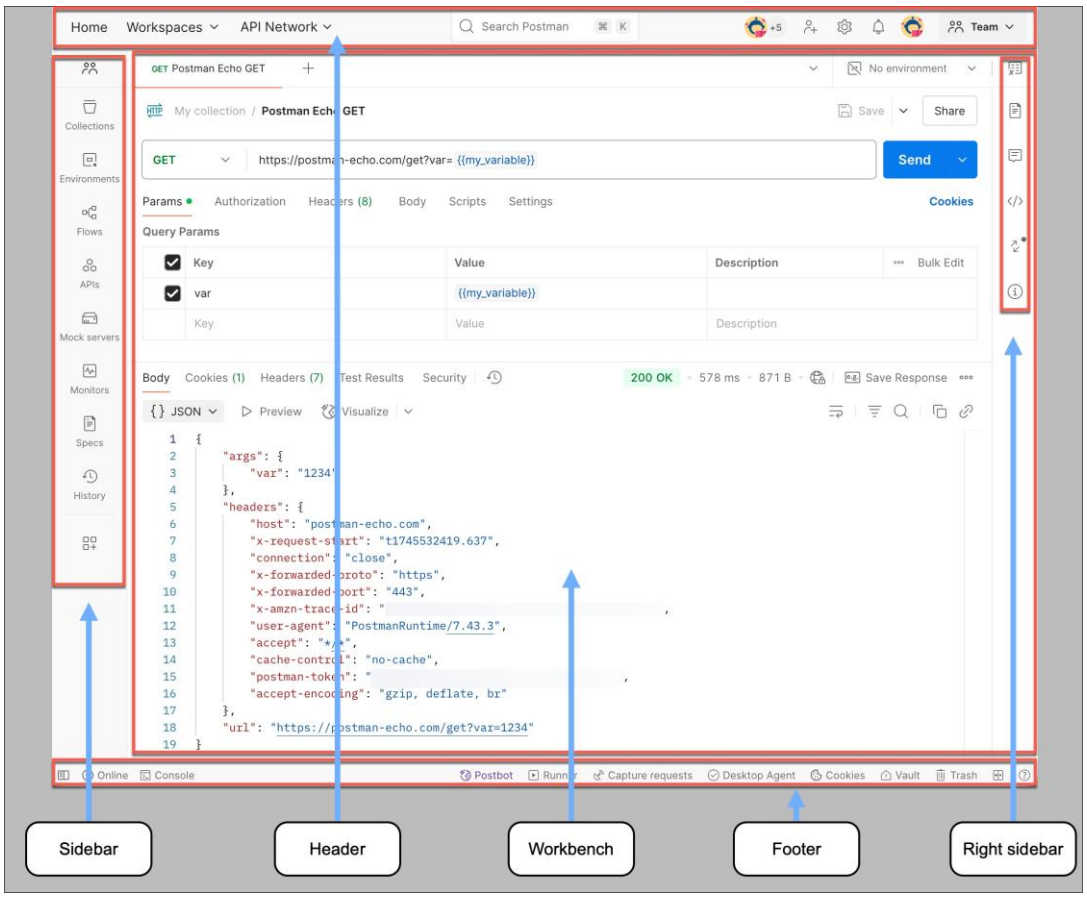
Regressiotestauksessa tavoitteena on varmistaa, että uudet ominaisuudet tai päivitykset eivät ole rikkoneet jo olemassa olevaa toiminnallisuutta. Regressiotestaus on hyvä suorittaa aina jokaisen muutoksen jälkeen, sillä tämä auttaa ylläpitämään toimintaa eri versioiden välillä ja on erityisen tärkeää varsinkin nopeissa julkaisusykleissä. Erilaisia regressiotestauksen tyyppejä käytetään riippuen siitä, millaisia ja kuinka laajoja muutoksia ohjelmiston koodiin on tehty. Yksittäisten komponenttien sisällä tehdyissä muutoksissa on esimerkiksi perusteltua käyttää osittaista regressiotestausta, jossa tarkastellaan vain niitä alueita, joihin muutokset ovat vaikuttaneet. Isompien päivitysten jälkeen olisi hyvä suorittaa

täydellinen regressiotestaus, jossa testataan koko sovellus, jotta voitaisiin varmistua, ettei päivitys ole rikkonut mitään järjestelmän osaa. (BrowserStack. 2025.)

2.2.2 Rajapintatestauksen työkalut

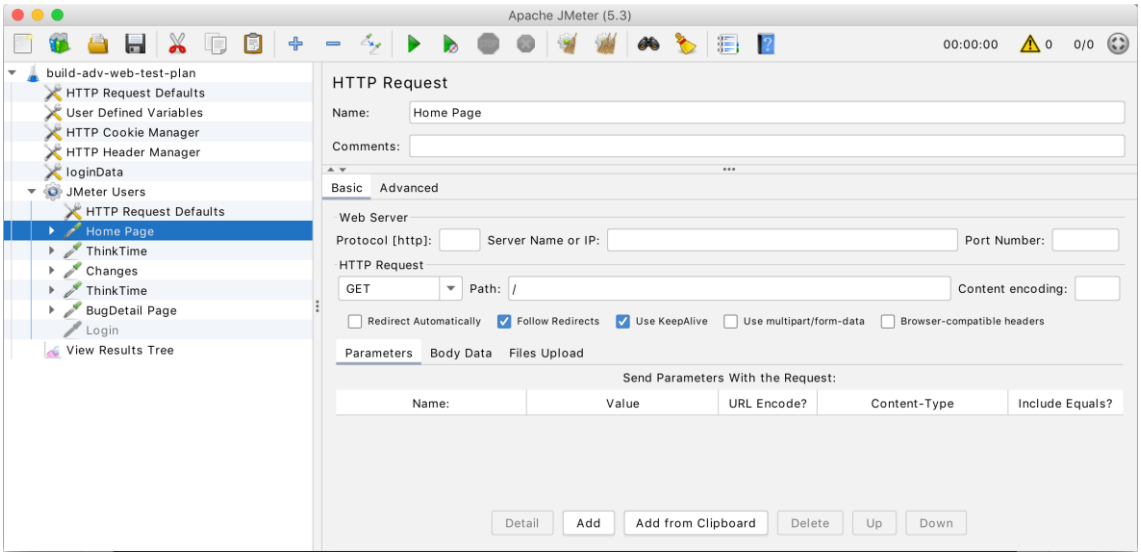
Kuten jo yllä todettiin, APIen testaus on hyvin laajaa ja monipuolista, näin ollen erilaisten rajapintatestien suorittamiseen on kehitetty monia työkaluja kaikkiin näihin tarkoituksiin. Työkalut mahdollistavat testien automaattisen ja manuaalisen suorittamisen, vastausten analysoinnin sekä käyttäjäkuormien simuloinnin. Monipuolinen testaustyökalujen käyttö auttaa havaitsemaan virheitä laajasti, sekä ylläpitämään ohjelmiston laatua. Näistä työkaluista muun muassa Postman ja JMeter ovat tunnettuja ja laajasti APIen testauksessa käytettyjä, joita myös käytettiin Broker projektin testauksessa. Postman ja JMeter muodostavat hyvän parin rakentamaan kattavaa testausstrategiaa.

Postman on REST API rajapintojen manuaaliseen ja automaattiseen testaukseen käytetyistä työkaluista yksi suosituimmista. Postmanilla on käyttäjäystävällinen ja selkeä graafinen käyttöliittymä, jonka avulla niin kehittäjät kuin testaajat pystyvät tuottamaan, ajamaan ja dokumentoimaan API kutsuja (kuva 3). Postmanin huomattavana etuna on sen helppokäyttöisyys, joka mahdollistaa sujuvan käytön sekä aloittelijoille, että kokeneemmille käyttäjille. Tämä työkalu myös tukee API dokumentaatiota ja testien jakamista tiimin sisällä työtiloissa, mikä auttaa tehostamaan testausprosessia. Postmanissa on mahdollisuus kirjoittaa testiskriptit JavaScriptillä ja näiden avulla lähettää HTTP pyyntöjä, tarkastella näiden pyyntöjen vastauksia. (Postman Inc. 2025b.)



KUVA 3. Kuvaus Postman työkalun käyttöliittymästä ja testiskriptistä (Kuva: Postman Inc, 2025c)

Apache JMeter on avoimen lähdekoodin Java pohjainen suorituskykytestauksen työkalu, joka soveltuu sekä kuormitus, että stressitestaukseen. JMeter on alun perin suunniteltu pääasiassa web sovellusten testaukseen, mutta on myös toimiva työkalu API rajapintojen testauksessa. JMeterin vahvuutena on sen kattava valikoima tilastointivaihtoehtoja ja graafeja, joilla voi koota testauksen tulokset sekä luettavaan, että visuaaliseen muotoon kätevästi yhteen HTML-raporttiin. Testit rakennetaan graafisessa käyttöliittymässä, ja näillä on mahdollista tehdä monimutkaisiakin simulaatioita useista samanaikaisista käyttäjistä, sekä analysoida, miten API tämän kuormituksen kestää (kuva 4). (Apache Software Foundation. 2025a.)



KUVA 4. Esimerkki JMeterin käyttöliittymästä (Kuva: Apache Software Foundation. 2025b)

2.2.3 Testaus Broker-projektissa

Broker projektissa testaustyökaluina käytettiin Postmania, sekä Apache JMeteriä. Pääasiallisena testaustyökaluna toimi Postman, jonka avulla suoritettiin regressiotestausta, jolla pyrittiin varmistamaan, että tiedot järjestelmien välillä pysyivät eheinä, eikä muutokset koodissa rikkoisi mitään jo toimivaa ratkaisua. Postmanin avulla myös pystyttiin varmistamaan, että tekstikenttiin syötetyt erikoismerkit toimivat oikein sekä uudessa, että vanhassa ympäristössä ja siirtyivät näiden välillä vaivatta.

Tyypillisessä testausprosessissa valitaan sopiva testiskripti, jonka avulla voidaan varmistaa, että kutsu täyttää määritellyt validointiehdot, ja että palvelin palauttaa odotetun vastauksen. Esimerkiksi testatessa sähköpostikenttää voidaan suorittaa HTTP POST-kutsu kohteena olevaan verkko-osoitteeseen, ja syötteenä annetaan sähköpostiosoite ilman @-merkkiä. Mikäli järjestelmässä on määritelty, että sähköpostiosoitteessa on oltava @-merkki, järjestelmän tulisi palauttaa virheilmoitus. Jos kuitenkaan virheilmoitusta ei synny, voidaan päätellä, että validointisääntö puuttuu, tai se ei toimi oikein. Tällöin olisi tarpeellista korjata järjestelmän koodia vastaamaan määrittelyn vaatimuksia. Kutsujen lisäksi testaajan tehtävänä oli käydä katsomassa järjestelmissä myös käyttöliittymän puolella, onko tiedot tallentuneet sinne asianmukaisesti.

Apache JMeter oli käytössä järjestelmien kuormitustestauksessa. Testiskriptit siirrettiin kätevästi yhtenä pakettina Postmanista JMeterille, ja tämän jälkeen luotiin Testplan, jossa määriteltiin millä testiskripteillä kuormitustesti haluttiin tehdä ja millaisilla kriteereillä kohdetta kuormitettiin. Testin luonnissa määriteltiin muun muassa, kuinka monta käyttäjää simuloidaan sekä kuinka usein ja kuinka monta pyyntöä käyttäjät lähettävät. Kuormitustestaus suoritettiin kolmelta eri työasemalta ja testausta seurattiin CLI käyttöliittymän kautta reaaliajassa sekä tarkkailtiin mahdollisia muutoksia kutsujen ja vastausten vasteajoissa. Testiajojen jälkeen JMeter kokosi tuloksista koosteen, joka sisälsi kattavan taulukoinnin kutsuista ja niiden vasteajoista, sekä testaustuloksia havainnollistavan pylväsdiagrammin. Kuormitustestausten tuloksista tuotettiin myös monesti raportti, joka välitettiin asiakkaalle.

3 TEKOÄLYN KÄYTTÖ TESTAUKSESSA

Tekoälytestauksessa pyrkimyksenä on hyödyntää tekoälyä kehittämään ja tehostamaan ohjelmistotestauksen eri osa alueita. Testauksen kannalta tekoälyä on käytetty muun muassa testitapausten luomisessa, testidatan keräämisessä ja analysoinnissa sekä testien ylläpidossa. Tekoälyn avulla testauksesta pyritään saamaan entistä nopeampaa ja tarkempaa, ja sen odotetaan vähentävän myös manuaalisen työn määrää, jotta testausprosessin laatu ja tehokkuus kasvaisi.

3.1 Yleistä tietoa tekoälyn käytöstä

AI eli tekoäly on tietojenkäsittelytieteen osa-alue, jossa tavoitteena on saada tietokonejärjestelmät tai ohjelmistot simuloimaan oppimista, reagointia, ongelmanratkaisua, päätöksentekoa ja jopa luovuutta ihmisen kaltaisesti.

Vaikka viimevuosina tekoälyn käyttö on yleistynyt monilla aloilla valtavasti, on sen kehitys kuitenkin vielä alkuvaiheessa, ja sen täydellisen toiminnan saavuttamiseksi menee vielä kauan. Tekoälyä hyödyntäviä teknologioita on käytössä muun muassa terveysalalla tukemassa potilastietojen analysointia sekä diagnosointia analysoimalla röntgen- tai magneettikuvia, liiketoiminnassa ja asiakaspalvelussa chatbottien ja virtuaalivirkailijoiden muodossa ja jopa liikenteessä itseajavien autojen ympäristöä analysoimassa ja tekemässä tämän perusteella ajopäätöksiä. (Putty. 2025.)

Tekoälyn rooli myös ohjelmistotestauksessa on kasvanut viimevuosina merkittävästi. AI:n käyttö testauksessa edistää testausprosessien tehostamisen, testitulosten tarkkuuden ja luotettavuuden kehitystä. Sillä tekoäly on kaiken kaikkiaan suhteellisen tuore työkalu myös testauksessa, on syytä tarkastella hieman sen tuomia hyötyjä ja mahdollisia riskejä. (Sharma. 2025)

3.1.1 Tekoälyn käytön hyödyt testausprosesseissa

Tekoälyn käyttö ohjelmistotestauksessa tuo mukanaan merkittäviä etuja, kuten tehokkuuden testien generoinnissa ja uudelleenajossa koodimuutosten jälkeen,

joka on perinteisesti tehty manuaalisesti. Tekoälypohjaiset testit voivat mukautua koodissa tehtyihin muutoksiin automaattisesti, mikä nopeuttaa testiskriptien päivittämistä, sekä varmistaa testien laadun ja näin ollen on suureksi avuksi testien ylläpidossa. Manuaalinen testiskriptien mukauttaminen muuttuneen sovelluksen mukaisiksi voi myös olla aikaa vievää ja altistaa inhimillisille virheille, kuten huolimattomuus- ja kirjoitusvirheille. Tekoälyä hyödyntävien työkalujen etuna on myös sen kyky automatisoida toistuvia ja yksinkertaisia tehtäviä kuten testien luontia ja niiden tulosten analysointia, jotta testaajat voisivat keskittyä monimutkaisempiin ja kriittisempiin testauskohteisiin. (Li. 2024)

Tekoälyn erityisenä etuna on sen kyky käsitellä suuria määriä dataa, joten se pystyy havaitsemaan laajasti testiskenaarioiden kattavuuden testattavassa ohjelmistossa. Puutteiden ilmaantuessa Tekoäly kykenee paikkaamaan näitä testaamatta jääneitä aukkoja luomalla älykkäitä testitapauksia, jotta testiskenaariot kattaisivat kaiken ohjelmiston kannalta oleellisen, eikä virheitä pääsisi huomaamatta tuotantoon. Tästä ominaisuudesta on erityisesti hyötyä silloin, kun järjestelmään tulee suuri määrä mahdollisesti monimutkaisiakin muutoksia. Suuren datankäsittelykyvyn avulla on myös mahdollista kerätä testaustuloksista saatua dataa tehokkaasti, ja koostaa näistä raportteja testaajien, kehittäjien ja asiakkaan tarpeiden mukaisesti. (Li. 2024)

Yhteenvetona voidaan todeta, että tekoälyn integroiminen osaksi testausprosessia edesauttaa nopeampaa testausta, parempaa virheiden tunnistusta, tehokkaampaa analysointia ja raportointia, sekä vapauttaa testaajia yksinkertaisemmista toistuvista tehtävistä monimutkaisempiin ja kriittisempiin testiskenaarioihin perehtymiseen.

3.1.2 Tekoälyn käytön haasteet testausprosesseissa

Tekoälyn integroiminen osaksi ohjelmistotestausta aiheuttaa monien hyvien puolien lisäksi myös haasteita. Vaikka tekoälyllä on potentiaalia tehostaa testausta merkittävästi, sen käyttöön liittyvät haasteet ovat myös tärkeä ottaa huomioon, ennen sen käyttöönottoa erityisesti kriittisissä ympäristöissä.

Yksi tekoälyn käytön suurimmista haasteista on saadun datan vaikutukset tekoälytyökalun toimintaan. Tekoäly vaatii suuria määriä dataa, josta se voisi oppia, mutta aina tällaista dataa ei ole välttämättä saatavilla, kuten esimerkiksi projektin alussa, kun toteutusta vasta suunnitellaan. Mikäli opetusdataa ei ole riittävästi tai se on virheellistä, se voi johtaa tekoälyn virheellisiin päätelmiin, joita voi olla hyvin vaikea havaita. Testaajien on myös tärkeä tietää millä perusteella ratkaisuja ja havaintoja tehdään, ja missä virheiden syyt piilevät. Tämä ei aina ole mahdollista tekoälyn läpinäkymättömän luonteen vuoksi, jossa monimutkaiset algoritmit tuottavat testituloksia ja tekevät päätöksiä sen mukaisesti. (Li. 2024)

Toinen merkittävä haaste liittyy erilaisten tekoälyratkaisujen ja työkalujen tuoreuteen. Kuten jo aiemmin on mainittu, tekoäly on kaiken kaikkiaan vielä uusi ja kehittyvä apuväline monella saralla. Tehokas ja oikeanlainen tekoälyn integroiminen projektiin ja käyttö vaatii ajantasaista osaamista ja koulutusta, jota ei kaikissa kouluissa tai yrityksissä ole vielä välttämättä saatavilla. Tekoälyn hyödyntäminen projekteissa oikeaoppisesti edellyttää sekä teknistä ymmärrystä, käytön hallintaa, että kykyä tulkita ja yhdistää tekoälyn tuottamia tuloksia oman testauksen tukena.

Tekoälyn käyttö ohjelmistotestauksessa herättää myös tietoturvaan liittyviä kysymyksiä. Tekoälyn kanssa toimiessa on oltava tarkkana, miten ja mihin sen antamia syötteitä käytetään. Mikäli malleja on koulutettu epäluotettavalla tai vanhentuneella datalla, riskinä on, että tekoälyn tuottama koodi sisältää tahattomasti haavoittuvuuksia, kuten SQL-injektioita tai vanhentuneita ja haitallisia paketteja. Testauksessa käytetyn datan kanssa on myös syytä olla huolellinen, sillä etenkin pilvipohjaiset ja generatiiviset tekoälymallit saattavat tallentaa syötettyä sensitiivistä dataa ja pahimmillaan myös tuottaa sen pohjalta vastauksia siten, että arkaluontoinen tieto paljastuu. (Soundararadjalou. 2025)

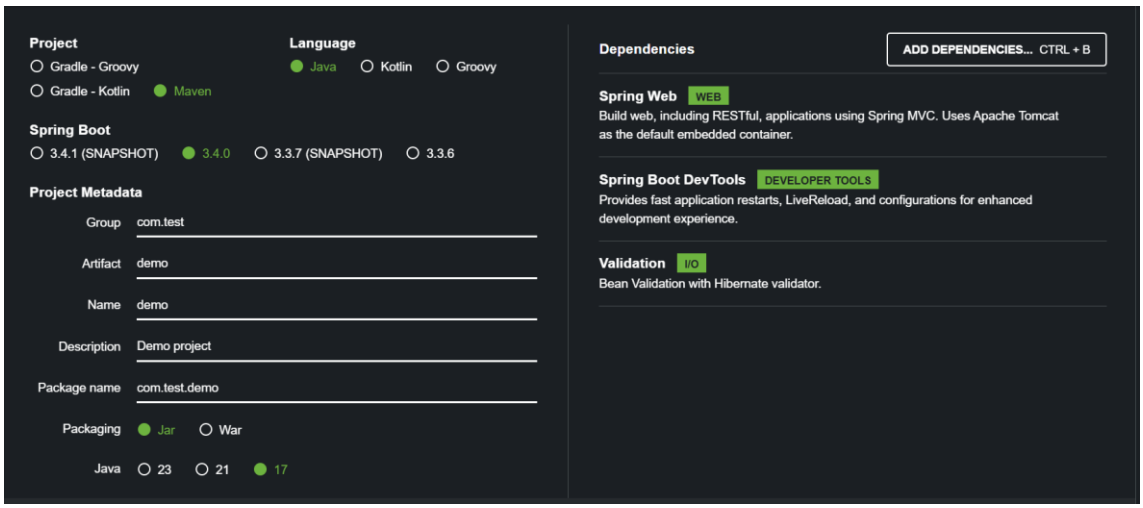
Näiden riskien minimoimiseksi, testauksessa olisi tärkeää käyttää pseudodataa aidon tuotannossa olevan datan sijasta, sekä käyttää koko prosessin ajan vain lokaaleja suljetussa ympäristössä toimivia tekoälyratkaisuja, jotta data ei pääsisi organisaation ulkopuolelle. Työkalua valitessa pitää myös tarkistaa, tallentaako AI sille syötettyä tietoa, jakaako tämä sitä eteenpäin ja käyttääkö se annettua syötettä oppimiseen.

4 TEKOÄLYTESTAUKSEN SOVELTAMINEN BROKERIIN

Seuraavaksi päästään yhdistämään rajapintoja ja tekoälyllä testausta pienen demon muodossa. Demoympäristö on yksinkertainen perinteistä kirjautumissivua muistuttava toteutus, jossa käyttäjältä kysytään tietoja. Demo mukailee Broker toteutusta siten, että siinä on käytetty suurilta osin saman kaltaisia työkaluja ja metodeita kuin Brokerissakin.

4.1 Projektin alustus

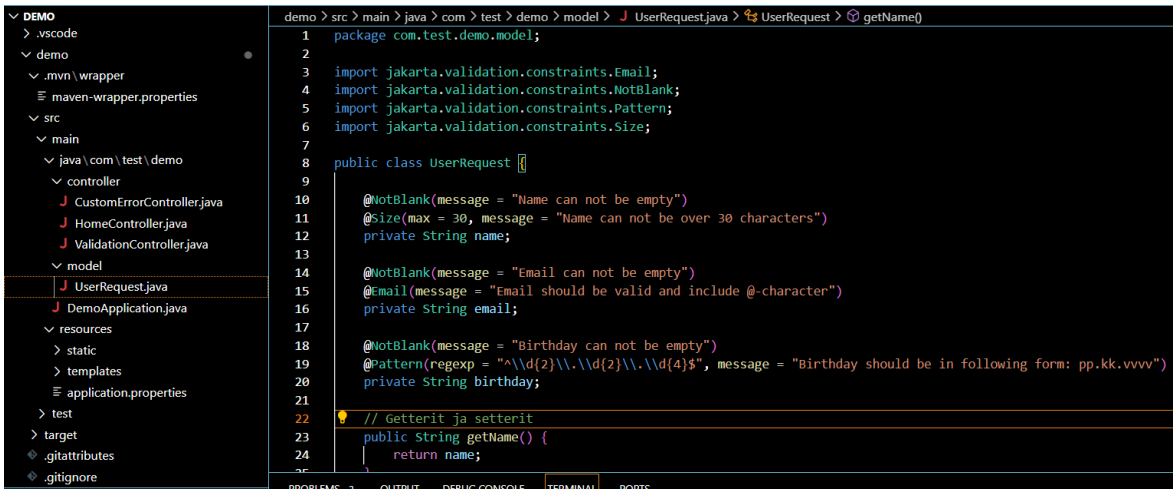
Projektissa tavoitteena oli rakentaa yksinkertainen validointiin keskittyvä API-rajapinta, joka käyttää Java-ohjelmointikieltä sekä Spring Boot -sovelluskehitystä. Rajapinnan tarkoituksena on vastaanottaa käyttäjän nimi, sähköpostiosoite ja syntymäaika, sekä tämän jälkeen validoida tiedot annettujen ehtojen mukaisesti. Toteutus sisälsi REST-periaatteiden mukaisen rakenteen, OpenApi-dokumentoinnin, eli Swaggerin sekä integroinnin Postmaniin testien suoritusta varten. Kehityksen aikana käytettiin myös monipuolisesti erilaisia ohjelmointi ja validointikirjastoja.



KUVA 5. Projektin aloitus luomalla Spring Boot -projekti (Kuva: Alina Kauppila. 2025)

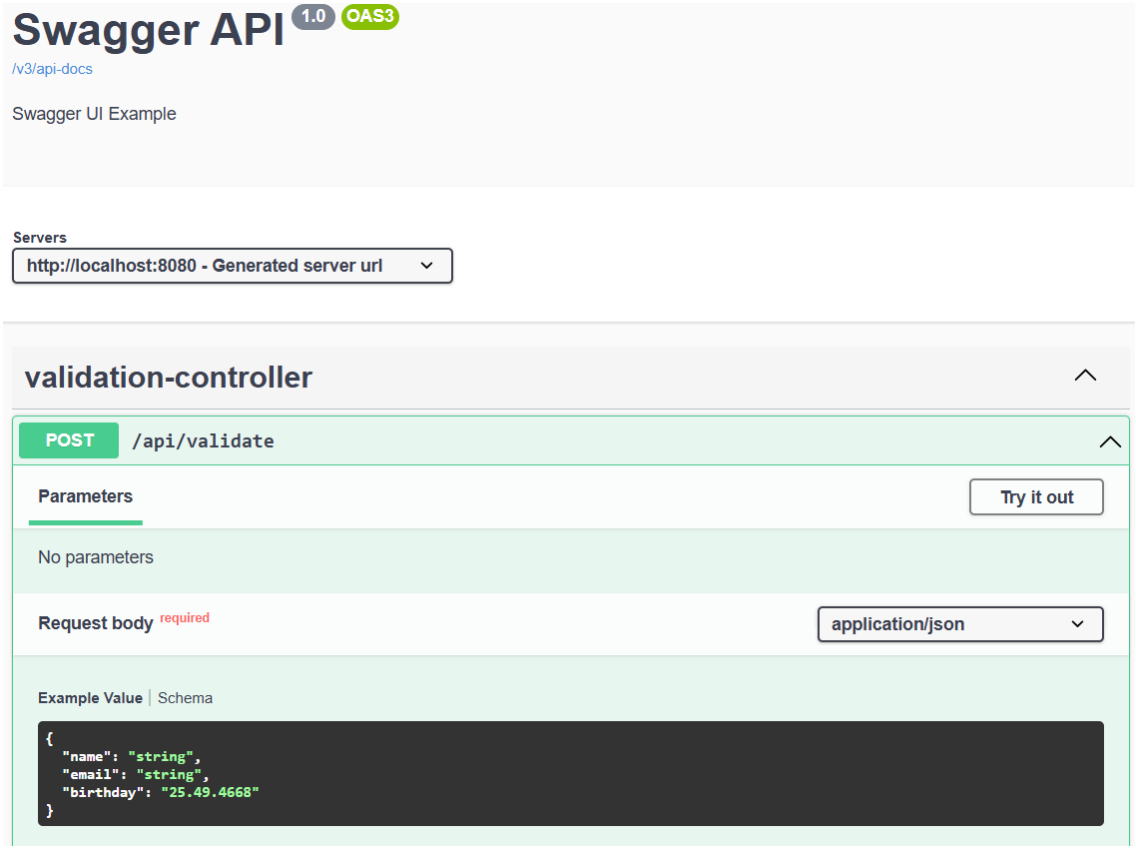
Kehitysympäristössä käytettiin Java 17 -versiota sekä Spring Bootin 3.4.0-versiota (kuva 5). Validoinnin toteutukseen käytettiin Spring Bootin Jakarta Validation API:a ja Hibrenate Validator -kirjastoa, jotka mahdollistavat määrittelyt suo-

raan sovelluslogiikassa, esimerkiksi sille, ettei nimi saa ylittää 30 merkkiä, sähköpostiosoitteen on sisällettävä ”@” -merkki, ja syntymäajan on oltava muodossa ”pp.kk.vvvv”. Projektissa rakennettiin yksittäinen REST kontrolleri, joka vastaanottaa HTTP POST -pyynnön JSON-muodossa, muuntaa sen ja validoi ennen käsittelyä. Sovellus ei sisällä erillistä käyttöliittymää vaan toimii täysin ulkoisten työkalujen, kuten Postmanin avulla.



KUVA 6. Projektin rakenne ja java tiedosto henkilön tietojen kyselyä varten kriteereineen. (Alina Kauppila. 2025)

Model paketin alta löytyy UserRequest, jossa on API:n varsinainen toiminnallisuus, eli käyttäjätietojen kysyminen, johon on myös liitetty validointia varten tarkistukset ja niihin liittyvät viestit (kuva 6). DemoApplicationin tarkoituksena taas on toimia sovelluksen käynnistyspisteenä, joka muun muassa lataa Spring-sovelluskehyksen ja käynnistää palvelimen komennolla ”mvn spring-boot:run”. Ilman tätä REST-rajapinta ei olisi käytettävissä. Controller-paketin alla olevat Java-tiedostot sisältävät pääasiassa validointipuolen. CustomErrorControllerin alla tarkistetaan HTTP-statuskoodi; mikäli statuskoodina on 404, palautetaan virheviesti ”Virhe: Resurssia ei löytynyt!” ja kaikille muille statuskoodista johtuville virheille on määritetty oletusviesti ”Virhe: Jotain meni pieleen!”. HomeController sisältää käytännössä vain etusivulle määritellyn tekstin, ja ValidationControllerissa tarkistetaan, että UserRequestissa määritellyt ehdot täyttyvät.



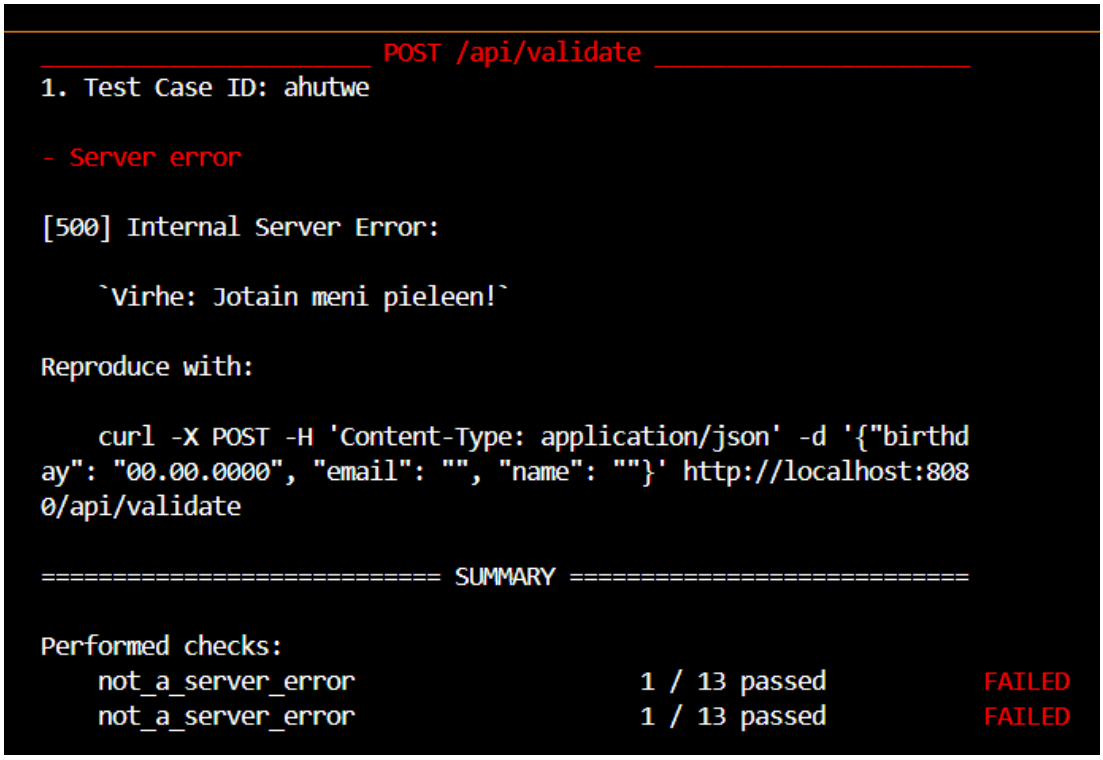
KUVA 7. Projektin Swagger käyttöliittymä. (Alina Kauppila. 2025)

Rajapinnan esittämiseen käytettiin Swaggeria, joka integroitiin projektiin Spring-Doc:in OpenApi UI -kirjaston avulla (kuva 7). Swaggerin avulla oli mahdollista testata ja tarkastella rajapintaa, ja sen välityksellä API oli mahdollista siirtää myös Postmanille varsinaista testausta varten. Kun ympäristö oli todettu toimivaksi, alettiin pohtia tekoälyominaisuuden lisäystä sovellukseen. Kriteereinä oli mahdollistaa testipatterien automaattinen luonti, mahdollisuus päivittää testejä muutosten ilmaantuessa, sekä se, että työkalun tulisi olla maksuton.

4.1.1 Schemathesis

Schemathesis on avoimen lähdekoodin automaattiseen API-testaamiseen tarkoitettu työkalu, joka perustuu OpenApi-spesifikaatioon. Se hyödyntää Hypothesis-kirjastoa tuottaakseen satunnaisia testisyötteitä rajapinnalle. Schemathesis ei käytä tekoälyä sanan varsinaisessa merkityksessä, mutta se testaa älykkäästi ja systemaattisesti erilaisia syötevariaatioita ja etsii poikkeamia, esimerkiksi odottamattomia statuskoodeja tai virheellisiä vastauksia.

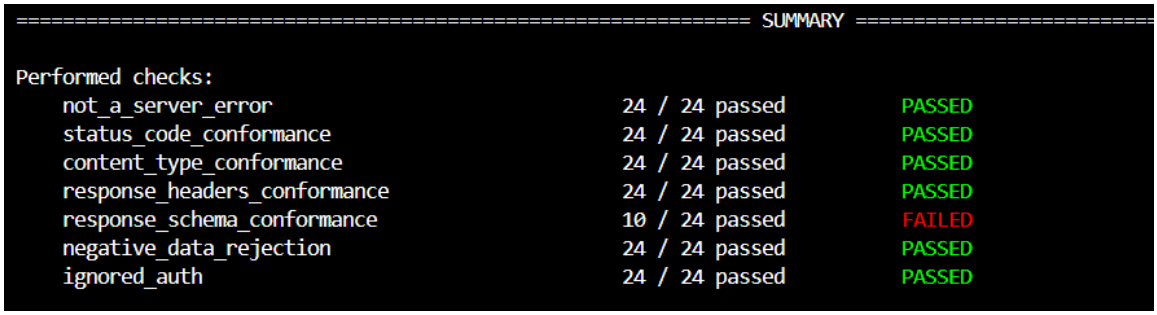
Schemathesis työkalun integroiminen projektiin oli varsin helppoa, sillä se oli asennettavissa vain yhdellä pip -komennolla. Kun työkalu on asennettu, sovelluksen voi käynnistää normaalisti, ja tämän jälkeen ajaa komento ”schemathesis run http://localhost:8080/v3/api-docs” jolloin Schemathesis lukee API:n osoitteesta löydetyt rajapintaoperaatiot. Tämän jälkeen työkalu lähettää API:lle sekä virheettömiä, että virheellisiä pyyntöjä, ja näin tarkistaa sen vakautta ja virheiden käsittelyä. Vastauksena Schemathesis tuottaa testiraportin, johon on koottu testauksen kohteet ja tieto siitä olivatko testit onnistuneita vai päättyivätkö virheeseen (kuva 8).



KUVA 8. Ote projektin Schemathesis ajon raportista. (Alina Kauppila. 2025)

Kuten yllä olevasta testiraportista voi huomata, Schemathesis raportoi vain palvelinvirheistä, vaikka manuaaliset testit Postmanilla palautti myös 200-statuskoodeja. Tämä johtui siitä, että OpenApi-dokumentaatiosta puuttui kuvaus siitä, mitä vastauksia palvelin odottaa, ja ilman näitä kyseinen työkalu ei osaa arvioida, sovelluksen toimivuutta oikein. Koska Schemathesis perustaa testinsä OpenApi-dokumentaatioon, ongelma oli ratkaistavissa sillä, että syötteen määrittelyitä tarkennettiin, sekä lisättiin controller metodiin annotaatiot, jotka dokumentoivat statuskoodit, ja mitä nämä tarkoittavat. Kun nämä muutokset oli tehty, Schemathesis

testasi automaattisesti sekä oikeita, että virheellisiä syötteitä ja tarkisti, että virheellinen syöte johtaa 400-koodiin ja oikeanlainen syöte 200-koodiin. Tämän jälkeen myöskään serverivirheitä ei enää ilmaantunut (kuva 9).



KUVA 9. Onnistunut testiajo muutosten jälkeen. (Alina Kauppila. 2025)

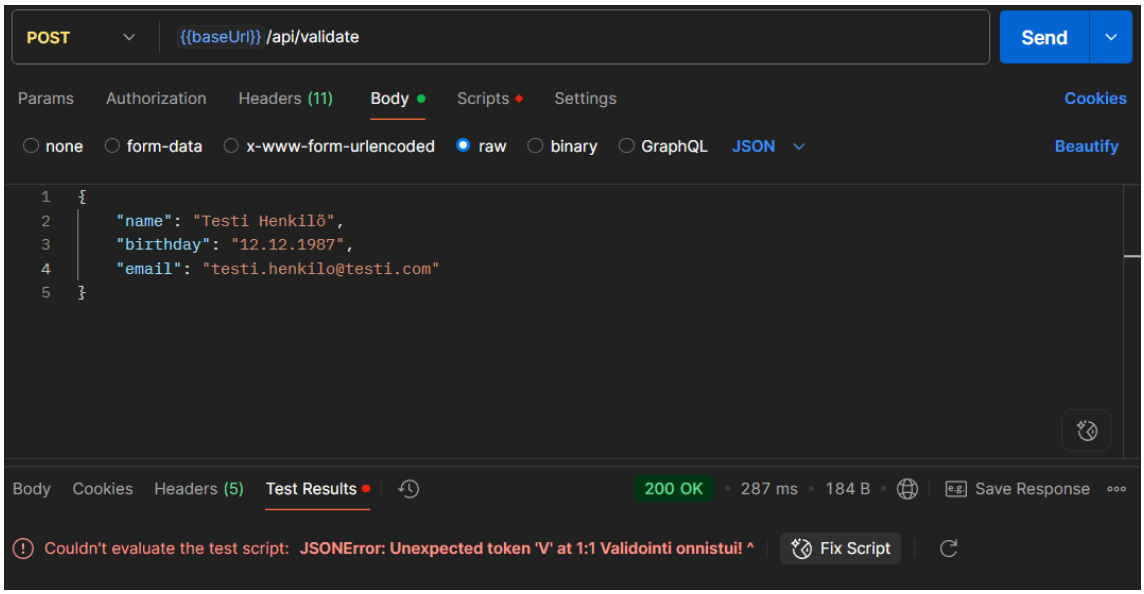
Yhteenvetona voi todeta, että työkalu edellytti todella huolellisesti dokumentoituja validointisääntöjä ja statuskoodien määrittelyä OpenApissa, johon se perustaa testinsä. Vaikka työkalu ei suoraan ilmoittanut, että määrittelyt eivät ole kunnossa, testin tulos paljasti suuria puutteita projektin dokumentaatiossa, joita ei olisi perinteisillä testausmenetelmillä välttämättä huomattu näin herkästi. Näin ollen Schemathesis ei ainoastaan testannut rajapinnan toimintaa, vaan toimi myös erinomaisena työkaluna dokumentaation laadun ja kattavuuden parantamiseen.

4.1.2 Postbot

Projektin kannalta oli oleellista tarkastella sitä, löytyykö jo jostain käytössä olevasta työkalusta tekoälyominaisuuksia, joita voisi hyödyntää, eikä näin ollen tarvitsisi asentaa ja integroida uusia työkaluja ja ympäristöjä monimutkaistamaan projektia. Postmanilta löytyikin tekoälyä hyödyntävä ominaisuus, Postbot, joka hyödyntää LLM-kielimalleja API-testauksessa ja dokumentoinnin automatisoinnissa. Postbotin avulla on mahdollista luoda testiskriptejä chat-tyyppisen ominaisuuden avulla luonnollisella kielellä kuvaillen testaustarvetta, ja kuvauksen perusteella tämä loisi JavaScript-koodin halutuista testeistä. (Kumar. 2023)

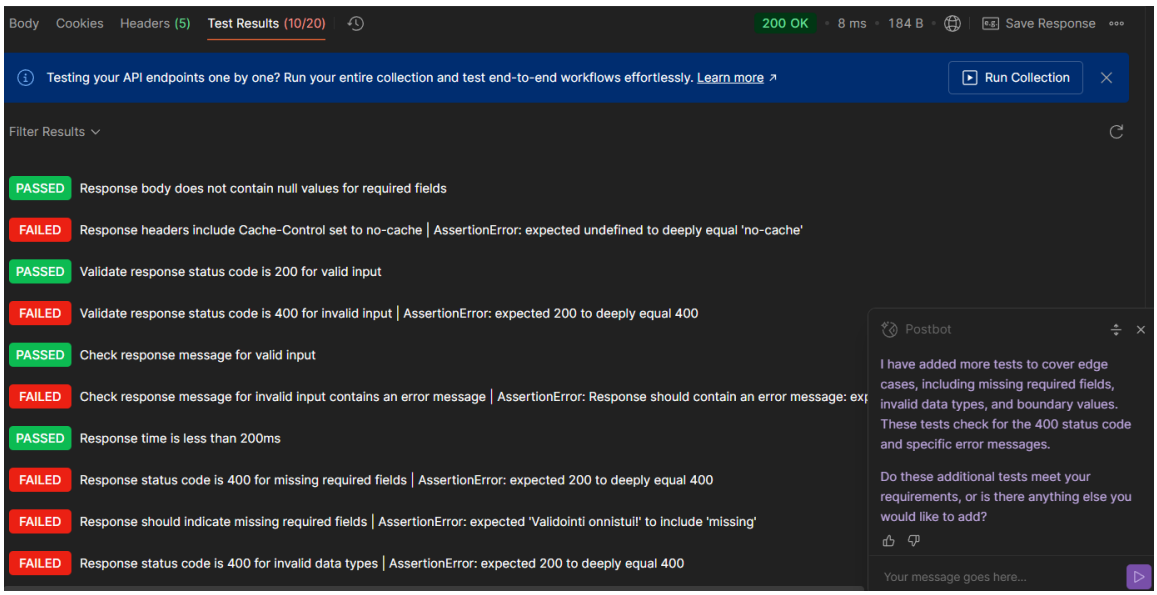
Postmaniin on luotu testausta varten hyvin yksinkertainen testiaihio, johon syötetään erilaisia variaatioita halutuista tiedoista, ja näiden vastausten perusteella varmistetaan, että oikeanlaiset syötteet antavat statuskoodin 200 ja vääränlaiset

statuskoodin 400. Myös Postmanilla testatessa virheellinen syöte antoi aluksi 500-statuskoodin, mutta tämä korjaantui samoilla muutoksilla, mitä Schemathesis:n kohdallakin tehtiin. Ensimmäinen testi ajettiin oikeanlaisella syötteellä ja tämä sai statuskoodin 200. Kuitenkaan testitulokset ei ollut nähtävillä JSON-parsintavirheen takia, joka aiheutuu siitä, että Postman yrittää tulkita tekstimuotoisen vastauksen JSON-muotoisena (kuva 10).



KUVA 10. Ilmaantunut parsintavirhe onnistuneen testin yhteydessä. (Alina Kauppila. 2025)

Tällaisen virheen kanssa tuli loistava tilaisuus käyttää Postbotin ”Fix Script”-ehdotusta, ja vain yhdellä klikkauksella Postbot muokkasi Scripts välilehden testiskriptejä siten, ettei tämä virhe enää ilmaantuisi. Kun korjaus oli tehty, siirryttiin määrittelemään erilaisia haluttuja testejä Postbotille. Seuraavaksi halutaan luoda testi, joka varmistaa, että virheellinen syöte palauttaa statuskoodin 400 ja oikea statuskoodin 200. Tämä onnistui yksinkertaisesti kirjoittamalla Postbotin chat kenttään kriteerit siitä, mitä kaikkea halutaan testata. Postbot generoi tämän perusteella jälleen Scripts tiedostoon uudet testiskriptit, jonka tuloksena saatiin testitulosten raporttiin arvokasta tietoa halutuista tapauksista. Postbotilla on myös itsessään valmiita ehdotuksia testin kehittämiselle, kuten ”Add more tests”, joka lisää automaattisesti testiskriptejä sellaisista tilanteista, jotka olivat jääneet lisäämättä alkuperäiseen testiin, kuten puuttuvat pakolliset kentät, virheelliset tietotyypit ja raja-arvot (kuva 11).



KUVA 11. Postbotin vastaus testien automaattiseen lisäämiseen, ja näiden testien tuloksia. (Alina Kauppila. 2025)

Vaikka monet testit näyttävät epäonnistuneen, tässä tapauksessa se kertoo siitä, että testi odottaa virhettä eli statuskoodia 400, ja kun tämä ei toteudu oikeassa muodossa annetun syötteen takia, tämä tarkistus päätyy erroriin. Postbotin valmiiden ehdotusten pohjalta on myös mahdollista valita spesifisti mitä osaa vastauksesta halutaan testata, kuten esimerkiksi sisältötyyppiä, jolloin Postbot luo nimenomaan tälle räätälöidyn testiskriptin ja ajaa sen.

Yhteenvetona Postbotin osalta voi todeta sen olevan nopea ja helppokäyttöinen tapa luoda perustason testiskriptejä Postman ympäristössä, eikä tämä vaadi käyttäjältään osaamista koodin tai skriptin kirjoittamisessa, sillä kaikki hoituu chat tyyppisen ominaisuuden kautta. Postbot antaa myös valmiita ehdotuksia mahdollisesti oleellisista testiskripteistä, mikä nopeuttaa manuaalista testausta ja luo kattavat testiskriptit API:n perustoiminnallisuuden varmistamiseksi. Postbotin rajoitteena kuitenkin on se, ettei se itse kuitenkaan generoi erilaisia testisyötteitä, vaan nämä pitää edelleen kirjoittaa manuaalisesti itse, minkä lisäksi testien tarkkuus riippuu suurilta osin käyttäjän antamista lähtötiedoista. Miinuksena oli myös se, että ilmaisversion käyttö on rajattua ainakin toistaiseksi.

5 POHDINTA

Projektissa tekoälyn hyödyntäminen API-testauksen apuna on mielenkiintoinen aihe, mutta toisaalta myös hieman haasteellinen. Ensimmäisenä huomio kiinnittyi sopivien AI-ominaisuuksia sisältävien työkalujen löytämisen haastavuuteen, sillä useat alun perin lupaavilta vaikuttaneet työkalut olivat joko muuttuneet merkittävästi, muuttuneet maksullisiksi tai poistuneet kokonaan käytöstä projektin suunnittelun ja toteutuksen aikana. Tähän vaikuttaa erityisesti tekoälyratkaisujen nopea kehitys, sekä se että monet ehdotetut ratkaisut olivat toki älykkäitä, kuten Schemathesis, mutta eivät varsinaisesti täyttäneet tekoälytoteutuksen kriteereitä. Toinen huomion arvoinen asia on, että projektin mittakaava oli hyvin pieni, ja työkalujen tuomat edut jäivät tästäkin syystä hieman rajallisiksi. Suuremmissa projekteissa tekoälyn kyky generoida ja automatisoida testejä korostuisi paremmin, kun käytettävissä olisi myös pilvipohjaiset agenttipalvelut ja suurempi budjetti toteutukselle.

Projektin alkuperäinen toteutus perustui manuaalisten testien kirjoittamiseen, jotta saataisiin varmuus siitä, että vääränlaiset syötteet aiheuttavat virheilmoituksen. Näiden testien generointia ja ajoa varten projektiin integroitiin Schemathesis. Schemathesis tarjosi tehokkaan tavan automatisoida testaus OpenApi-määrittelyiden pohjalta generoimalla ja suorittamalla testejä rajapinnan validointisääntöihin ja statuskoodeihin perustuen. Tämän avulla paljastui huomattavia puutteita dokumentaatiossa ja validointien tarkkuudessa. Vaikka Schemathesis ei varsinaisesti ole tekoälyä hyödyntävä ratkaisu, se kuitenkin generoi testejä ja raportoi testien tuloksia hyvin älykkäästi, sekä auttoi kehittämään testausprosessien ja dokumentaatioiden kattavuutta.

Postmanin Postbot puolestaan tarjosi nopean tavan generoida testiskriptejä jo olemassa olevan työkalun sisällä. Postbotin käyttö oli helppoa erityisesti siksi, että se osasi itse suositella parannusehdotuksia testiskripteihin ja antoi valmiita vaihtoehtoja siihen, millaisia testiskriptejä Apin testauksessa voisi käyttää. Vaikkakin Postbotin luomat testiskriptit olivat monipuolisia, testaus oli hyvin pinnallista, eikä se kuitenkaan automaattisesti generoinut API:lle lähetettäviä oikeita ja virheellisiä syötteitä, tai käsitellyt monimutkaisempia virhetilanteita niin kattavasti

kuin Schemathesis. Tämän perusteella voisi todeta, että Postbot sopii mainiosti pienten perusvalidointien testaukseen, mutta testattavan ympäristön ollessa monimutkaisempi ja testisyötteen määrän ollessa suurempi, työkalun hyötysuhde laskee.

Yhteenvetona voisi todeta, että sekä Schemathesis ja Postbot kumpikin ovat toimivia ratkaisuja etenkin tämän kaltaisissa pienissä projekteissa, ja voivat myös samassa ympäristössä käytettynä täydentää toisiaan ja näin olla apuna testaajalle. Työkalujen vertailusta ja ominaisuuksista on koostettu taulukko vertailun helpottamiseksi (Liite 1).

Tekoälyssä on potentiaalia API-testauksen saralla, mutta sen tehokas hyödyntäminen vaatisi vielä hieman työkalujen ja integraatiomahdollisuuksien kehittymistä. Tulevaisuudessa on odotettavissa, että tekoäly jatkaa kehitystään ja tämä kehitys näkyy myös API-testauksen puolella monipuolisempina ja älykkäämpinä testausprosesseina, mutta se vaatii sitä, että myös projekteissa käytetyt työkalut kehittyvät tukemaan koneoppimista ja tekoälyratkaisuja sisältäviä agentteja.

Vaikka tässä työssä tekoälyn käytön hyödyt jäivät suhteellisen pieniksi, sen potentiaali API-testauksen maailmassa on merkittävä, erityisesti testaajien asiantuntemuksen kehittyessä tekoälyn käytöstä ja kun työkalut kehittyvät ja kykenevät tuottamaan koko ajan monimutkaisempia testausprosesseja älykkäästi.

LÄHTEET

Amazon Web Services. n.d. What is an API (Application Programming Interface)? Verkkosivu. Viitattu 30.6.2025.
https://aws.amazon.com/what-is/api/

Apache Software Foundation. 2025a. Verkkosivu. Viitattu 2.6.2025.
https://jmeter.apache.org/

Apache Software Foundation. 2025b. Verkkosivu. Viitattu 2.6.2025.
https://github.com/apache/jmeter?tab=readme-ov-file

Bloch, J. 2006. How to Design a Good API and Why it Matters. Konferenssijulkaisu. Viitattu 30.6.2025.
https://dl-acm-org.libproxy.tuni.fi/doi/pdf/10.1145/1176617.1176622

BrowserStack. 2025. Regression Testing: A Detailed Guide. Verkkosivu. Viitattu 30.6.2025.
https://www.browserstack.com/guide/regression-testing

Chen, M. 2025. What Is an API (Application Programming Interface)? Verkkosivu. Viitattu 30.6.2025.
https://www.oracle.com/fi/cloud/cloud-native/api-management/what-is-api/

Richardson, L. & Amundsen, M. 2013. RESTful Web APIs. E-Kirja. Viitattu 30.6.2025.
https://learning.oreilly.com/library/view/restful-web-apis/9781449359713/

Global App Testing. 2024. What is Interface Testing And How To Conduct It? Blogi. Viitattu 30.6.2025.
https://www.globalapptesting.com/blog/interface-testing

Kumar, H. 2023. How to Use Postman’s Postbot for AI-Powered API Testing. Blogi. Viitattu 6.6.2025.
https://blog.binaryrepublik.com/2025/05/how-to-use-postmans-postbot-for-ai.html

Li, T. 2024. AI Based Testing: Benefits, Challenges, Best Practices and More. Blogi. Viitattu 4.6.2025.
https://www.headspin.io/blog/the-state-of-ai-in-software-testing-what-does-the-future-hold

Netum. 2022. KEHA-keskukselle räätälöitiin täsmäratkaisu tietojärjestelmien integrointiin. Verkkosivu. Viitattu 3.6.2025.
https://www.netum.fi/asiakkaat/keha-keskukselle-raataloitiin-tasmaratkaisu-tietojarjestelmien-integrointiin/

Nick, The Iron.io. 2020. What Is REST API? Blogi. Viitattu 2.6.2025.
https://blog.iron.io/what-is-rest-api/

Podutwar, P. 2025. REST API Testing: A Beginner's Guide With Best Practices. Verkkosivu. Viitattu 30.6.2025.
https://www.lambdatest.com/learning-hub/rest-api-testing

Postman Inc. 2025a. What is an API? Verkkosivu. Viitattu 30.6.2025.
https://www.postman.com/what-is-an-api/

Postman Inc. 2025b. What is Postman? Verkkosivu. Viitattu 30.6.2025.
https://www.postman.com/product/what-is-postman/

Postman Inc. 2025c. Navigating Postman. Verkkosivu. Viitattu 2.6.2025.
https://learning.postman.com/docs/getting-started/basics/navigating-postman/

Putty, C. 2025. Examples of Artificial Intelligence (AI) in 7 Industries. Blogi. Viitattu 4.6.2025.
https://www.thoughtful.ai/blog/examples-of-artificial-intelligence-ai-in-7-industries

Sharma, S. 2025. AI in Software Testing | What it is & How to use AI in Testing. Blogi. Viitattu 3.6.2025.
https://testsigma.com/blog/is-ai-really-important-in-software-test-automation/

Soundararadjalou, J. 2025. Security Risks Of AI In Software Development: What You Need To Know. Blogi. Viitattu 4.6.2025.
https://www.opsmx.com/blog/security-risks-of-ai-in-software-development-what-you-need-to-know/

LIITTEET

Liite 1. Taulukko käytettyjen työkalujen ominaisuuksista

	Työkalu	
Ominaisuus	Schemathesis	Postbot
Käyttää tekoälyä	EI	KYLLÄ
Automaattinen testien generointi	KYLLÄ	OSITTAIN
Testaa oikeita ja vääriä syötteitä	KYLLÄ	EI
Vaatii tarkan OpenApi kuvauksen	KYLLÄ	EI
Antaa palautetta ja korjausehdotuksia	EI	OSITTAIN
Käyttöliittymä	EI	KYLLÄ
Sopiva tämän kokoluokan projektiin	KYLLÄ	KYLLÄ
Aloittelijaystävällisyys	OSITTAIN	KYLLÄ
Tuottaa visuaalista dataa kuten graafeja	EI	KYLLÄ
Ilmainen	KYLLÄ	OSITTAIN