



TAMPERE UNIVERSITY OF TECHNOLOGY

VILLE VIRONMÄKI
YHTENÄINEN WEB-OPPIMISYMPÄRISTÖ OHJELMOINNIN
JOHDANTOKURSSILLE

Diplomityö

Tarkastaja: apulaisprof. Petri Ihantola
Tarkastaja ja aihe hyväksytty Tieto-
ja sähkötekniikan tiedekuntaneuvoston
kokouksessa 17.08.2016

TIIVISTELMÄ

VIRONMÄKI, VILLE: Yhtenäinen Web-oppimisympäristö ohjelmoinnin johdantokurs-
sille

Tampereen teknillinen yliopisto

Diplomityö, 69 sivua

Lokakuu 2016

Tietotekniikan koulutusohjelma

Pääaine: Pervasive Systems

Tarkastaja: apulaisprof. Petri Ihantola

Avainsanat: automaattinen arviointi, ohjelmoinnin opetus, palvelupohjaisuus

Ohjelmoinnin massakursseilla käytetään usein erilaisia teknisiä työkaluja. Näitä ovat esimerkiksi oppimisenhallintajärjestelmät ja automaattista palautetta antavat työkalut. Nämä ovat kuitenkin monesti toisistaan irrallisia, mikä aiheuttaa oppilaille usein hämmennystä ja ylimääräistä työtä.

Tässä diplomityössä vastaamme kysymyksiin: ”Miten pystymme rakentamaan yhtenäisen ohjelmoinnin oppimisympäristön? Mitä osia sen pitää sisältää ja miten näiden tulisi kommunikoida keskenään?”. Kysymyksiin vastataan tapaustutkimuksen kautta rakentamalla uusi oppimisympäristö Tampereen teknillisessä yliopistossa pidettävälle opintojaksolle Johdatus ohjelmointiin. Rakentamisen lähtökohtana on, että ympäristössä käytetään oppimisenhallintajärjestelmänä Aalto-yliopistossa kehitettyä A+:aa.

Aluksi tutkimme vaatimuksia, joita opintojakso kohdistaa ympäristölle, ja määrittelemme komponentit, joista ympäristö rakentuu. Komponenttien tunnistamisen jälkeen hahmottelemme ympäristölle arkkitehtuurin käyttäen apuna sovellettua 4+1-arkkitehtuurimallia ja tarkoitukseen sopivia tekniikoita, kuten REST. Tämän jälkeen tutustumme kirjallisuudesta löytyviin komponenttitoteutuksiin ja valitsemme niistä sopivimmat.

Työn tuloksena on yhtenäinen oppimisympäristö, joka rakentuu A+-oppimisenhallintajärjestelmän ympärille. A+:aan on toteutettu REST-rajapinta, jonka kautta se kommunikoi oppilaan IDE:n ja ympäristön muiden komponenttien kanssa. Oppilaat pääsevät kaikkiin ominaisuuksiin käsiksi yhden kirjautumispisteen kautta ja mm. voivat palauttaa tehtävät suoraan IDE:stä. A+:aan tehdyt palautukset siirtyvät Mooc-grader-nimiselle komponentille, joka tarkistaa ne automatisoiduilla testeillä ja palauttaa tulokset A+:aan.

Toteuttamamme oppimisympäristö helpottaa oppilaiden työskentelyä edelliseen kurssilla käytettyyn järjestelmään verrattuna. Enää opiskelijan ei tarvitse kirjautua järjestelmästä toiseen, eikä etsiä tehtävien palautuspaikkoja, vaan hän voi palauttaa tehtävän IDE:stä napin painalluksella.

ABSTRACT

VIRONMÄKI, VILLE: Uniform Web-based learning environment for introductory programming course

Tampere University of Technology

Master of Science Thesis, 69 pages

September 2016

Master's Degree Programme in Information Technology

Major: Pervasive Systems

Examiner: Assistant Professor Petri Ihantola

Keywords: automatic assesment, programming teaching, service-orientation

Various technical tools like automatic assesment and learning management systems are often used in programming courses for large student groups. However, tools are often interconnected which causes confusion and extra work for students.

In this thesis, we answer the following questions: "How can we build a uniform programming learning environment? Which software components are needed to implement the system and how these components should communicate with each other?". We answer these questions by conducting a case study where we develop a new learning environment to be used in the introductory programming course at the Tampere University of Technology. We build our environment around a learning management system called A+, originally developed at Aalto University.

In this thesis, we will first examine the requirements. Then, we define the components that build up an environment. After defining the components, we sketch an architecture for the environment by using applied 4+1 architecture model with technologies fitting for purpose, like REST. After this, we familiarize ourselves with components already existing, identified from the literature. Finally, we choose the most convenient ones as the basic building blocks of our system.

As a result of our thesis, we have a uniform learning environment built around the A+ learning management system. A+ has a REST interface which it uses to communicate with student's IDE and with the rest of components of the environment. Students get access to every feature via the single sign-up and i.a. can submit exercises through IDE. Submissions done to A+ are transferred to the component called Mooc-grader which inspects them with automated tests, and returns the results to A+.

The learning environment we have developed eases students' working compared the previous system used in the course. Student has neither longer to log in to various systems nor search submission locations of the exercises, but submit the exercise via IDE by just pressing the button.

ALKUSANAT

Tämä diplomityö on tehty Tampereen teknillisen yliopiston tietotekniikan laitoksen opetuksenkehittämishankkeen yhteydessä helmikuusta 2016 syyskuuhun 2016. Työtä on tehty yhteistyössä Aalto-yliopiston Learning+Technology -tutkimusryhmän kanssa. Tästä ryhmästä haluan kiittää erityisesti Teemu Lehtistä ja Jaakko Kantojärveä neuvonannosta A+-järjestelmään liittyen.

Haluan esittää suuret kiitokset apulaisprofessori Petri Ihantolalle työn ohjauksesta ja korvaamattomista kommenteista. Lisäksi kiitokset Essi Isohannille kommenteista ja korjauksista. Työn kielellinen asu olisi ollut paljon huonompaa ilman näitä.

Tampereella 20. syyskuuta 2016

Ville Vironmäki

SISÄLLYSLUETTELO

1. Johdanto	1
1.1 Ohjelmointi	1
1.2 Ohjelmoinnin opettaminen	2
1.3 Johdatus ohjelmointiin -opintojakso	2
1.4 Tutkimusongelma ja sen ratkaisu	4
1.5 Diplomityön rakenne	5
2. Käyttäjien tarpeet ja järjestelmän vaatimusmäärittely	6
2.1 PyCharm	7
2.2 Kurssisivut	7
2.2.1 Kurssitiedot- ja aikataulut	7
2.2.2 Oppimateriaali	8
2.2.3 Tehtävänannot	8
2.3 Repolainen	8
2.3.1 Automaattinen arviointi	9
2.3.2 Manuaalinen arviointi	9
2.4 Moodle	10
2.4.1 Oppimisenhallintajärjestelmä	10
2.4.2 Sosiaalinen kanssakäyminen	11
2.5 Uudet ominaisuudet	11
2.5.1 Ohjelmavisualisointi	11
2.5.2 Vertaisarviointi	12
2.5.3 Etenemisen hahmottaminen ja pelillistäminen	12
2.5.4 Tentti	12
2.5.5 Oppimisen seuranta	13
2.6 Yhteenvedo yhdistettävistä komponenteista	13
2.6.1 Komponenttien käyttötapaukset	13
2.6.2 Komponenttien yhdistäminen	14
3. Arkkitehtuuriratkaisut	16
3.1 Looginen näkökulma	17
3.1.1 KnowledgeTree	17
3.1.2 Järjestelmä loogisesta näkökulmasta	18
3.2 Prosessinäkökulma	19
3.2.1 REST	19
3.2.2 LTI-protokolla	20
3.2.3 GLUE!	22
3.2.4 PETCHA	23

3.2.5	Tin Can API	25
3.2.6	Järjestelmä prosessinäkökulmasta	25
3.3	Fyysinen näkökulma	26
3.3.1	Test My Code	26
3.3.2	Mailing it in	28
3.3.3	Järjestelmä fyysisestä näkökulmasta	28
4.	Katsaus olemassaoleviin järjestelmiin ja komponentteihin	30
4.1	Oppimisenhallintajärjestelmät	30
4.2	Automaattinen arviointi	31
4.3	Manuaalisen palautteen antaminen	32
4.4	Vertaisarviointi	35
4.5	IDE	35
4.6	Visualisointi	37
4.7	Oppimisen seuranta	39
4.8	Tehtävien ja materiaalin arkistointi	40
4.9	Sosialisointi	40
4.10	Tentti	41
4.11	Lopullinen arkkitehtuuri	42
5.	Järjestelmän toteutus	44
5.1	A+:-n REST-rajapinta	44
5.1.1	Autentikaatio ja auktorisointi	44
5.1.2	Rajapintamäärittely	45
5.2	PyCharm-plugin	49
5.2.1	Kurssirakenteen haku	50
5.2.2	Tehtävän palautus	52
5.3	Mooc-grader:in tarkistusskriptit	53
5.4	Yhteenveto	55
6.	Toteutuksen arviointi	56
6.1	Yhtenäisyys	56
6.2	Osien etsintä	56
6.3	Kommunikointitapa	57
6.3.1	Richardsonin kypsyysmalli	57
6.3.2	Microsoftin REST-ohjeistus	58
6.4	Yhteenveto	60
7.	Yhteenveto ja jatkokehitys	61
	Lähteet	63

1. JOHDANTO

Nykypäivänä yhä useampi käyttämämme laite on ohjelmoitavissa. Yhteiskuntamme liikkuu yhä enenevin määrin kohti tietoyhteiskuntaa, jossa nämä laitteet ovat suuressa osassa työnteossa ja vapaa-ajalla. Tämä tarkoittaa sitä, että ohjelmointi tulee kuulumaan ihmisten yleissivistykseen ja osaajia tarvitaan yhä enemmän. Tämä on huomioitu jo peruskouluopetuksessakin siten, että tänä vuonna ohjelmointi tulee peruskoulun opetussuunnitelmaan osaksi matematiikan tuntijakoa [40, s.9]. Tässä diplomityössä tulemme paneutumaan ohjelmoinnin opetukseen opetusteknisestä näkökulmasta, mutta sitä ennen määritellään mitä ohjelmointi on ja mitä haasteita sen opettamisessa usein ilmenee.

1.1 Ohjelmointi

Ohjelmoinnilla tarkoitetaan täsmällisten, tiettyä syntaksia eli kielioppia noudattavien, toimintaohjeiden syöttämistä tietokoneelle, jolla tässä tarkoitetaan mitä tahansa ohjelmoitavaa laitetta. Ohjelmoinnissa tuotettua ohjelmakoodia voi verrata vaikkapa ruoanlaitto-ohjeisiin, jossa kerrotaan vaihe vaiheelta, mitä kokin tulee tehdä ja mitä ruoka-aineita hänen tulee käyttää. Tietokone käy kokin tapaan näitä rivejä läpi ja suorittaa jokaisella rivillä olevan käskyn [40, s.17-18]. Näiden käskyjen suorittamisesta muodostuu lopulta vaikkapa tietokonepelin toimintalogiikka.

Ohjelmoinnin apuna käytetään yleensä integroitua ohjelmointiympäristöä (eng. Integrated Development Environment, IDE), jolla tarkoitetaan ohjelmointityökalua, joka sisältää ainakin tekstieditorin ja ohjelmakoodin suoritusympäristön. Tekstieditori voi sisältää erilaisia ohjelmointia helpottavia ominaisuuksia, kuten ohjelmakoodin automaattisen täydennyksen, tekstin värjäämisen käytettävän ohjelmointikielen syntaksin mukaan ja tyyli tarkastimen. Tarkastin varmistaa, että ohjelmoija noudattaa kielelle määritellyjä tyyliohjeita, eli tuottaa selkeää ohjelmakoodia, jota muut ohjelmoijat pystyvät vaivatta lukemaan.

Ohjelmoijat tekevät aina enemmän tai vähemmän virheitä, mikä voi johtaa väärään toiminnallisuuteen, tietoturvaongelmiin tai jopa ohjelman täydelliseen toimimattomuuteen. Tätä varten ohjelmia on testattava. Jos ohjelmassa huomataan jokin virhe, sen aiheuttaja pitää pystyä paikallistamaan. Tämä voidaan tehdä esimerkiksi käyttämällä nk. debuggeria, joka sisältyy yleensä nykyaikaisiin IDEihin. Debuggerilla voimme ajaa ohjelmaa osissa, laittaa se pysähtymään tietyille ohjelmakoodiriville

ja tarkastella tarkemmin ohjelman sisäistä tilaa.

1.2 Ohjelmoinnin opettaminen

Tullakseen hyväksi ohjelmoijaksi oppilaan täytyy hankkia vankka teoriapohja, joka koostuu spesifisistä yksityiskohdista. Tämän lisäksi hänen tulee harjoitella runsaasti ohjelmointia käytännössä, jotta hän oppii soveltamaan teoreettisia tietojaan käytännön ongelmien ratkaisuun [66]. Oppilaille tulee siis tarjota runsas ja monipuolinen oppimateriaali, josta he saavat tietopohjan opetettavasta aiheesta, sekä paljon harjoitustehtäviä, joilla harjoitella näiden tietojen soveltamista käytäntöön.

Opiskelijan tulee lisäksi saada palautetta harjoitustöistään mahdollisimman usein. Tällä pyritään estämään väärinkäsitykset ja huonojen ohjelmointitapojen, kuten ohjelmakoodin kommentoimatta jättämisen, juurtuminen opiskelijaan. [29] [19, s.93] Kuitenkin, koska harjoitustöitä ja yleensä opiskelijoita on paljon, tulee jatkuvan palautteen antaminen opettajalle ennen pitkää rankaksi.

Näiden asioiden takia ohjelmoinnin opetuksessa käytetään usein apuna tietoteknisiä apuvälineitä, jotka voidaan jakaa seuraaviin osa-alueisiin: visualisaatiotyökalut, automaattiset arviointityökalut, ohjelmointia tukevat työkalut ja mikromaaailmat [42, s.209-211]. Vaikka tutustumme näihin tarkemmin vasta myöhemmin, voimme silti jo päätellä tästäkin jaottelusta, että opetustyökalut ovat erilaisia, irrallisia ja niitä on olemassa paljon [26]. Näin ollen kaikenkattavaa, yhtenäistä oppimisympäristöä, joka sisältää kaikki tehokkaaseen opetukseen tarvittavat työkalut, on haastava rakentaa. Tämä on ollut ongelma myös Tampereen teknillisessä yliopistossa pidetyllä Johdatus ohjelmointiin -opintojaksolla.

1.3 Johdatus ohjelmointiin -opintojakso

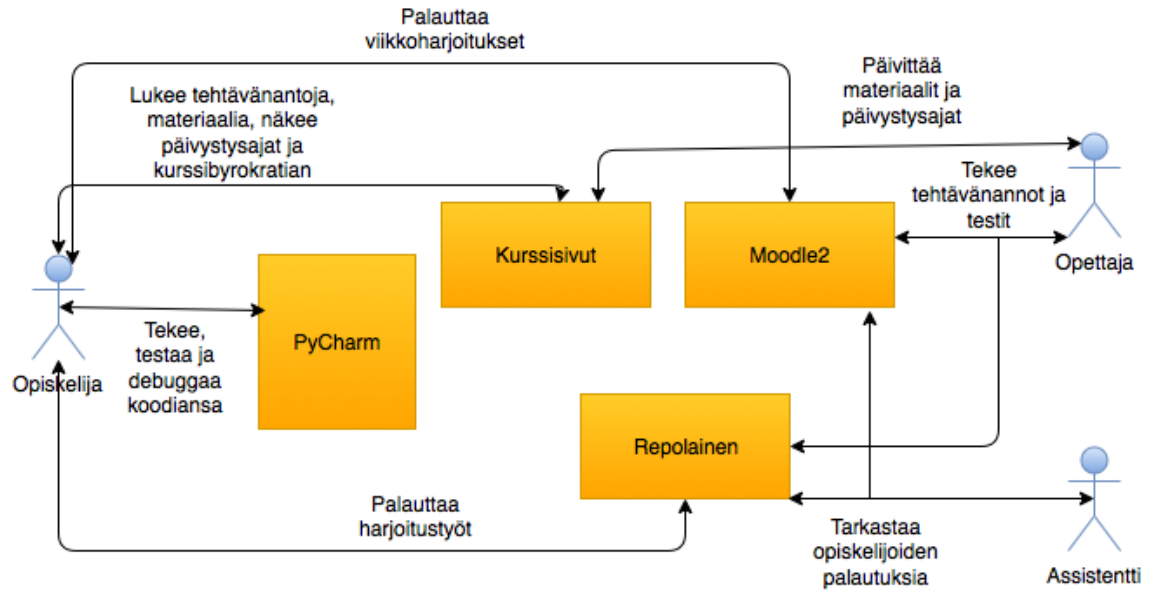
Johdatus ohjelmointiin -opintojakso järjestetään Tampereen teknillisessä yliopistossa sekä syksyisin että keväisin 2 periodin pituisina toteutuksina. Käytännössä kurssin pituus on noin 14 viikkoa ja arvioitu työmäärä yhtä opiskelijaa kohti on 143 tuntia. Osallistujamäärä vaihtelee 300-600 välillä, mikä osaltaan tekee opetuksesta haasteellista. Tämän vuoksi kurssilla käytetään runsaasti tietoteknisiä apuvälineitä opetuksen ja arvioinnin tukena.

Kurssi eroaa luentotyylisestä, jossa asiat käydään ensin luennolla läpi ja tämän jälkeen niitä harjoitellaan ohjatuissa viikkoharjoituksissa. Opiskelijat tutustuvat opintomonisteesta ja luentokalvoista koostuvaan kurssimateriaaliin itsenäisesti ja tekevät harjoitustehtävät, ennen kuin asia on käsitelty luennoilla. Seuraavalla viikolla luenolla syvennetään tietämystä niistä asioista, jotka opiskelija on itsenäisesti edellisviikolla opiskellut. Opiskelijat eivät kuitenkaan joudu opiskelemaan asioita ilman tukea, vaan he voivat tulla ”kooditorioksi” nimettyyn monitoimitilaan kysymään as-

sistenteilta neuvoa tehtävien tekemiseen. Harjoitukset eivät siis ole ohjattuja, vaan opiskelijat saavat tulla ja mennä, miten itse parhaaksi näkevät ja assistentit päivystävät ”kooditoriossa” tietyn määrän tunteja viikossa. Assistenttien lisäksi opiskelijoilla on itseopiskelun apuna erinäisiä työkaluja, joihin tutustutaan seuraavaksi.

Nykyinen oppimisympäristö on kuvattu kuvassa 1.1. Järjestelmä koostuu JetBrains-nimisen yrityksen tekemästä IDE:stä nimeltä PyCharm, kurssin internetsivuista, Moodlesta ja Repolaisesta. Moodle on nk. oppimisenhallintajärjestelmä, eli LMS (eng. Learning Management System), jonka avulla kurssin opiskelijoiden tietoja voidaan hallinnoida. Nämä tiedot koostuvat esimerkiksi opiskelijan nimestä, opiskelijanumerosta, tiedosta läpäistyistä tehtävistä ja kerätystä pistemäärästä. Viikkoharjoituspalautusten automaattiseen tarkastukseen kurssilla käytetään Moodleen saatavaa VPL-lisäosaa [70]. Repolainen taas on eräänlainen helppokäyttöinen graafinen käyttöliittymä Gitlab-repositorioon, mikä helpottaa Gitin käyttöön perehtymättömien työskentelyä. Git on versionhallintaohjelmisto, jolla voidaan kirjoitetusta ohjelmakoodista säilyttää eri versioita sen mukaan, mitä ohjelmakoodin kirjoitus etenee ja tarvittaessa palauttaa ohjelmakoodin vanhoja versioita. Paikkoja, joissa näitä versioita säilytetään, kutsutaan repositorioiksi. Gitlab on repositorio, joka sisältää muutamia lisätoimintoja, kuten wikin.

PyCharmia käytetään itse ohjelmakoodin kirjoittamiseen, ajamiseen ja debuggaamiseen. Se sisältää myös reaaliaikaisen tyylitarkastimen. Kurssisivuilla on luentomoniste, kurssikalenteri, jossa kerrotaan päivystysajat ”kooditoriossa”, tehtävänannot viikkoharjoitukseen sekä harjoitustöihin ja muu kurssiin liittyvä informaatio. Moodle on viikkoharjoitusten palauttamista ja tarkastamista varten, kun taas Repolaiseen palautetaan harjoitustyöt.



Kuva 1.1: Käyttäjäkunta koostuu opiskelijoista, opettajista ja assistenteista

1.4 Tutkimusongelma ja sen ratkaisu

Kurssilla käytetyssä järjestelmässä ongelmaksi on noussut opiskelijoiden työskentelyprosessin kärsiminen monen järjestelmän takia. Esimerkiksi, kun opiskelija on saanut tehtyä tehtävän valmiiksi ja testannut sen, hänen pitää kirjautua Moodleen tai Repolaiseen, etsiä palautusnäkyä ja palautettuaan odottaa arviointitulosta. Lisäksi jokaista tehtävää varten oppilaan pitää luoda IDE:ssään ohjelmakooditiedosto ja mahdollinen projektikansio. Tehtäviä kurssilla on yli 100, joten pidemmän päälle tästä tulee oppilaalle turha taakka joka vie huomiota itse ohjelmoinnista. Tämänkaltaisista ylimääräisistä työvaiheista halutaan siis päästä eroon.

Näistä ongelmista nousee tutkimuskysymys: ”**Miten pystymme rakentamaan yhtenäisen ohjelmoinnin oppimisympäristön? Mitä osia sen pitää sisältää ja miten näiden tulisi kommunikoida keskenään?**” Tässä diplomityössä etsimme vastauksen edellä mainittuihin kysymyksiin konstrukttiivisen tapaustutkimuksen avulla. Konstrukttiivisella tutkimuksella tarkoitetaan ongelmanratkaisua mallin, koneen, tietojärjestelmän tai vastaavan rakentamisen avulla. Konstrukttiivinen tutkimusote kytkeytyy aikaisempaan teoriaan, kirjallisuuteen ja tutkimukseen, joiden pohjalta pyritään luomaan käytännössä toimiva ratkaisu. Konstrukttiivinen tutkimus on usein tapaustutkimus, jossa tutkitaan yksittäistä kohdetta, jossa tehty ratkaisu voidaan lopulta yleistää [69]. Tässä työssä siis etsimme yleispätevät vastaukset aikaisemmin mainittuihin kysymyksiin rakentamalla yhtenäisen järjestelmän Johdatus ohjelmointiin -opintojaksolle.

Rajoitus tulevalle järjestelmälle on, että siinä ei käytetä Moodlea. Tämä rajoi-

tus nousee monista sen käytössä ilmenneistä ongelmista. Esimerkiksi, raporttien tuottaminen ei onnistu suuren osallistujamäärän takia, sekä tehtävien muokkausten saaminen voimaan voi viedä useita kymmeniä minutteja. Lisäksi järjestelmässä on käytettävyyso ongelmia tehtävien numeroinnin suhteen: jos tehtävälisan välistä poistetaan tehtävä, joudutaan numeroimaan sen jälkeen tulevat tehtävät uudestaan käsin.

Moodle korvataan oppimisenhallintajärjestelmällä nimeltä A+, jonka on kehittänyt Aalto-yliopiston Learning + Technology -tutkimusryhmä [26]. A+ tarjoaa ai-noastaan oppilaiden palautusten ja pisteiden tallettamisen, mutta siihen voi liittää muita palveluita sen tarjoaman rajapinnan avulla. Esimerkiksi tehtävien automaattinen tarkastaminen voidaan ja täytyy tehdä erillisessä palvelussa, joka kommunikoi A+:n kanssa.

1.5 Diplomityön rakenne

Vastausten etsiminen tutkimuskysymyksiin aloitetaan tutkimalla nykyään kurssilla käytettyjä ohjelmia. Luvussa 2 tutustumme Moodlen, Repolaisen, PyCharmin ja kurssisivujen tarjoamiin ominaisuuksiin. Samalla syvennämme ymmärrystä eri käyttäjäryhmien tarpeista toteutettavan järjestelmän näkökulmasta. Tällöin osaamme nimetä muita järjestelmäominaisuuksia, joita nykyinen järjestelmä ei tarjoa.

Kaikki halutut ominaisuudet eivät löydy A+:sta, vaan joudumme käyttämään useampia, mahdollisesti eri valmistajien, ohjelmia. Tällöin ongelmaksi nousee se, miten eri ohjelmat liitetään yhdeksi yhtenäiseksi oppimisympäristöksi. Luvussa 3 teemme kirjallisuuskatsauksen olemassaoleviin opetusjärjestelmiin ja kommunikointitekniikoihin sekä hahmottelemme näiden avulla järjestelmälle arkkitehtuurin. Käytämme tässä lisäksi apuna sovellettua 4+1-arkkitehtuurimallia.

Kun olemme hahmotelleet järjestelmällemme arkkitehtuurin, voimme tutustua valmiisiin komponentti-implemентаatioihin. Luvussa 4 teemme katsauksen valmiisiin komponentteihin, valitsemme niistä tarpeisimmat ja määrittelemme lopullisen järjestelmäarkkitehtuurin.

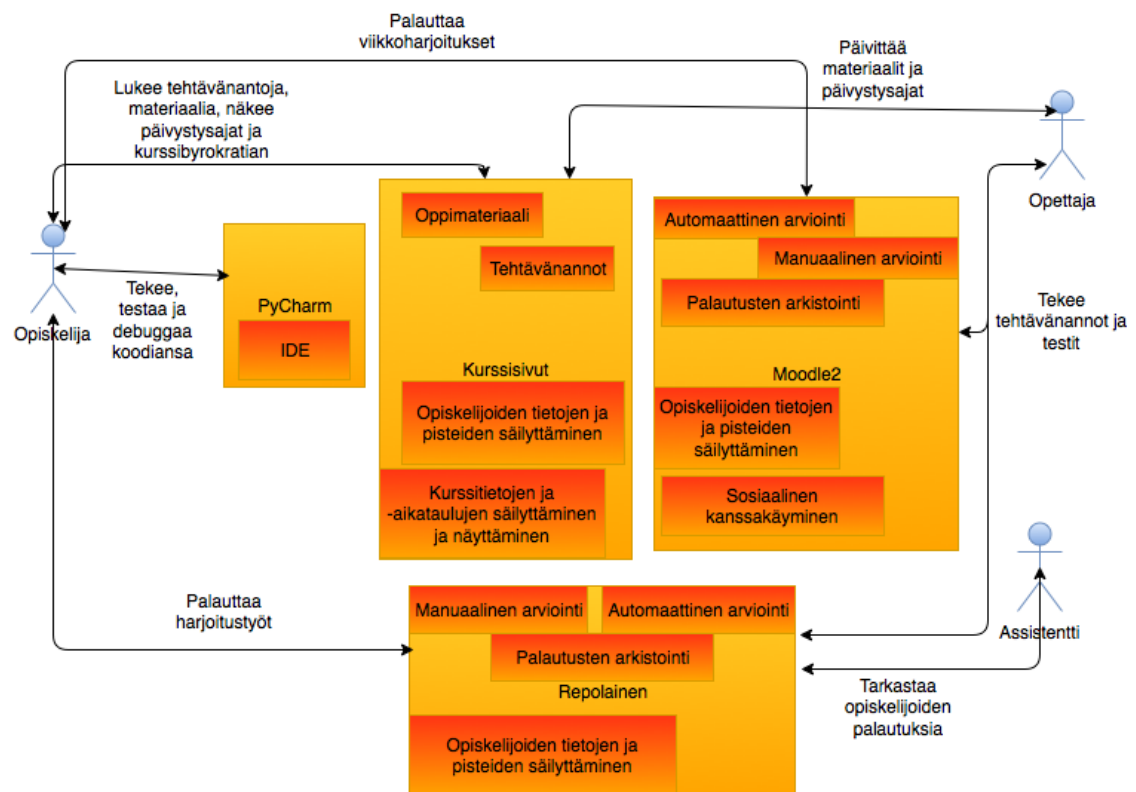
Kaikkia toiminnallisuuksia ei voida toteuttaa valmiilla komponenteilla. Luvussa 5 kerrotaan järjestelmän käytännön toteutuksesta niiden osien kannalta, jotka joudumme itse kehittämään.

Luvussa 6 arvioidaan, kuinka hyvin pystyimme vastaamaan tutkimuskysymyksiin. Tämä tehdään arvioimalla kehittämämme järjestelmän yhtenäisyyttä, osien valintaa ja niiden kommunikointia. Lopuksi luvussa 7 tehdään yhteenveto työstä ja pohditaan jatkokehitysmahdollisuuksia.

2. KÄYTTÄJIEN TARPEET JA JÄRJESTELMÄN VAATIMUSMÄÄRITTELY

Tässä luvussa kerrataan ja tarkennetaan johdantokappaleessa mainittuja vaatimuksia. Aloitamme tämän tutkimalla nykyisen järjestelmän osia, jotka esiteltiin kuvassa 1.1. Tutustumme toiminnallisuuksiin, joita osat käyttäjäryhmille tarjoavat, kerromme mihin tarpeeseen nämä toiminnallisuudet vastaavat ja esittelemme toiminnallisuuksille parannusehdotuksia. Tällöin saamme pohjan vastaukselle tutkimuskysymykseen ”mitä osia järjestelmän pitää sisältää”. Tätä pohjaa täydennämme luvun lopussa listaamalla lisäominaisuuksia, joille kurssin henkilökunta on havainnut olevan tarvetta.

Kuvassa 2.1 on eroteltu järjestelmän osien tarjoamat toiminnallisuudet. Seuraavissa osien mukaan jaetuissa alaluvuissa tutustutaan näihin toiminnallisuuksiin tarkemmin.



Kuva 2.1: Nykyisen järjestelmän tarjoamat toiminnallisuudet ohjelmittain

2.1 PyCharm

Kurssin oppimistavoitteisiin kuuluu, että oppilas oppii käyttämään jotain oikeaa ohjelmointiympäristöä, eli IDE:ä. Kurssilla ohjelmoidaan Python -ohjelmointikielellä, mikä pitää ottaa huomioon IDEä valitessa, koska kaikki eivät Pythonia tue ainakaan ilman lisäosia.

IDEen voidaan luoda projekteja ja näiden sisälle ohjelmakooditiedostoja. Kun oppilas tekee tehtäviä, hänen tulee tehdä jokaista ohjelmointitehtävää varten ohjelmakooditiedosto ja mahdollisesti uusi projekti. Tämä on triviaali toiminto, jota pitää toistaa monta kertaa kurssin aikana, joten jos tämä voitaisiin tehdä automaattisesti, opiskelijoiden oppimisprosessi virtaviivaistuisi. Eräs vaihtoehto on, että IDE hakee kurssipalvelimelta listan tehtävistä, jonka avulla se automaattisesti luo projektit ja tiedostot. Toinen oppimisprosessia nopeuttava tekijä olisi mahdollisuus palauttaa tehtävätiedosto suoraan IDE:stä kurssipalvelimelle arvioitavaksi.

Nykyisessä järjestelmässä käytetään IDEä nimeltä PyCharm, jonka opiskelija asentaa omalle tietokoneellensa. PyCharm tukee ns. plugineja, eli lisäosia. Plugin integroituu IDEen siten, että IDE:stä voidaan käyttää pluginin tarjoamia palveluita, kuten kommunikaatiokanavaa muihin komponentteihin.

2.2 Kurssisivut

Nykyisillä kurssisivuilla kerrotaan kurssin byrokraattiset asiat ja aikataulut. Siellä myös jaetaan kurssimateriaali, joka koostuu ”prujusta” eli luentomonisteesta ja luentokalvoista. Tehtävänannot ja tehtävien ohjelmakoodipohjat ovat myös kurssisivuilla, vaikka tehtävien palautus tehdään Moodleen ja Repolaiseen. Opiskelijat voivat myös tarkastella henkilökohtaisia tietojaan, kuten tenttituloksiaan, kurssisivuilta. Seuraavissa alakohdissa käydään läpi tarkemmin näitä kurssisivuilla käsiteltyjä asioita.

2.2.1 Kurssitiedot- ja aikataulut

Oppilaat käyvät luennoilla ja kooditoriossa, jossa assistentit auttavat heitä tehtävien tekemisessä. Järjestelmässä täytyy olla näkyvillä kooditorion päivystysajat, luentojen pitoajat sekä yleiset kurssiin liittyvät käytännön asiat. Harjoituksilla on määräajat, jotka tulee näkyä selkeästi järjestelmän oppilaiden näkymässä. Mahdolliset poikkeusmääräajat esimerkiksi oppilaan sairastumisen takia tulee sallia.

Näihin vaatimuksiin on nykyisessä järjestelmässä vastattu staattisilla, eli reaaliaikaisesti sisällöltään muuttumattomilla, internetsivuilla. Näissä kerrotaan muuttumattomat luentoaikataulut ja harjoitusten määräajat sekä kurssibyrokraattiset asiat. Kooditorioajat taas voivat päivittyä jopa päivittäin, joten näiden näyttämiseen on käytetty Google Kalenteri -komponenttia, joka on upotettu kurssisivuille.

2.2.2 Oppimateriaali

Harjoitusten teon apuna opiskelijoilla tulee olla monipuolista ja selkeää materiaalia, joka koostuu tekstistä, kuvista, interaktiivisista ohjelmavisualisoinneista ja videoista. Nykyinen järjestelmä vastaa tässä vain osaan vaatimuksista. Kurssisivuilla jaetaan kurssimateriaali, joka koostuu luentomonisteesta ja luentokalvoista, PDF-tiedostoina. PDF tulee sanoista Portable Document Format, eli kyseessä on tiedostomuoto joka sisältää muotoiltua tekstiä ja kuvia, sekä jopa videotiedostoja. Kurssin PDF-tiedostot tosin sisältävät vain tekstiä ja kuvia.

Erillisten PDF-tiedostojen lataus, avaaminen ja selaaminen tosin hidastavat opiskelijoiden työskentelyprosessia. Kurssihenkilökunta on päätenyt siihen tulokseen, että oppimateriaali tulisi sisällyttää samaan paikkaan limitettynä tehtävänantojen kanssa web-sivuille. Lisäksi kurssivastuuhenkilöiden mukaan helpointa olisi tehdä materiaalit oman tietokoneen tekstieditorilla ja tallettaa materiaali käyttäen Git-repositoriota, josta tiedostot voidaan hakea oppimisympäristön materiaalin web-näkymään. Tätä repositoriokomponenttia kutsumme tässä työssä materiaaliarkistoksi.

2.2.3 Tehtävänannot

Opettajat lisäävät nykyisessä järjestelmässä tehtävänannot kurssisivuille. Tehtävänannot ovat kurssiaikataulujen tapaan staattista sisältöä, joka sisältää tekstiä, kuvia ja linkkejä ohjelmakoodipohjatiedostoihin. Tehtävänannot tehdään samaan tyyliin, kuin muukin oppimateriaali, joten ne voidaan uudessa järjestelmässä tallettaa materiaaliarkistoon.

Assistenttien työtehtäviin kuuluu valmistautuminen kooditoriopäivystykseen, mikä käytännössä tarkoittaa tehtävänantojen läpikäymistä ja tehtävien koodaamista. Assistentit antavat kurssivastuuhenkilöille kommentteja ja korjausehdotuksia tehtävänannoista sähköpostin välityksellä. Kommentointi nopeutuisi, jos assistentit pääsisivät tekemään kommenttimerkintöjä suoraan tehtävänantoihin.

2.3 Repolainen

Kun opiskelija palauttaa ohjelmansa Repolaiseen tämän web-käyttöliittymän kautta, palatus tallennetaan Gitlab-repositorioon ja palautetulle tiedostolle ajetaan automaattiset testit. Kun testit on ajettu, opiskelija näkee niiden tulokset Repolaisesta. Assistentit voivat myös tarkastaa tehtäviä manuaalisesti, eli antaa kirjallista palautetta ja määrätä arvosanan. Seuraavissa alakohdissa tutustutaan tarkemmin Repolaisessa tehtävään automaattiseen ja manuaaliseen arviointiin.

2.3.1 Automaattinen arviointi

Harjoitustehtäviä ja opiskelijoita on paljon, joten kurssilla käytetään automaattista arviointia (eng. automatic assessment). Termi ”automaattinen arviointi” viittaa ohjelmiin, jotka tuottavat tietoa, jota voidaan käyttää arviointitulosten muodostamiseen [3, s.27]. Meidän järjestelmässämme automaattinen arviointi toimii siten, että opiskelija palauttaa tekemänsä työn järjestelmään, minkä jälkeen järjestelmä ajaa ohjelmakoodin käyttäen opettajan tekemiä testitapauksia ja vertaa malliratkaisun tulosteita opiskelijan ohjelman tulosteisiin.

Oppilaan palauttaman koodin ajo tulee tehdä turvatussa ympäristössä, jota kutsutaan hiekkalaatikoksi (eng. sandbox). Tätä turvatoimenpidettä tulee käyttää, koska oppilaat voivat palauttaa mitä tahansa kooditiedostoja, jotka pahimmassa tapauksessa ajettaessa voivat ottaa hallintaansa koko oppimisympäristön. Nykyisessä järjestelmässä ei ole tällaista turvajärjestelyä, vaan opiskelijoiden palautukset ajetaan kurssipalvelimella käyttäen samaa käyttäjätunnusta, jolla muokataan kurssisivuja. Vihamielisten käskyjen ajo estetään kieltämällä kirjastojen käyttö palautuskoodissa. Lisäksi koodin palautuksen yhteydessä ajetaan skripti, eli kooditiedosto, joka poistaa mahdollisesti vaarallisia elementtejä palautuskoodista ennen sen ajoa.

Automaattisen arvioinnin opiskelijalle antama palaute kurssillamme voi olla kahdenlaista: eroavaisuustuloste tai epäonnistuneet testitapaukset kertova tuloste. Eroavaisuustuloste kertoo, mitä ohjelman olisi pitänyt tulostaa, ja sen rinnalla näyttää opiskelijan koodin tuottamat tulosteet. Jälkimmäinen palaute toimii siten, että testitapaukset nimetään sen mukaan, mitä niissä testataan, esim. ”tässä testissä testataan suuren opiskelijamäärän lisäämistä rekisteriin” ja näytetään epäonnistuneiden testien kuvaukset opiskelijoille.

Repolaisessa käytetään kumpaakin menetelmää, mutta opiskelijoille näytetään oletuksena vain jälkimmäisen tyyppinen tuloste. Aiemmilla toteutuskerroilla on tosin huomattu, että suuri osa kooditoriossa käyvistä opiskelijoista ei tajua tällaisten palautteiden perusteella, mikä ongelma oikeasti on. Joskus voi käydä myös siten, että testitapauksessa on testattu jotain muutakin asiaa, kuin mitä opiskelijalle kerrotaan. Näissä tapauksissa on auttanut, että opiskelijalle näytetään myös eroavaisuustuloste.

2.3.2 Manuaalinen arviointi

Assistentit antavat oppilaille palautetta myös manuaalisesti. Käytännössä tämä tarkoittaa sitä, että assistentti lukee opiskelijan tekemän koodin läpi, analysoi koodin järjestyksen ja tyylikkyyden, sekä mahdollisesti testaa koodia. Tästä on erityistapauksena viimeinen harjoitustyö, jossa harjoitellaan graafisen käyttöliittymän tekoa. Tätä ohjelmaa ei testata automaattisesti, joten assistentin on pakosti testattava kyseinen ohjelma. Suuren oppilasmäärän takia tarvitaan monta assistenttia tarkasta-

maan harjoitustöitä, joten palautteen antamisen nopeuttaminen on tarpeen.

Nykyisessä järjestelmässä hidastavaksi tekijäksi tulee se, että koodi ja mahdolliset muut tiedostot, kuten kuvat, pitää kopioida ensin omalle tietokoneelle, jotta ohjelma voidaan ajaa. Prosessi nopeutuisi huomattavasti, jos ohjelmaa pystyisi ajamaan suoraan selaimella. Myös palautetekstin ja arvosanan antaminen on suhteellisen hidasta. Kurssivastuuhenkilö tekee assistenteille arvostelupohjan, jossa on lueteltu yleisempiä virheitä ja onnistumisia. Assistentti kopioi tämän pohjan Repolaisessa sijaitsevaan kansioon, jonne arvosteltava palautus on luotu. Kopioinnin jälkeen teksti muokataan sopivaksi poistamalla ylimääräiset rivit ja lisäämällä omia kommentteja. Arvosana annetaan tekemällä samaan kansioon arvosana -tekstitiedosto, johon kirjoitetaan arvosana. Eräs tapa nopeuttaa palautteen tekoa olisi nk. rubriikkien käyttö, joista kerrotaan lisää kappaleessa 4.3.

2.4 Moodle

Kurssilla käytetään oppimisenhallintajärjestelmää nimeltä Moodle, minkä lisäksi viikkotehtävien automaattiseen tarkistamiseen käytetään VPL (Virtual Programming Lab) -lisäosaa [70]. VPL antaa lähes samat mahdollisuudet tehtävien automaattiseen ja manuaaliseen tarkastamiseen kuin Repolainen, mutta se antaa oppilaalle automaattisen palautteen ainoastaan eroavaisuustulosteena, josta kerrottiin luvussa 2.3.1. Moodle sisältää tehtävien palautus- ja arviointiominaisuuksien lisäksi keskustelufoorumin, jonka kautta oppilaat voivat olla sosiaalisessa kanssakäymisessä. Seuraavissa alakohdissa käsitellään tarkemmin oppimisenhallintajärjestelmiä ja sosiaalista kanssakäymistä.

2.4.1 Oppimisenhallintajärjestelmä

Oppilaiden tiedot ja pistelistaukset tulisi tallentaa yhteen paikkaan, josta kurssivastuuhenkilöt pääsevät näitä helposti tarkastelemaan. Tähän tarkoitukseen voimme käyttää oppimisenhallintajärjestelmää, joka on integroitu joukko ohjelmia, jotka auttavat kurssinhallinnassa, oppilaiden seurannassa ja kurssiraporttien teossa. Se tarjoaa keskitetyn ratkaisun kurssin aikataulutukseen, oppilaiden rekisteröintiin ja oppimistulosten arviointiin [41, s.645].

Nykyään oppimisenhallintajärjestelmää käytetään tällä opintojaksolla ainoastaan viikkoharjoitusten arvosteluun ja näiden arvostelujen tallentamiseen. Sitä voitaisiin käyttää keskitettynä komponenttina, jonka kautta oppilas pääsee käsiksi kaikkiin muihin palveluihin, esimerkiksi automaattiseen arviointiin.

2.4.2 Sosiaalinen kanssakäyminen

Oppilaiden tulisi jotenkin pystyä keskustelemaan tehtävistä ja muusta kurssiin liittyvästä helposti. Lisäksi keskustelujen tulisi jäädä kaikkien näkyviin. Oppilaiden olisi hyvä myös nähdä muiden edistyminen oman edistymisen lisäksi, jolloin oppilas voi arvioida, kuinka hyvin hän on aikataulussa tehtävien teossa.

Moodle sisältää keskustelufoorumin, jonne voidaan tehdä aihealueita, joihin voidaan perustaa viestiketjuja. Viestit jäävät näkymään määritellyksi ajaksi, joten kahden ensimmäiseen vaatimukseen vastataan. Viimeiseen vaatimukseen ei ole nykyisessä järjestelmässä vastattu, mutta siihen palataan myöhemmin luvussa 2.5.3.

2.5 Uudet ominaisuudet

Kurssihenkilökunta on kuluneiden toteutuskertojen ajan tarkkaillut opiskelijoiden oppimista kurssipalautteiden, kooditoriossa ilmenneiden ongelmien ja harjoitustöiden avulla. Seuraavissa alaluvuissa tutustutaan tästä tarkkailusta ja muusta pohtimisesta nousseisiin ominaisuuksiin, joista oppilaiden odotetaan hyötyvän.

2.5.1 Ohjelmavisualisointi

Oppilaiden voi olla aluksi vaikea hahmottaa, mitä ohjelmat oikeasti tekevät ja miten tietorakenteet toimivat. Tämän hahmottamisprosessin helpottamiseksi voidaan käyttää apuvälineenä interaktiivista ohjelmavisualisointia. Stasko et al:n määrittelyn [55, s.4] mukaan ohjelmavisualisointi (eng. software visualization) on typografian, graafisen konstruktion, animaation ja elokuvauksen soveltamista käyttäen ihmisen ja tietokoneen välistä vuorovaikutusta ja tietokonegrafiikkaa ihmisen ymmärryksen lisäämiseksi ja tietokoneohjelman tehokkaan käytön helpottamiseksi.

Tässä työssä interaktiivisella ohjelmavisualisoinnilla tarkoitetaan ohjelmakoodissa ajettavien käskyjen ja muuttujien sisältöjen esittämistä graafisesti siten, että opiskelija voi vaikuttaa esityksen kulkuun. Tämä voidaan tehdä perinteisesti piirtämällä taululle, mutta tämä on mahdollista vain, kun assistentti tai opettaja on oppilaan kanssa. Itseopiskelua varten on olemassa ohjelmia, jotka esittävät oppilaalle graafisesti ohjelman suorituksen kulun. Kutsutaan tällaista komponenttia tässä työssä visualisointikomponentiksi. Kyseinen komponentti on läheisissä tekemisissä materiaaliarkiston kanssa, koska ohjelmavisualisoinnit ovat osa kurssimateriaalia. Tarkempi integraatio riippuu valittavan visualisointikomponentin toimintaperiaatteesta.

2.5.2 Vertaisarviointi

Ohjelmoijan tulee osata myös lukea muiden tekemään ohjelmakoodia, joten tätäkin tullaan kurssilla harjoittelemaan ohjelmakoodipohjien ja vertaisarvioinnin avulla. Ohjelmakoodipohjilla tarkoitetaan opiskelijoille valmiina annettavia ohjelmakoodinpätkiä, jotka opiskelija täydentää valmiiksi ohjelmaksi. Vertaisarvioinnissa taas opiskelijat saavat luettavaksi ja analysoitavaksi muiden opiskelijoiden palautuksia, joista he palauttavat raportin.

Vertaisarviointikomponentin tulee jakaa ohjelmakoodiarvioinnit siten, että jokainen oppilas saa jonkun toisen oppilaan ohjelmakoodin arvioitavaksi. Komponentin tulee myös tarjota oppilaille rajapinta tai käyttöliittymä, jossa arviointi voidaan tehdä. Komponentin tulee myös pystyä lähettämään arvioinnit LMS:ään, josta ne siirtyvät assistenttien tarkastettavaksi manuaalisen arvioinnin komponentille.

2.5.3 Etenemisen hahmottaminen ja pelillistäminen

Opiskelijan oppimisprosessin etenemisen hahmottaminen voidaan toteuttaa monella tavalla. Yksi tapa on pelillistäminen, jossa opiskelijat saavat esim. mitaleja tai arvonimiä saavutettuaan tiettyjä virstanpylväitä. [18] Tähän voi liittää myös saavutusten näkymisen muille opiskelijoille, jolloin opiskelijat saavat kuvan siitä, missä vaiheessa muut opiskelijat ovat. Tämä toimii myös ilman saavutuksia näyttämällä esim. mitkä tehtävät opiskelija on tehnyt vaikkapa taulukkomuodossa. Opiskelijoiden on myös saatava tietty määrä tehtäviä tehtyä, että he pääsevät kurssista läpi ja saavat bonuspisteitä. Tämän tilanteen voisi järjestelmä näyttää esimerkiksi prosentuaalisesti opiskelijalle.

Koska pelillistäminen on vahvasti yhteydessä oppilaiden ansaitsemiin pisteisiin, olisi pelillistäminen järkevää integroida LMS:ään, jonka aiemmin kerrottiin sisältävän opiskelijoiden pistetiedot. Pelillistämisoimaisuus on kuitenkin tärkeydeltään niin vähäinen, että se sivuutetaan ja jätetään jatkokehityksaiheeksi.

2.5.4 Tentti

Kurssin lopuksi opiskelijan täytyy tehdä sähköinen tentti yliopiston EXAM-nimisessä järjestelmässä. Tentti koostuu samantyyppisistä ongelmista, joita kurssilla on viikkoharjoituksissa ja harjoitustöissä ratkottu. Kyseessä on web-pohjainen palvelu, joten tenttitulokset lienee mahdollista siirtää automaattisesti A+:n tietokantaan, jolloin opiskelijan kaikki tiedot ovat samassa paikassa. A+:aa voitaneen muokata myös siten, että EXAM-järjestelmään voidaan tentin aikana hakea A+:sta tehtävä sivu, joka on normaalisti piilotettu.

2.5.5 Oppimisen seuranta

Oppilaiden oppimistapoja halutaan seurata tutkimus- ja opetuksen parantamismielessä. Erityisesti halutaan seurata oppilaiden koodaustaitojen kehittymistä ottamalla kaappauksia heidän koodeistaan tietyllä aikavälillä. Lisäksi materiaalin käytöstä halutaan saada dataa, jolloin voidaan päätellä, onko esim. jokin opetusvideo tehty hyvin. Täten seurantakomponentin pitäisi kommunikoida ainakin IDE:n ja materiaaliarkiston kanssa.

Opettajien tulee myös saada opiskelijoiden pistetilastot LMS:stä mahdollisimman helposti esimerkiksi csv-tiedostona tai JSONina. JSON, joka tulee sanoista JavaScript Object Notation, on tiedostomuoto palvelinten väliseen tiedonvälitykseen.

2.6 Yhteenveto yhdistettävistä komponenteista

Tässä luvussa jaoin nykyisen järjestelmän toiminnallisuuksien mukaan osiin, minkä lisäksi esittelimme uusia haluttuja toiminnallisuuksia. Näin saatujen tietojen perusteella päätämme, että järjestelmä tulee koostumaan seuraavista komponenteista: IDE, LMS, materiaaliarkisto, ohjelmavisualisointi, palautusarkisto, automaattinen arviointi, manuaalinen arviointi, vertaisarviointi, tentti ja oppimisen seuranta. Seuraavissa alakohdissa hahmottelemme alustavasti komponenttien yhteydet käyttötapausten kautta.

2.6.1 Komponenttien käyttötapaukset

Tutkitaan miten eri käyttäjäryhmät käyttävät eri komponentteja hyväksi ja miten nämä tarvitsevat toistensa palveluita. Rajataan käyttötapausluettelo niihin tapauksiin, joissa komponentit tarvitsevat toistensa palveluita tai muuten käyttävät toisinaan.

- (a) Oppilas hakee IDEensä tehtävänannot LMS:stä. Tällöin LMS pyytää arkistolta kyseisiä tiedostoja, jonka jälkeen arkisto lähettää ne LMS:lle ja tämä edelleen IDE:lle.
- (b) Oppilas palauttaa tehtävän käyttäen IDEä. IDE lähettää tehtävän, sekä oppilaan tiedot LMS:sään. LMS lähettää tehtävän arvioitavaksi automaattiseen arviointikomponenttiin ja tallentaa palautuksen palautusarkistoon. Testauksen tulos ja arvosana lähetetään arviointikomponentilta LMS:sään, joka lähettää arvosanan edelleen IDE:lle, joka tehtävän lähetti.
- (c) Oppilas lukee materiaalia LMS:stä. LMS hakee sivut materiaaliarkistosta ja näyttää ne oppilaalle. Materiaalin seassa voi olla visualisointeja, jotka haetaan ohjelmavisualisointikomponentilta.

- (d) Oppilas tekee vertaisarviointitehtävän. Oppilas kirjautuu LMS:ssään, josta hänellä on näkymä vertaisarviointikomponenttiin. Tämä komponentti poimii jonkin oppilaan ohjelmakoodin ja antaa tämän arvioitavaksi oppilaalle. Kun arviointi on tehty, se lähtee vertaisarviointikomponentille LMS:stä.
- (e) Oppilas tekee tentin tenttijärjestelmässä, josta arvosana lähtee LMS:sään, kunhan kurssivastuuhenkilö on tentin arvioinut.
- (f) Assistentti arvioi manuaalisesti tietyt työt. Assistentti kirjautuu manuaaliseen arviointikomponenttiin. Tämä hakee LMS:stä palautetun tehtävän sekä oppilaan tiedot. Kun arviointi on valmis, manuaalinen arviointikomponentti lähettää arvosanan ja palautteen LMS:sään.
- (g) Assistentti arvioi vertaisarvioinnin. Assistentti kirjautuu vertaisarviointikomponenttiin, joka listaa oppilaiden vertaisarvioinnit, sekä näiden tiedot. Kun arviointi on tehty, lähetetään arvosanat sekä palautteet LMS:sään.
- (h) Kurssivastuuhenkilö asettaa seurantakomponentissa lokitettavat tapahtumat. Komponentti tarkkailee IDEä ja LMS:ssää keräten näistä käyttödataa. Tämän datan prosessointi ja visualisointi tehdään seurantakomponentissa.

2.6.2 Komponenttien yhdistäminen

Edellä mainitut käyttötapaukset, komponentit sekä niiden ja käyttäjien suhteet ovat näkyvillä kuvassa 2.2. Kuvasta nähdään, että LMS (A+) toimii eräänlaisena keskuskomponenttina, johon muut komponentit ottavat yhteyden. Kysymykseksi kuitenkin jää, miten komponentit käytännössä yhdistetään toisiinsa. Tähän paneudutaan seuraavassa luvussa.

3. ARKKITEHTUURIRATKAISUT

Edellisessä luvussa luonnostelimme eri järjestelmäosien yhteyksiä toisiinsa ja käyttäjiin, ja esitimme tuloksen kuvassa 2.2. Suunnitelmissa jäätiin kuitenkin suhteellisen abstraktille tasolle, eikä eri komponenttien väliseen kommunikaatioon otettu täsmällisesti kantaa. Tätä varten täytyy järjestelmälle määritellä arkkitehtuuri. Mikkonen et al. [27, s.18] suomentavat IEEE:n arkkitehtuurien kuvaamista koskevan standardin määritelmän ohjelmistoarkkitehtuurista [22] seuraavasti: ”järjestelmän perusorganisaatio, joka sisältää järjestelmän osat, niiden keskinäiset suhteet ja niiden suhteet ympäristöön sekä periaatteet, jotka ohjaavat järjestelmän suunnittelua ja evoluutiota”.

Arkkitehtuurin suunnittelussa lähdemme liikkeelle sillä olettamuksella, että se sisältää LMS:n, johon muut komponentit ovat jollain tavalla yhteydessä. LMS voi olla monoliittinen, modulaarinen kuten esimerkiksi Moodle on, tai A+:n tapaan palvelupohjainen. Monoliittisuudella tarkoitetaan LMS:n yhteydessä sitä, että se itsessään sisältää kaikki ominaisuudet, eikä laajennus ole käytännössä mahdollista muuttamatta itse LMS:sää [9]. Modulaarinen LMS taas sallii lisäosat eli pluginit, mutta näiden ongelmana on, että niiden teossa täytyy käyttää samoja tekniikoita, kuin kyseisissä LMS:sä eikä lisäosan siirto LMS:stä toiseen yleensä onnistu [26]. Palvelupohjaisuuteen perustuvaan LMS:sään lisäkomponentit voidaan yhdistää käyttämällä löyhän sidoksen (loose coupling) periaatetta. Tällä tarkoitetaan sitä, että jokainen komponentti tarjoaa rajapinnan, jonka kautta muut voivat tehdä sille pyyntöjä ja mahdollisesti tehdä muutoksia sen sisäiseen tilaan koskematta sen sisäiseen toteutukseen. Palvelupohjaisten web-komponenttien tapauksessa viestit lähetetään internetprotokollia käyttäen.

Etsimme ratkaisun komponenttien yhdistämiseen käyttäen työkaluna 4+1 -arkkitehtuurimallia [28], joka koostuu neljästä eri näkökulmasta: looginen, kehitys, prosessi ja fyysinen. Lisäksi arkkitehtuuri sisältää skenaariot eli käyttötapaukset, jotka on jo esitelty luvussa 2.6.1. Valitsimme työkaluksi nimenomaan 4+1 arkkitehtuurimallin sen takia, että saamme kokonaiskuvan järjestelmästä aina käyttötapauksista fyysisten palvelimien sijoittamiseen asti.

Määrittelemme tulevissa aliluvuissa eri näkömien tarkoitukset sekä tutustumme kirjallisuudesta löytyvien esimerkkien avulla jokaiseen näkökulmaan poislukien kehitysnäkökulman ja skenaariot.

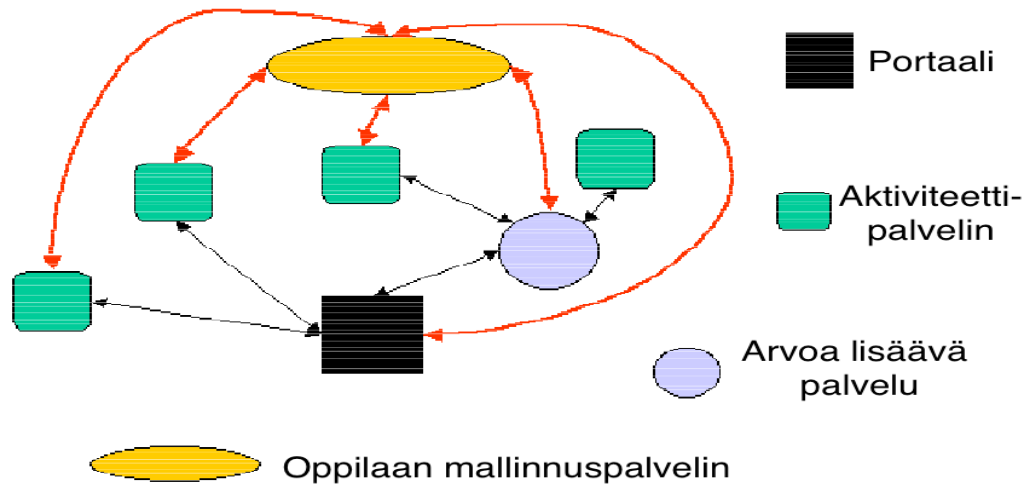
3.1 Looginen näkökulma

Looginen näkökulma kertoo, mitä palveluita järjestelmä tarjoaa käyttäjille ja miten nämä palvelut jakautuvat osiin [28]. Johdimme eräänlaisen loogisen näkökulman käyttötapauksistamme kuvaan 2.2. Vaihtoehtoisesti Brusilovsky esittää artikkelissaan KnowledgeTree: A Distributed Architecture for Adaptive E-Learning [8] omaa näkökulmaamme korkeammalla abstraktiotasolla olevan arkkitehtuurimallin, joka soveltuu erillisten opetuskomponenttien integroimiseen toisiinsa. Tutustutaan seuraavaksi KnowledgeTreen tapaan jakaa toiminnallisuudet osiin, jotta kuvassa 2.2 esitellyt komponentit saadaan jaoteltua paremmin.

3.1.1 KnowledgeTree

KnowledgeTree-arkkitehtuuri koostuu neljästä erilaisesta komponenttityypistä: oppimisportaali, aktiviteettipalvelut, arvoa lisäävät palvelut ja oppilaan mallintamispalvelu (kuva 3.1). Oppimisportaali on kuin LMS, se tarjoaa oppilaille keskitetyn sisäänkirjautumispisteen opetusympäristöön ja kaikki kursseilla tehtävät aktiviteetit, kuten tehtävänantojen luku ja tehtävien palautus. Lisäksi se antaa kurssivastuuhenkilön asettaa oppilaiden saataville kursseilla vaaditut resurssit. Portaali eroaa kuitenkin LMS:stä siten, että kurssimateriaali ja aktiviteetit eivät ole sen sisäisiä ominaisuuksia, vaan erillisiä palveluita, joita nimitetään aktiviteettipalvelimiksi [8].

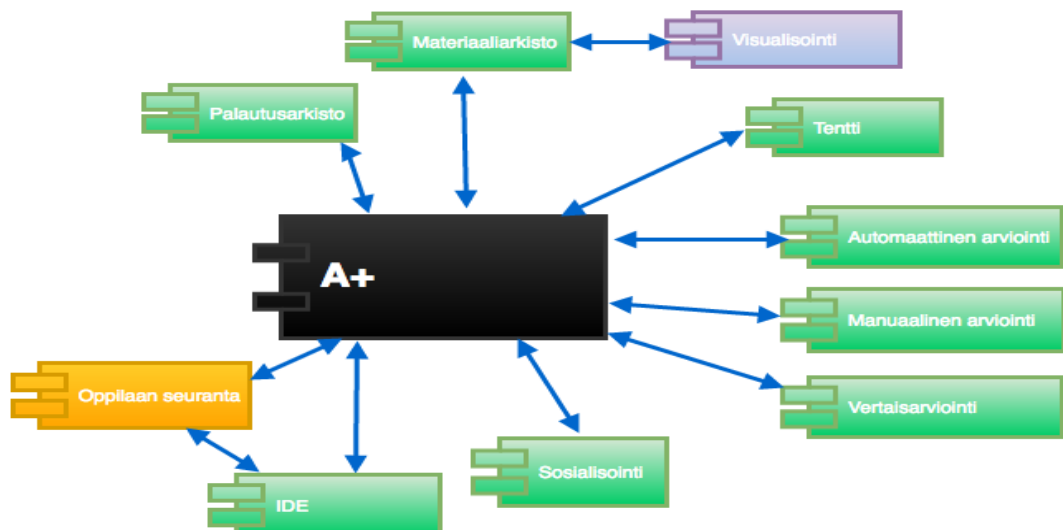
Aktiviteettipalvelimet voivat sisältää mitä vain yksinkertaisesta staattisesta sisällöstä aina keskustelufoorumiin ja vertaisarviointipalveluun asti. Nämä palvelut ja resurssit kurssivastuuhenkilö voi linkittää oppimisportaaliin, siten että ne näkyvät oppilaille osana portaalia. Arvoa lisäävät palvelut ovat eräänlaisia tiedon muokkaajia, joiden läpi sitä voidaan viedä. Arvonlisääjä voi olla esimerkiksi ohjelmavisualisointikomponentti, joka ohjelmakoodinpätkän saadessaan muodostaa siitä interaktiivisen visualisaation, mikä taas helpottaa ohjelmakoodin ymmärtämistä. Oppilaan mallintamispalvelulla tarkoitetaan eräänlaista tietovarastoa oppilaasta ja tämän käyttäytymisestä, jota voidaan käyttää hyödyksi tutkimuksessa ja oppimisympäristön personalisoinnissa oppilaalle. Mallintamispalvelu tarkkailee muita komponentteja ja sitä, miten oppilaat niitä hyödyntävät tallentaen nämä tiedot tietokantaansa.



Kuva 3.1: KnowledgeTree -arkkitehtuuri [8]

3.1.2 Järjestelmä loogisesta näkökulmasta

Palataan käsittelemään rakennettavaa oppimisympäristöä. Jakaaksemme kuvassa 2.2 esitetyt komponentit loogisiin kokonaisuuksiin, käytämme KnowledgeTree-arkkitehtuurissa määriteltyä komponenttijakoa. LMS vastaa portaalia, oppilaan seuranta-komponentti oppilaan mallinnuspalvelinta ja visualisointi on arvoa lisäävä palvelu, joka muokkaa dataa eli tässä tapauksessa ohjelmakooditiedostoja, ymmärrettävämpään muotoon. Loput komponenteista ovat erilaisia aktiviteetteja varten, eli ne ovat aktiviteettipalvelimia. Jaottelun lopputulos näkyy kuvassa 3.2.



Kuva 3.2: Arkkitehtuuri loogisesta näkökulmasta

3.2 Prosessinäkökulma

Prosessinäkökulmaa käytetään yleensä ei-toiminnallisten asioiden, kuten suorituskyvyn ja saatavuuden mittaamiseen. Prosessi on tietokoneella suoritettavien tehtävien joukko, joka muodostaa ajettavan yksikön. Yksikkö voidaan määritellä monella abstraktiotasolla, säietasosta aina yksittäisiin palvelimiin asti [28]. Meidän tapauksemme halutaan prosessinäkökulman kautta tutkia erityisesti komponenttien välistä viestintää, ei niinkään suorituskykyä tai muuta ei-toiminnallista vaatimusta.

Seuraavissa aliluvuissa tutustumme erilaisiin tapoihin, joilla komponentit voivat viestiä keskenään.

3.2.1 REST

Nykypäivänä palvelupohjaiset järjestelmät toteutetaan useimmiten käyttäen REST-arkkitehtuurityyliä. REST (Representational State Transfer) on Fieldingin väitöskirjassaan [12] esittelemä arkkitehtuurityyli, joka perustuu hallittuihin rajoitteisiin, joista työmme ymmärtämisen kannalta tärkeitä ovat vain vaatimukset asiakas-palvelin (client-server) -arkkitehtuurin käytöstä ja yhdenmukaisesta rajapinnasta (uniform interface) [12, s. 78-85].

Asiakas-palvelin -arkkitehtuuri muodostuu tietokoneista, joiden rooli voi olla joko asiakas tai palvelin. Palvelin odottaa passiivisena, kunnes yksi tai useampi asiakas ottaa siihen yhteyttä pyytäen jotain palvelua. Erityisesti REST:in tapauksessa palvelimen tehtävä on datan varastointi ja sen manipuloinnin tekninen toteutus, kun taas asiakaspuolelle jää käyttöliittymän toteutus. Asiakas-palvelin -arkkitehtuurin pyrkimyksenä on mahdollistaa kummankin osapuolen kehittyminen itsenäisenä ilman riippuvuutta toisesta komponentista, kunhan näiden välinen rajapinta ei muutu [12, s.45].

Rajoite yhdenmukaisesta rajapinnasta REST:issä jaetaan neljään eri peruseräiteeseen: resurssien tunnistaminen, resurssien muokkaaminen tunnisteiden perusteella, läpinäkyvä viestintä ja hypermedian ohjaama tilallisuus [12, s. 82]. Resurssilla tarkoitetaan mitä tahansa tietoa, jonka voi nimetä. Resurssi voi olla yksittäinen tiedosto, kuten kuva tai se voi viitata joukkoon muita resursseja [12, s.88]. Resurssien tunnistamisella tarkoitetaan sitä, että resurssit on saatavilla palvelimelta URI:n, eli yksikäsitteisen osoitteen perusteella. Resursseja ei anneta asiakkaalle sellaisenaan, vaan esimerkiksi JSON-formaatissa. Resurssien muokkaaminen tunnisteiden perusteella antaa asiakkaalle mahdollisuuden muokata ja poistaa resursseja palvelimelta rajapinnan kautta. Läpinäkyvä viestintä taas tarkoittaa sitä, että viestit sisältävät tiedon siitä, miten ne tulee prosessoida. Viesti esimerkiksi sisältää tiedon, missä formaatissa sen sisältö on. Hypermedian ohjaama tilallisuus (Hypermedia as the Engine of Application State, HATEOAS) tarkoittaa sitä, että viestit sisältävät vastaanotta-

jalle ohjeet mitä hän voi seuraavaksi tehdä. Se voi sisältää esimerkiksi hyperlinkkejä erilaisiin asiaan liittyviin resursseihin [72].

REST hyväksikäyttää yhtenäisen rajapinnan rakentamisessa HTTP-protokollan metodeja, tärkeimpinä POST, GET, PUT, PATCH, ja DELETE. Nämä metodit jaetaan turvallisiin (safe), idempotentteihin (idempotent) ja muihin metodeihin. Turvallinen metodi ei vaikuta kohdistettuun resurssiin mitenkään ja idempotentin metodin vaikutus resurssiin on sama, kutsuttiinpa sitä samoilla parametreilla monta kertaa peräkkäin tai vain kerran [62]. Metodien tarkoitukset ja luokitukset on lueteltu taulukossa 3.1.

Metodi	Käyttötarkoitus	Luokitus
POST	Uuden resurssin luominen	Muu
GET	Resurssin hakeminen	Turvallinen
PUT	Uuden resurssin luominen tai muokkaaminen	Idempotentti
PATCH	Resurssin muokkaaminen	Muu
DELETE	Resurssin poistaminen	Idempotentti

Taulukko 3.1: Käytetyimmät HTTP-metodit RESTissä [62]

POST ja PATCH on luokiteltu kategoriaan muu, eli kummatkaan eivät ole turvallisia tai idempotentteja. POSTin tapauksessa tämä tarkoittaa sitä, että jos sama POST-pyyntö tehdään monta kertaa peräkkäin, jokainen todennäköisesti luo kohdejärjestelmään uuden resurssin, eikä vain yhtä. PATCH taas on resurssin muokausmetodi, joka ei vaadi kokonaan uutta resurssia, vaan vain muutettavat kohdat. Tämä voi johtaa siihen, että jos sama PATCH ajetaan monta kertaa peräkkäin, resurssi muuttuu joka kerta erilaiseksi.

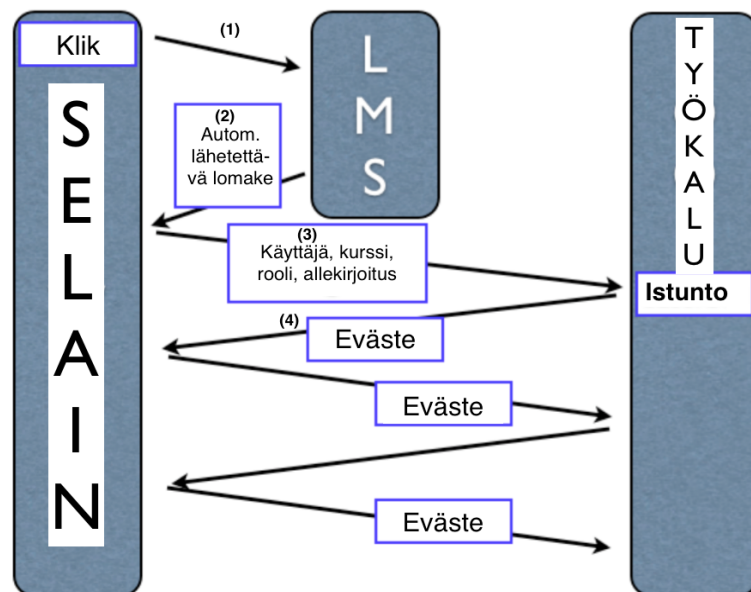
Esimerkki REST-paradigmaa käyttävästä implementaatiosta on IMS Global Learning Consortium -organisaation luoma opetusjärjestelmien väliseen kommunikointiin tarkoitettu standardi nimeltä Learning Tools Interoperability (LTI), jota käsitellään seuraavassa alaluvussa.

3.2.2 LTI-protokolla

LTI:n avulla LMS:n käyttäjä voi siirtyä ulkoiseen oppimisyökaluun, ilman että tämän tietokantaan joutuu luomaan toista käyttäjätunnusta. Täten LTI toimii eräänlaisena SSO:n (Single Sign-On), eli kertakirjautumisen tarjoajana, mutta lisäksi sen avulla voidaan lähettää ja vastaanottaa tietoa, kuten opiskelijan arvosanat, järjestelmästä toiseen. LTI-protokollan käytöstä hyötyvät oppilaiden lisäksi sekä oppimisympäristön ylläpitäjät, että opettajat. Ensinnäkin järjestelmään on helppo lisätä uusia palveluita, jolloin opettajatkin pystyvät niitä lisäämään. Lisäksi järjestelmää

on helpompi hallinnoida tietoturvan kannalta, koska sen voi jakaa yksittäisiin, toisistaan erotettuihin, palveluihin [23, s.3].

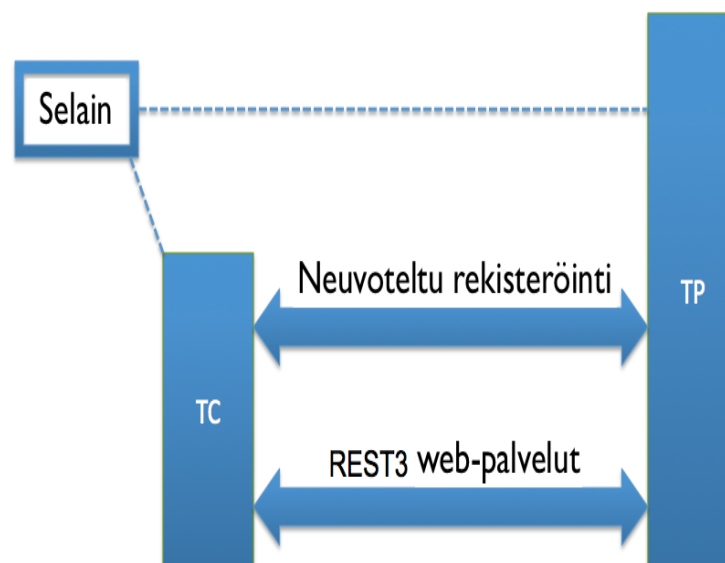
LTI-protokollassa sitä käyttävät järjestelmät jaetaan työkalujen käyttäjiin (Tool Consumer, TC) ja työkalujen tarjoajiin (Tool Provider, TP). TC:t käyttävät TP:iden palveluita, eli esim. LMS on TC ja vaikkapa ohjelmakoodin automaattinen tarkistin on TP [23]. TP:n kutsuprosessi TC:stä on esitetty kuvassa 3.3, jossa TP on kuvattu oikeassa reunassa olevalla työkalu-laatikolla ja TC vasemmassa reunassa olevalla LMS-laatikolla. Kun käyttäjä valitsee LMS:ssä toiminnon (1), joka on toteutettu ulkoisella komponentilla, LMS rakentaa käyttäjälle näkymättömän lomakkeen, johon se asettaa käyttäjän ja kurssin tiedot sekä kyseisen käyttäjän roolin kurssilla. Lomake allekirjoitetaan käyttäen OAuthia, joka on käyttäjien tunnistamiseen käytetty standardi. Allekirjoitus sisällytetään lomakkeeseen ja lomake lähetetään takaisin käyttäjän selaimen (2). Jos selaimessa on JavaScriptin käyttö sallittu, se lähettää lomakkeen automaattisesti käytettävälle komponentille (3). Lähetettyjen tietojen perusteella komponentti tunnistaa OAuthin avulla käyttäjän ja avaa tälle istunnon, jota pidetään yllä esimerkiksi evästeillä (4) [52]. Evästeillä tarkoitetaan palvelimen lähettämiä tietoja käyttäjän selaimen, joita tämä voi lähettää kyseisellä palvelimelle takaisin tehdessään uusia kyselyjä. Näin palvelin tunnistaa käyttäjän myös myöhemmillä kyselykerroilla.



Kuva 3.3: LTI 1.0:n toimintaperiaate [52, s.3]

LTI:stä on eri versioita, joista mainitsemisen arvoiset työmme kannalta ovat: 1.0, 1.1, ja 2.0. Nämä vaihtelevat integraation laajuuden suhteen siten, että 1.0 tarjoaa pienimmän integraation mahdollisuuden ja 2.0 korkeimman [37]. Versio 1.0 ei tarjoa muuta, kuin erillisen TP:n käynnistykseen TC:stä käsin. Se ei mahdollista

mitään palautteen saamista TP:stä TC:hen. Versio 1.1 sen sijaan tarjoaa työkalun käynnistämisen lisäksi yksittäisen kokonaisluvun palauttamisen TP:stä TC:hen, kun työkalun käyttösesio päättyy. Tämä ei usein riitä, koska työkalusta halutaan muutakin kuin arvosana. LTI 2.0 eroaa 1.1:stä siten, että TP ja TC voivat kommunikoida työkalun avaamisen jälkeen molempisuuntaisesti ja rajattomasti, sekä paljon monimutkaisemmin kuin yhdellä kokonaisluvulla, käyttäen REST-rajapintoja (kuva 3.4). Sekä TC, että TP toimivat siis kumpikin asiakkaina ja palvelimina. Esimerkiksi TC:n rajapinta sisältää resurssin nimeltä "result" eli tulos. Sille on määritelty metodit GET ja PUT siten, että GET-metodilla TP voi lukea tietyn tuloksen tiedot ja PUT-metodilla päivittää ne [1].



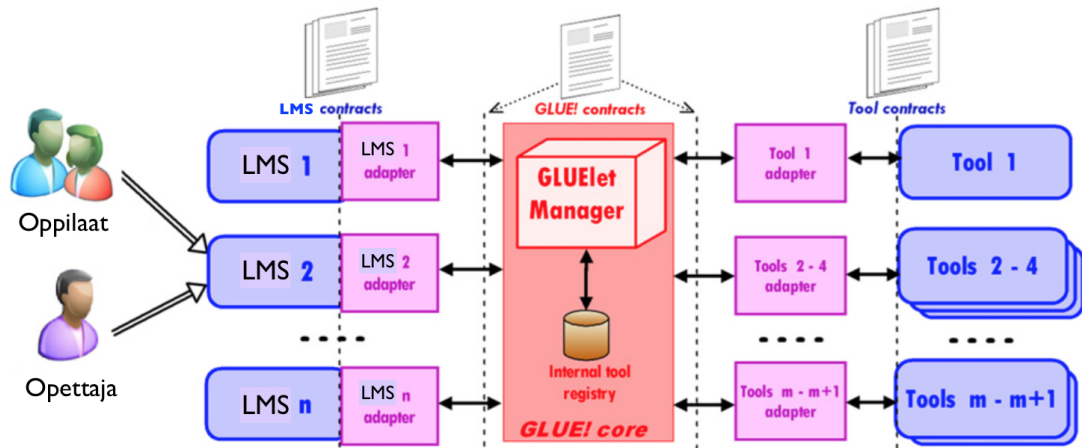
Kuva 3.4: LTI 2.0:n toimintaperiaate [37]

REST ja sen erityistapaus LTI perustuivat sopimukseen yhteisestä viestintäformaattista ja resurssien nimeämispolitiikasta. Joskus ei ole mahdollista tehdä tällaista sopimusta vaan täytyy käyttää eräänlaista sovitinta eli adapteria, joka tulkkaa komponenttien välisen viestinnän. Tästä esimerkkinä GLUE on eräänlainen adapteri, joka pyrkii yhdistämään LMS:siä ja lisäpalveluita toisiinsa.

3.2.3 GLUE!

GLUE (The Group Learning Uniform Environment) on integraatioarkkitehtuurimalli, joka perustuu 3-kerros- ja adapteri -arkkitehtuurimalleihin. LMS:ät ja ulkoiset työkalut voidaan yhdistää keskikerroksessa olevan ohjelmiston ja adaptereiden avulla. Kuvassa 3.5 vasemmalla näemme LMS-kerroksen, keskellä GLUEn ydinkerroksen ja oikealla työkalukerroksen. Jokaisen komponentin ja sen adapterin välillä

on REST-pohjainen sopimus, joka määrittelee, mitä tekniikoita, rajapintoja ja datamalleja näiden välillä käytetään. Myös adapterien ja keskeisemmän GLUE:n ydin-kerroksen välissä on sopimuksensa, jotka määrittelevät ytimen ja adapterien välisen toiminnan. GLUE-arkkitehtuuri mahdollistaa ns. monestamoneen -integraation, eli kun esimerkiksi järjestelmään integroidaan uusi työkalu, se on käytettävissä kaikista VLE:istä [4] [5].



Kuva 3.5: Glue -arkkitehtuuri mukailen artikkelin GLUE!: An architecture for the integration of external tools in Virtual Learning Environments [4] kuvaa 1

GLUE asettaa kolme ehtoa LMS:ille. Ensinnäkin LMS:n on pystyttävä muodostamaan graafista web-sisältöä, jotta työkalu voidaan sisällyttää LMS:n sivuille. Toiseksi LMS:sä on oltava lisäosarajapinta, jotta LMS-adapteri voi kommunikoida sen kanssa ilman LMS:n lähdekoodin muuttamista. Kolmanneksi, LMS:n täytyy ymmärtää työkalun konsepti, jotta integroinnissa ylipäätään olisi mitään mieltä [4].

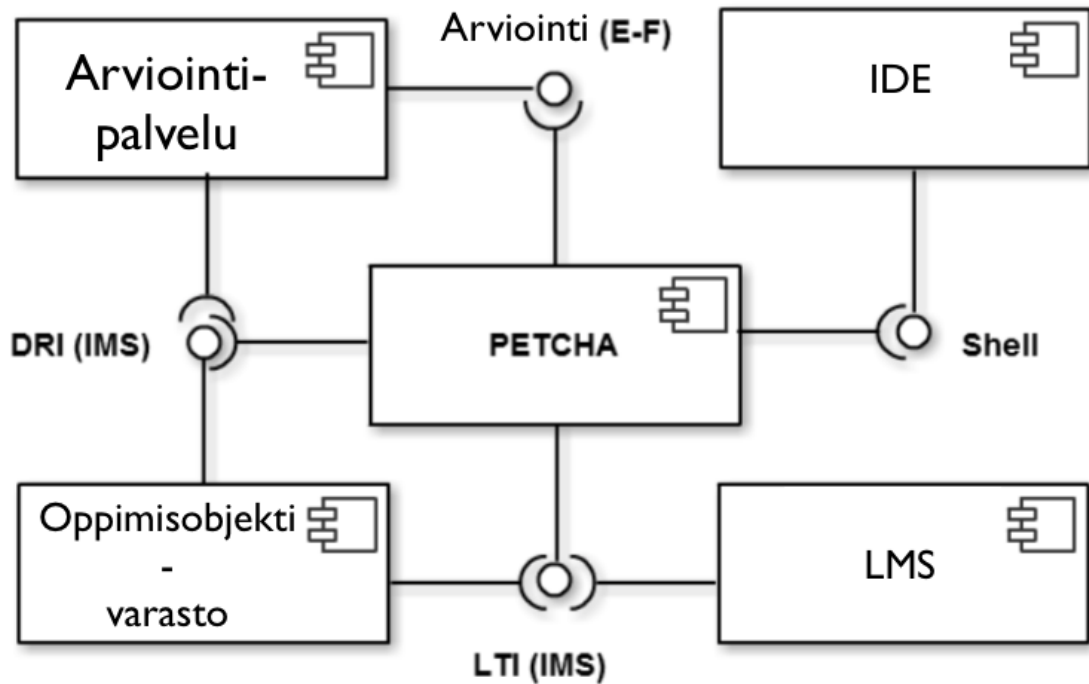
Tarkastelemme vielä järjestelmää nimeltä PETCHA, jossa on käytetty REST:iä, LTI:tä ja adapteria.

3.2.4 PETCHA

Queiros ja Leal esittelevät [47] arkkitehtuurin, joka rakentuu eräänlaisen keskuskomponentin ympärille. Koko oppimisympäristö voidaan rakentaa valmiista komponenteista, jotka yhdistetään toisiinsa tämän keskuskomponentin kautta.

Kuvassa 3.6 esitetään, miten komponentit yhdistetään. Queiros et al käyttävät keskuskomponenttina PETCHA-nimistä ohjelmaa, jonka he tekivät suunnittelun proof-of-conseptina. Kyseessä on jokaisen opiskelijan tai opettajan omalla tietokoneella ajettava ohjelma, joka integroituu tietokoneessa käytettävään IDE:en komentorivin kautta. Keskuskomponentti ei siis ole yksittäisellä keskuspalvelimella toimiva

ohjelma, vaan enemmänkin IDE:lle tehty adapteri, jonka kautta se voi kommunikoida muiden komponenttien kanssa [47, s.1458].



Kuva 3.6: PETCHA-järjestelmän arkkitehtuuri [48]

Muita komponentteja ovat automaattinen arviointikomponentti (Evaluation Engine), IDE, materiaaliarkisto (Learning Objects Repository) ja LMS. Nämä komponentit kommunikoiivat PETCHAN kanssa käyttäen opetusjärjestelmiä varten tehtyjä standardikommunikaatioprotokollia, joista LTI-protokolla esiteltiin jo [48, s.195]. DRI-standardi taas koostuu suosituksista, mitä rajapintafunktioita materiaaliarkistolle tulisi toteuttaa, ja esittää XML-muotoiset esitykset näille funktioille. Suosituksena on lisäksi, että nämä funktiot tulisi toteuttaa web-palvelurajapintoina käyttäen esimerkiksi REST-tyyliä [47, s.1461]. Automaattisen arviointikomponentin ja PETCHAN välinen kommunikointi tapahtuu arviointikomponentille tehdyn web-rajapinnan kautta, jonka määrittely löytyy E-Framework:ista [47, s.1468]. E-Framework on eräänlainen elektronisen oppimisen standardiohjeisto, joka pyrkii helpottamaan teknistä yhteensopivuutta eri oppimisjärjestelmien välillä [30, s.4].

Käyttäjän kirjautuminen tapahtuu LMS:ssä, josta oppilaan tiedot lähetetään keskuskomponentille käyttäen LTI 1.0-protokollaa [47, s.1466]. Keskuskomponentilla oppilas voi hakea tehtävänannot materiaaliarkistosta ja palauttaa tehtävän tarkastuskomponentille. Tehtävänannon haettuaan keskuskomponentti näyttää sen opiskelijalle ja luo uuden projektin IDE:en automaattisesti tehtävänannon määrittelyjen mukaisesti. Kun tehty ohjelmakoodi lähetetään tarkastimelle, tämä hakee kyseisen

tehtävän tiedot materiaaliarkistosta. Näiden tietojen avulla tarkastin arvostelee palautuksen ja lähettää tulokset keskuskomponentille [47, s.1469, 1470, 1473].

3.2.5 Tin Can API

Tin Can API on oppimistekninen spesifikaatio, joka mahdollistaa oppilaan oppimiskokemuksien keräämisen. API määrittelee eri järjestelmien välisen kommunikointisysteemin, jonka kautta järjestelmän eri osat lähettävät määritelmän mukaisia viestejä. Viestien muoto on ”substantiivi, verbi, objekti”, eli esim. ”Ville teki tehtävän 5” tai ”tein tämän tehtävän”. Nämä oppimiskokemukset voidaan tallentaa erilliseen LRS eli Learning Record Store -komponenttiin, joka voi olla täysin itsenäinen komponentti tai osa LMS:sää [73].

Teknisemmin tarkasteltuna viestit lähetetään HTTP:n POST-metodilla JSON-muodossa esim. OAuthilla suojatun yhteyden yli. JSONin muoto on esitetty listauksessa 3.1 esimerkissä, jossa Teemu Teekkari -niminen opiskelija palauttaa tehtävän nimeltä ”Moro maailma!” palautusautomaatille.

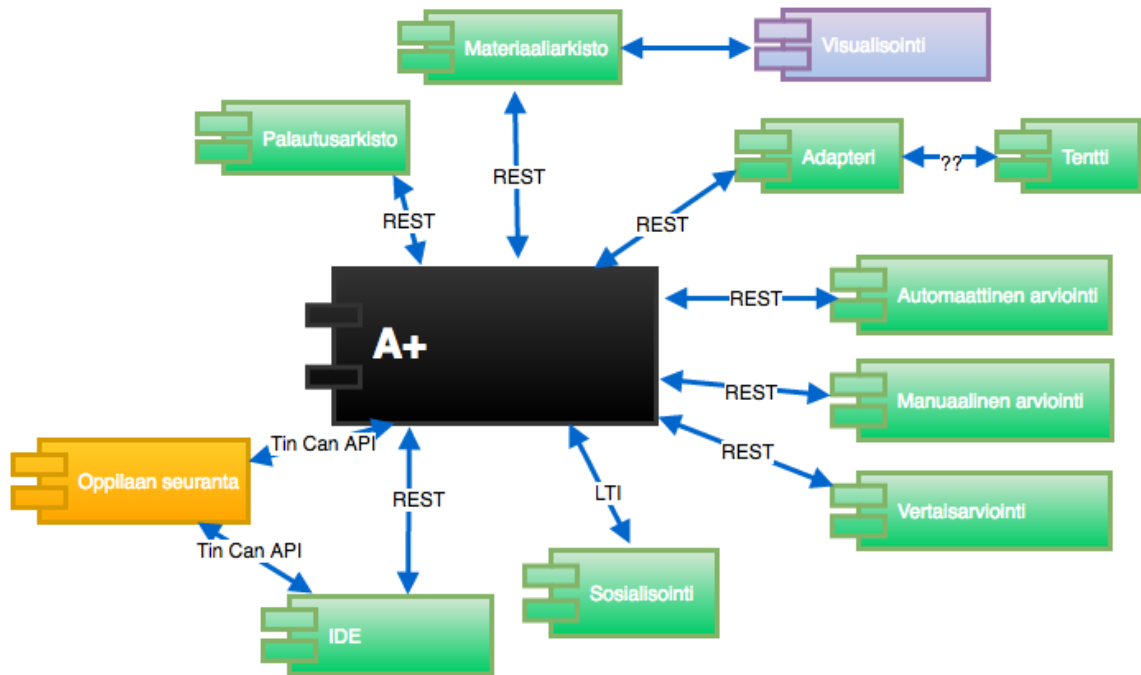
Listaus 3.1: Tin Can API:n viestiformaatti

```
{
  "actor": {
    "name": "Teemu_Teekkari",
    "studentID": "229565"
  },
  "verb": {
    "id": "http://grader.cs.tut.fi/",
    "display": {"fi-FI": "palautti"}
  },
  "object": {
    "id": "http://plus.cs.tut.fi/johoh/kk2016/moro",
    "definition": {
      "name": {"fi-FI": "Moro_maailma!" }
    }
  }
}
```

3.2.6 Järjestelmä prosessinäkökuilmasta

Jatketaan rakennettavan järjestelmän arkkitehtuurin suunnittelua prosessinäkökuilmasta. Komponenttien välisiä kommunikaatiotapoja ei tosin vielä voida päättää, koska käytettäviä komponenttien implementaatioita ei ole valittu. Kuvassa 3.7 on

kuitenkin esimerkkinä annettu eri vaihtoehtoja jokaisesta esittelystä kommunikaatiotavasta. Suurin osa komponenteista tultaneen yhdistämään A+:aan käyttäen sen REST-rajapintaa, koska A+ ei tue LTI 2.0:aa. LTI:tä käytetään aina, kun liitettävä komponentti sitä tukee, ja kun siltä ei tarvitse saada tietoja A+:aan. Tin Can API:a taas käytetään opiskelijoiden seurannan yhteydessä. Adapteria voidaan käyttää silloin, kun liitettävää komponenttia ei muuten saada yhdistettyä A+:n REST-rajapintaan.



Kuva 3.7: Arkkitehtuuri prosessinäkökulmasta

3.3 Fyysinen näkökulma

Fyysinen näkökulma tutkii komponenttien fyysisiä sijainteja. Oppilaiden tietokoneet ovat tästä näkökulmasta kiinnostavimmat ajoympäristöt, koska niiden määrää ja sijaintia ei tarkkaan tiedetä. Edellisessä luvussa esiteltyssä PETCHA-arkkitehtuurissa opiskelijan tietokoneelle sijoitettiin IDE ja adapteri, jonka kautta kaikki järjestelmän komponentit viestivät. Seuraavissa kahdessa alaluvussa tutustumme kahteen muuhun vaihtoehtoon siitä, mitä komponentteja opiskelijoiden tietokoneille voidaan sijoittaa.

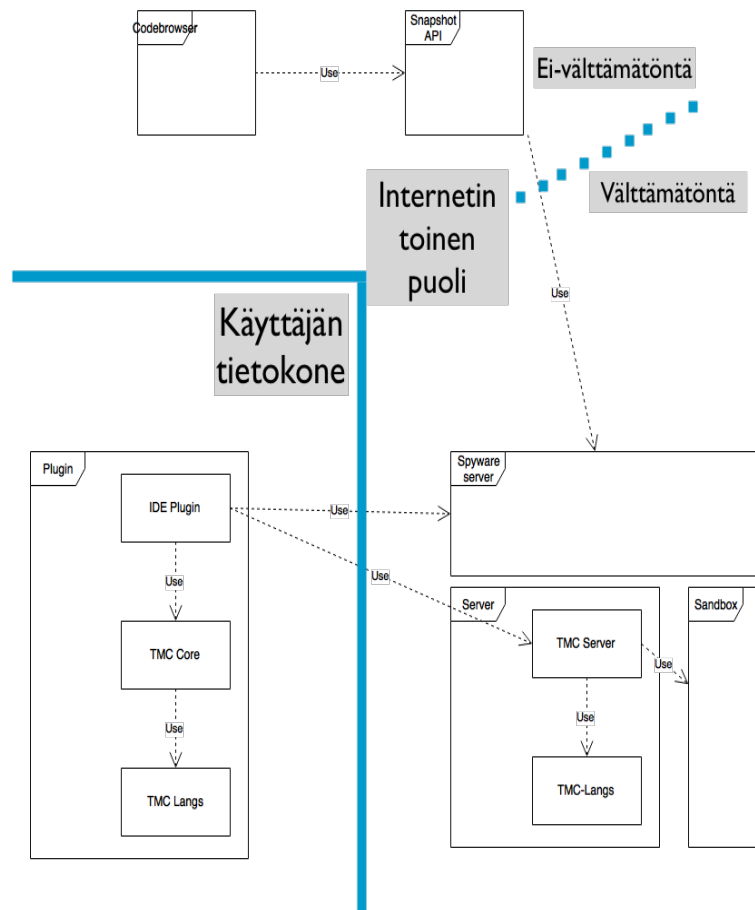
3.3.1 Test My Code

Test My Code on oppilaiden palautusten tarkastamiseen keskittyvä monesta komponentista koostuva ohjelmoinnin opetusympäristö. Tällä hetkellä saatavilla olevia

komponentteja ovat IDE-plugin, LMS:n tapainen TMC-palvelin, oppilaiden seurantaan tarkoitettu ”vakoilupalvelin” ja oppilaiden toteuttamien ohjelmien tarkastamiseen tarkoitettu hiekkalaatikkoympäristö [45]. Järjestelmä ei toistaiseksi tarjoa oppimateriaalia opiskelijoille.

Komponenttien väliset suhteet on esitetty kuvassa 3.8. Oppilaat asentavat omille koneilleen IDE:n, johon TMC-plugin lisätään. Tämän pluginin ansiosta IDE kommunikoi TMC-serverin kanssa mahdollistaen tehtävien automaattisen haun, sekä niiden lähettämisen arvioitavaksi suoraan IDE:n sisällä. Tehtävien mukana voidaan IDE:en tuoda myös ohjelmakoodipohjat ja testitapaukset, joilla opiskelija voi itse testata ohjelmakoodiansa [65]. TMC-serveri taas kommunikoi hiekkalaatikkoympäristön kanssa siten, että se lähettää IDE:stä saamansa ohjelmakooditiedostot arvioitavaksi hiekkalaatikkoon.

TMC-serverin lisäksi IDE-plugin kommunikoi myös ”vakoilupalvelimen” kanssa lähettämällä sille tietoa opiskelijan toimista IDE:ssä. Kurssihenkilökunta voi hakea ja analysoida näitä kaappauksia spyware-serveriltä erillisillä ohjelmilla, kuten Code Browserilla [20].



Kuva 3.8: TMC-järjestelmän arkkitehtuuri [45]

3.3.2 Mailing it in

IDE:n lisäksi käyttäjän tietokoneella voi sijaita myös muita komponentteja. Seuraavaksi esiteltävässä Mailing it in-järjestelmässä automaattinen tarkastin sijaitsee assistentin tietokoneella. Sant esittelee [51] tavan, jolla sähköpostiasiakasohjelmaan voi lisätä automaattisen tarkastusominaisuuden. Kun assistentti saa oppilaalta harjoitustyöpalautuksen sähköpostin kautta, hän voi sähköpostiasiakasohjelmansa käyttöliittymästä suorittaa automaattisen tarkastusskriptin, joka tarkistaa ohjelmakoodin tyylin ja ajaa ohjelmalle testitapaukset. Skriptin antama tulos ilmestyy ajon jälkeen sähköpostiohjelmaan, josta assistentti muokattuaan raporttia voi lähettää tuloksen oppilaalle. Oppilaiden palautukset arkistoidaan assistentin koneelle, joten myös arkistointikomponentti sijoittuu käyttäjän puolelle.

Mailing it in on vain yksi esimerkki oppimisympäristön käyttäjien tietokoneilla ajettavasta ohjelmakooditarkastuksesta. Helminen et al. esittelevät [21] käyttäjän selaimessa ajettavan ohjelmakoodieditorin, jolla voi ajaa ja testauttaa ohjelmakoodinsa. Lisäksi ohjelmakoodieditorissa on seurantaominaisuus, jolla opiskelijoiden edistymistä voidaan seurata. Karavirta et al. taas esittelevät [25] täysin opiskelijan omassa selaimessa ajettavan JavaScript-ohjelmointityökalun nimeltä js-assess, joka sisältää automaattisen ohjelmakooditarkastimen. Ohjelma ei lähetä tuloksia mihinkään, eli ohjelma soveltuu ainoastaan itseopiskeluun.

3.3.3 Järjestelmä fyysisestä näkökulmasta

Rakennettavan järjestelmän komponenttien fyysiset sijainnit voidaan jakaa kolmeen osaan: opiskelijan tietokone, yliopiston tietojärjestelmä ja muu maailma. Opiskelijan tietokoneella voi sijaita ainakin IDE, yliopiston verkossa ainakin A+, koska opiskelijoiden tietojen ei haluta päätyvän väärille tahoille ja muualla maailmassa sellaiset komponentit, jotka eivät sisällä tarkkoja tietoja opiskelijoista, esim. forumikomponentti. Yliopiston verkossa sijaitsevat komponentit voivat sijaita useammalla palvelimella, mutta tarpeen vaatiessa niitä kaikkia voidaan ajaa myös yhdellä palvelimella.

Kun lisäämme nämä fyysisen näkökulman asiat edellisissä näkökulmissa muodostettuun kuvaan 3.7, saamme kuvan 3.9 mukaisen arkkitehtuuriratkaisuun. Kuvan komponentteihin on myös lisätty edellisessä luvussa esille tulleet vaatimukset.

4. KATSAUS OLEMASSAOLEVIIN JÄRJESTELMIIN JA KOMPONENTTEIHIN

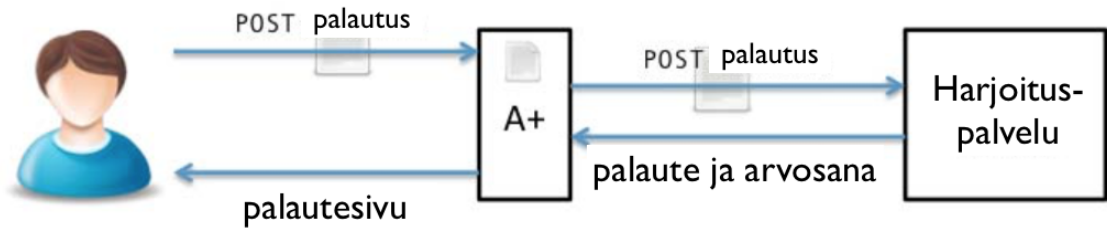
Tässä luvussa tutustutaan muutamiin valmiisiin komponentteihin, joista valitaan käytettäväksi parhaiten kurssimme tarpeisiin soveltuvat. Kelpuutamme ainoastaan avoimen lähdekoodin komponentit niiden ilmaisuuden ja laajennettavuuden takia. Lisäksi valintaan kaikkien komponenttien suhteen vaikuttaa se, tukeeko komponentti LTI-protokollaa, onko tekijältä saatavilla tukea integroinnissa ja kehitetäänkö komponenttia enää aktiivisesti. Valintaperusteista kuitenkin suurin komponenteille on se, miten helposti ne saadaan integroitua A+:-an.

4.1 Oppimisenhallintajärjestelmät

A+, jota tulemme uudessa järjestelmässämme käyttämään, esiteltiin lyhyesti johdannossa ja tutustumme siihen nyt tarkemmin. A+ on toteutettu Django-websoveluskehysellä, jonka lisäksi ulkoisten komponenttien autentikointiin käytetään OAuthia ja REST-rajapinnan toteutukseen TastyPie-nimistä kirjastoa [26] [32]. Käyttäjien tunnistaminen voidaan tehdä yliopiston tunnuksilla käyttäen Shibboleth-protokollaa [32]. Tällöin A+ siirtää tunnistamisvaiheen yliopiston tunnistautumisjärjestelmän vastuulle ja odottaa siltä vastausta, onko henkilö se joka väittää olevansa.

A+ tarjoaa käyttäjien tietojen, kuten tehtyjen tehtävien ja arvosanojen tallettamisen, sekä tietoturvallisen kirjautumisen. Tehtävänantojen säilyttämiseen ja tehtävien arvioimiseen tarvitaan erillinen palvelu. A+ sisältää tällaiselle palvelulle valmiin REST-kommunikaatorajapinnan. Sen tekijätiimi on myös toteuttanut Mooc-grader-nimisen palvelun, joka osaa käyttää tätä rajapintaa. Oppilaan tehtävän tarkistus toimii kuvan 4.1 mukaisesti: Kun oppilas painaa selaimessaan palautusnappulaa, lähtee hänen tekemänsä ratkaisu A+:-lle HTTP:n POST-metodilla. A+ tallentaa palautustiedostot ja lähettää ne eteenpäin tarkistettavaksi tarkistuspalvelulle. Tälle lähetykselle on olemassa kaksi tapaa, synkroninen ja asynkroninen. Synkronisessa tapauksessa A+ jää odottamaan vastausta arviointipalvelulta. Asynkronisessa A+ antaa arviointipalvelulle linkin, johon se saa arvostelun valmistuttua lähettää vastauksensa. Palvelun lähettämä vastaus on normaali HTML-sivu, joka sisältää arvostelutiedot. A+ ottaa tästä talteen arvostelutulokset ja lähettää sivun eteenpäin käyttäjälle [26]. REST-rajapinnan lisäksi A+ sisältää LTI 1.0 -protokollan mukaiset

erillisten palveluiden kutsuimetodit [32].



Kuva 4.1: Tehtävän arvostelu erillisessä palvelussa [26, s.3]

Vertailun vuoksi tutustumme myös Test My Code -järjestelmän tehtäväpalvelimeen, sekä LMS:sään nimeltä Sakai.

Test My Code -järjestelmään tutustuttiin jo edellisessä kappaleessa arkkitehtuurillisesti. TMC-palvelin on LMS:sän tapainen ohjelma, jossa opettaja pystyy tekemään kurssipohjia ja näiden perusteella kursseja, asettamaan määräaikoja, arvostelemaan tehtäviä käsin, asettamaan tehtäviä vertaisarvioitaviksi sekä katselamaan kurssin tilastoja, kuten kuinka paljon mitään tehtäviä on palautettu [46] [67].

TMC-palvelimella on REST-rajapinta, jota NetBeans-plugin ja muut asiakassovellukset voivat käyttää. Sen kautta voidaan lisäksi mm. hakea tietoja palautuksista [24]. Käyttäjänhallinta tapahtuu TMC-järjestelmässä itsessään, eli opiskelijat joutuvat luomaan alussa tunnukset järjestelmään, eikä esimerkiksi yliopiston tunnuksia pystytä käyttämään. TMC kuitenkin tukee OAuthia, joten autentikointitietojen siirtäminen muille komponenteille on mahdollista [68].

Sakai on avoimen lähdekoodin LMS, joka sisältää kaiken oppimisympäristössä tarvittavan kalenterista, wikiin ja harjoitustehtävägeneraattoriin asti. Tämän lisäksi se tukee LTI:tä versioon 2.0 asti, siinä on laajennettava REST-arkkitehtuurin rajapinta ja toimii Tin Can API:n kanssa [50].

Sakai sisältää käyttäjien hallinnan, heidän jakamisensa ryhmiin ja jokaiselle oman sosiaalisen profiilin tekemisen. Se myös tukee monia standardisoituja kertakirjaus (SSO) -tekniikoita, kuten SAML ja Shibboleth. Opiskelijoiden seuranta varten Sakai:ssa on tilastointityökalu, jolla voi tutkia sivuilla olevien resurssien käyttöä [50].

4.2 Automaattinen arviointi

Koska valitsemamme A+ ei sisällä automaattista ohjelmakoodin tarkistinta, käytämme erillistä ohjelmaa. Tässä alaluvussa esitellään Mooc-grader, joka toimii pienellä asennustyöllä A+:n kanssa.

Mooc-grader on A+:n kehittäjätiimin tekemä automaattinen ohjelmakooditarkistinympäristö, joka toimii A+:n kanssa vaatimatta lainkaan rajapintamuokkauksia. Ympäristö käyttää chroot-hiekkalaatikkaa oppilaiden ohjelmakoodien eristettyyn ja turvalliseen ajoon. Se vastaanottaa tarkastettavat ohjelmakoodit A+:lta joko synkronisesti tai asynkronisesti HTTP:n kautta. Järjestelmä palauttaa testituloksen A+:lle tämän antamaan palauteosoitteeseen, kun testit on suoritettu. Itse testaamiseen Mooc-grader ei ota kantaa, se vain tarjoaa ympäristön testien turvalliseen ajamiseen ja kommunikoimiseen oppimishallintajärjestelmän kanssa. Testaukseen käytettävä testiskripti on siis tehtävä itse, mistä kerrotaan enemmän kohdassa 5.3.

Jokaiselle tehtävälle on oma palautussivunsa, joka tuotetaan yaml-merkintäkielellä tehdystä pohjasta. Tällainen pohja sisältää oppilaalle näytettävät ohjeet, palautustavan määrittelyn, joka voi olla esimerkiksi tiedoston siirto oppilaan koneelta palvelimelle, sekä tarkastuksessa tehtävät toiminnot. Nämä toiminnot ovat esimerkiksi testitapausten kopioiminen jostain hakemistosta hiekkalaatikkoon ja tiettyjen kommentojen ajaminen tämän sisällä. Komento voi olla esimerkiksi seuraavanlainen:

```
- type: grader.actions.sandbox
  cmd: ["testscript.py", "1_6testit", "oppilaan_koodi.py", "3"]
```

Tämä ajaa testauskriptin ja antaa sille parametrina kansion nimen, jossa kyseisen tehtävän testitapaukset ovat, oppilaan palauttaman ohjelmakoodin sekä maksimipistemäärän, jonka tehtävästä voi saada. Testiskripti voi suoriuduttuaan yksinkertaisesti tulostaa pistemäärät, jotka Mooc-grader jäsentää tulosteesta ja palauttaa A+:lle tulossivun mukana [33].

4.3 Manuaalisen palautteen antaminen

A+ tarjoaa yksinkertaisen lomakenäkymän palautteen antamiseen (kuva 4.2). Assistentti voi muokata automaattisen tarkastimen antamaa pistemäärää ja palautesivua, minkä lisäksi hän voi antaa erillisessä lomakkeessa palautetta. Kun assistentti tallentaa arvionsa, oppilas saa A+-tiliinsä ilmoituksen tästä. Palautteen antamiseen ei ole tarjolla erityisempiä apuvälineitä vaan palauteteksti pitää kirjoittaa käsin tai kopioida muualta.

Palautuksen tiedot

Opiskelijat
Ville Vironmäki (██████)

Tila
ready

Palautusaika
20.7.2016 keskiviikko 03:08

Arvostelu
15 / 15

Palautetut tiedostot
10.1.painoindeksi_laskuri.py 5,8 KB

Palautekentät
Ei ole

Arvostelun tiedot

points
15

max_points
15

feedback
<div id="feedback"> <h1> Kokonaispisteet 15/15 </h1> <div class="grading-task"> Pisteet 15/15 <p>Koodissaasi on tehty seuraavat kohdat: a, b, c, d, e Antamisesi tietojen pohjalta saat alustavasti pisteitä 15/15. Assistentti tarkistaa pisteet kä aikanaan.</pre> </div> </div>

Arviointi

Points
15

Mahdollisia myöhästymisvähennyksiä ei tehdä automaattisesti. Pisteet korvaavat automaattisen arvioinnin tulokset!

Assistant feedback

+ Hieno käytöllitymä
+ Perustoinnot ok.
- Nollalajakoa ei ole otettu huomioon.

HTML-muotoilut sallitaan. Tämä ei korvaa automaattista palautetta.

Feedback

```
<div id="feedback">
<h1>
Kokonaispisteet 15/15
</h1>
```

HTML-muotoilut sallitaan. Tämä KORVAA automaattisen palautteen.

Tallenna Peruuta

Kuva 4.2: Palautteen antaminen A+:-ssa

Manuaalisen palautteen antamiseen haluttiin ohjelma, joka tarjoaa assistentille valmiit arvostelufraasit ja niiden nopean muokkaamisen sopivaksi. A+ ei näitä ominaisuuksia tarjoa, joten tutustumme ohjelmaan nimeltä Rubyric, jota on käytetty Aalto-yliopiston ohjelmointikursseilla yhdessä A+:-n kanssa. Tyydymme kuitenkin tämän työn puitteissa A+:-n omaan arvostelutyökaluun ja jätämme Rubyricin integroinnin jatkokehityskohteeksi.

Rubyric on verkkopohjainen arviointityökalu, joka käyttää ns. rubriikkeja palautteen antamiseen. Rubriikeilla tarkoitetaan arvostelusapluunoita, jotka sisältävät arvostelukriteerit ja valmiita lauseita, jotka kertovat onnistumisista ja epäonnistumisista tehtävän teossa. Rubriikit ovat taulukkomuodossa siten, että riveinä on eri ohjelmoinnin osa-alueita, kuten tyyli ja algoritmit sekä sarakkeina taso kuinka hyvin kussakin osa-alueessa on menestytty. Soluissa taas on valmiita lauseita, jotka kertovat opiskelijalle, kuinka hän menestyi (kuva 4.3).

Taulukko 1: Lyhyt esimerkkirubriikki ohjelmointiprojektin arvosteluun

	Heikko	Kelvollinen	Hyvä
Ohjelmointityyli	Et ole noudattanut tyyliohjeita.	Koodi on melko siistiä, mutta paikoin poikkeaa tyyliohjeista.	Koodi on kauttaaltaan siistiä ja helppolukuista.
Testit	Projektissa ei ole käytetty yksikkötestejä lainkaan.	Projektissa on käytetty jonkin verran yksikkötestejä.	Yksikkötestit ovat hyvin kattavat.
Arkkitehtuuri	Ohjelma kannattaisi jakaa selkeämmin itsenäisiin osiin.	Luokkien väillä on jonkin verran tarpeettomia riippuvuuksia.	Ohjelman rakenne on hyvin modulaarinen.

Kuva 4.3: Esimerkkirubriikki [7]

Kurssimme puitteissa rubriikkien käyttö on perusteltua, koska monilla opiskelijoilla on yleensä samanlaisia virheitä, jolloin valmiita fraaseja on mielekästä käyttää. Lisäksi rubriikit tarjoavat selvät arviointikriteerit, mikä mahdollistaa opiskelijoiden tasapuolisen arvioinnin.

Rubyric vastaanottaa opiskelijoilta palautuksia oman käyttöliittymänsä tai rajapintansa kautta, joten palautukset voidaan tuoda A+:sta Rubyriciin automaattisesti. Assistentit voivat tehdä Rubyriciin käyttäjätunnukset, joiden avulla varsinainen arvostelu tehdään. Ohjelma jakaa automaattisesti palautetut tehtävät assistenteille, joten tätä ei tarvitse tehdä manuaalisesti.

Valmiiden fraasien lisäksi assistenteilla on mahdollisuus muokata palautetta yksilöllisemmäksi. Esimerkiksi assistentti voi nostaa esille jonkin tietyn huonosti nimetyn funktion [7]. Assistentin arvostelunäkymä on esitetty kuvassa 4.4.

Review

Group: 00001 - g4_1.m (2009-02-01 18:06:08 UTC)

Programming

Implementation DONE

Testing DONE

Documentation

General DONE

Class diagrams DONE

Finish

Modular design Poor OK Good

All the functionality should not be implemented in one class. Modular design is important for the reusability of source code.

The classes have unnecessary dependencies. Pay more attention to modular design.

The program has been divided into loosely coupled modules. Very good.

Efficiency Poor OK Good

The program uses a brute-force algorithm. It will not be able to handle large inputs.

The algorithms are reasonably efficient.

The algorithms are very efficient.

Coding style Poor OK Good

You have not followed the coding conventions recommended on this course. Coding conventions are very important for maintainability.

You should pay more attention to coding conventions. Clean code is easier to maintain.

The code is very clean.

[Hide phrases](#) [Show phrases](#)

Grade: 4

[Save and proceed ->](#)

Strengths

The program has been divided into loosely coupled modules. Very good. The algorithms are reasonably efficient.

Weaknesses

You have not followed the coding conventions recommended on this course. Coding conventions are very important for maintainability. You should always use descriptive variable names.

Other comments

You could also consider hash maps for storing the data.

Figure 4.5: The assessment view of Rubyric

Kuva 4.4: Rubyricin assistentin käyttöliittymänäkymä [6, s.40]

4.4 Vertaisarviointi

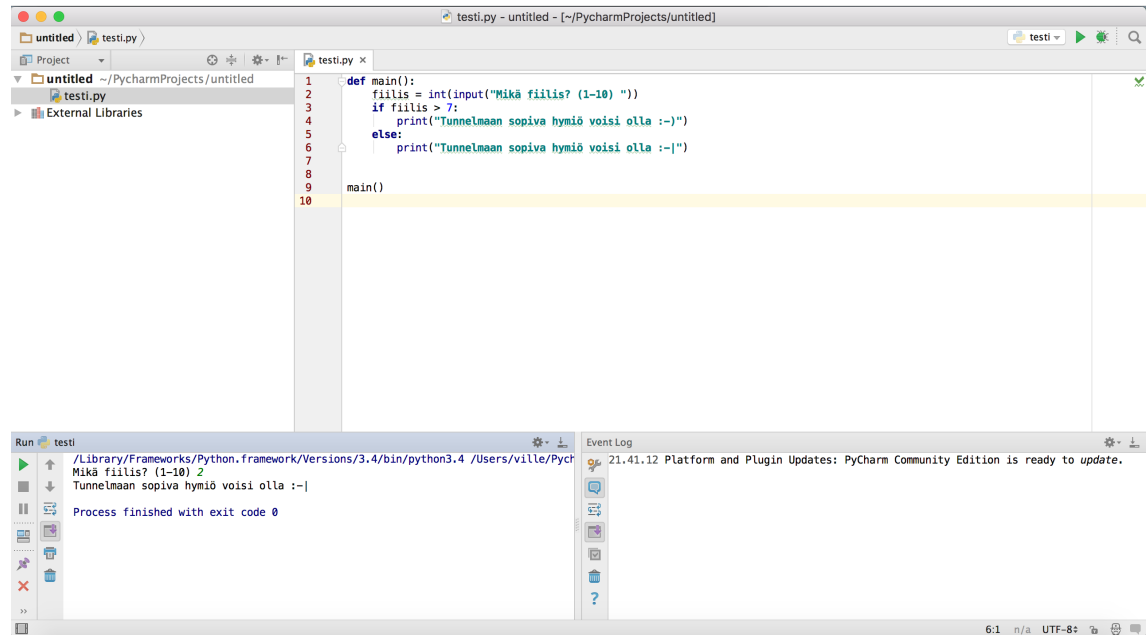
Emme löytäneet kirjallisuuskatsauksessamme avoimen lähdekoodin ratkaisuja LMS:ään ulkopuoliseen vertaisarviointiin, joten tämä jätetään jatkokehityskohteeksi. Esittelemme silti komponentin nimeltä WebPA, jonka avulla ryhmätyön tekijät voivat arvostella toistensa työpanoksia.

WebPA on ryhmätöiden sisäiseen vertaisarviointiin tarkoitettu työkalu. Assistentti antaa ryhmätyölle arvosanan, minkä jälkeen oppilaat arvioivat toistensa työpanokset. Näiden arvioiden avulla WebPA painottaa opiskelijoille yksilölliset arvosanat koko työlle annetusta arvosanasta. Tämän toiminnallisuuden ansiosta ryhmätöissä voidaan antaa jokaiselle oikeudenmukainen arvosana, eikä kukaan ryhmäläinen pääse ”vapaamatkustajana” kurssia läpi tekemättä mitään [71]. WebPA:lle on saatavilla LTI-lisäosa, jonka avulla se on liitettävissä LMS:sään helposti [36].

4.5 IDE

PyCharm (kuva 4.5) on Python IDE, joka sisältää monipuoliset työkalut ohjelmakoodin kirjoittamiseen mm. ohjelmakoodin värjäys, automaattinen sisennys ja

ohjelmakoodintäydennys. Lisäksi IDE sisältää ohjelmakoodin ajamiseen konsolin ja debuggerin [43].

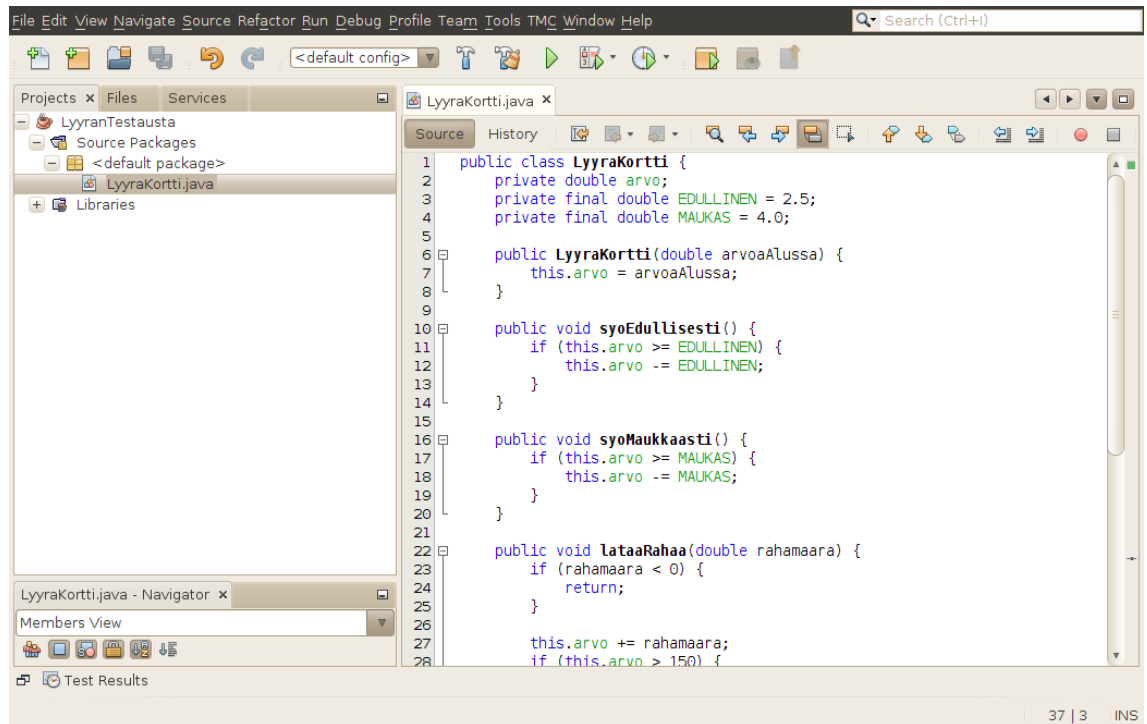


Kuva 4.5: Kuvankaappaus PyCharm IDE:stä

PyCharm IDE on todettu varmatoimiseksi ja sopivan kevyeksi tämän kurssin kannalta, joten se halutaan pitää jatkossakin oppilaiden IDE-työkaluna. Vertailun vuoksi tutustumme kuitenkin vielä NetBeans IDE:en, jota käytetään aiemmin esitellyssä TMC-järjestelmässä.

NetBeans IDE (kuva 4.6) on erityisesti Java-ohjelmointiin tarkoitettu ohjelmointiympäristö, johon tosin on saatavilla Python-plugin, jossa on mm. ohjelmakoodin automaattinen täydennys, debugger ja ohjelmakoodin väritys [44] [15].

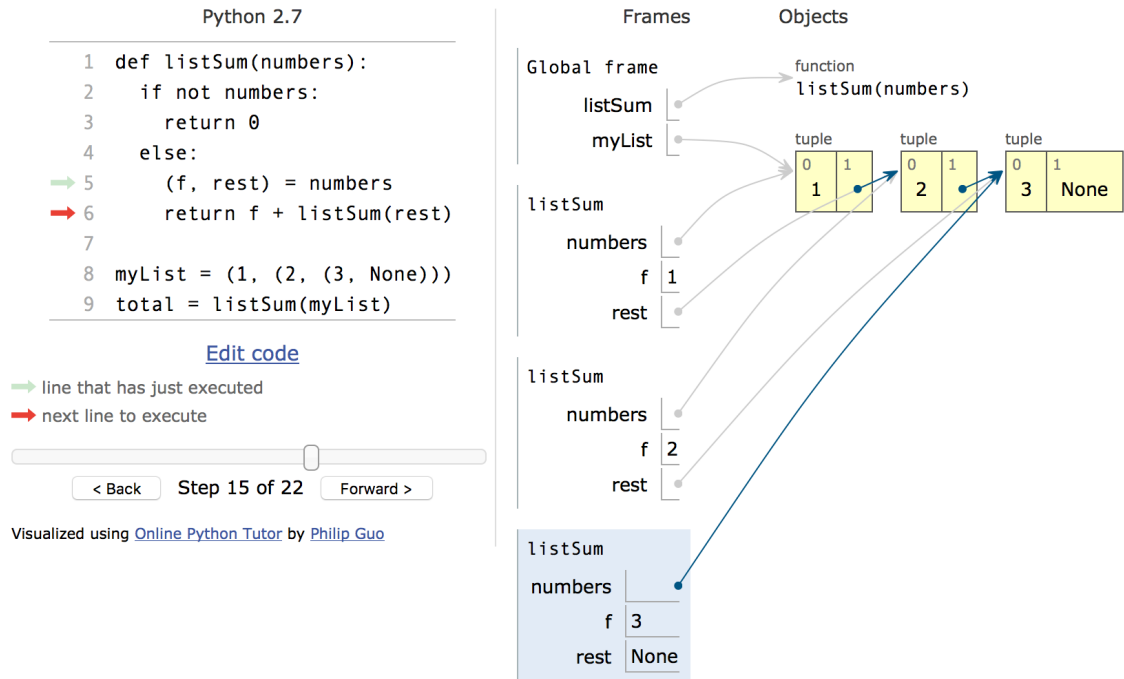
NetBeansille on saatavilla Test My Code -järjestelmään kuuluva plugin, jolla voi palauttaa tehtäviä suoraan IDE:stä ja hakea TMC-serveriltä eri tehtävissä tarvittavat tiedostot ja generoida niille kansiot.



Kuva 4.6: Kuvankaappaus NetBeans IDE:stä [38]

4.6 Visualisointi

Python Tutor (PT) on web-pohjainen ohjelmavisualisointiohjelma, jota on käytetty elektronisissa kirjoissa, kuten *How to Think Like a Computer Scientist: Interactive Edition*, näyttämään oppilaille ohjelmakoodipätkien toimintaa [17]. Python Tutor tukee Pythonin versioita 2.7 ja 3.3 sekä mm. C++:aa, Rubya ja Javaa. Kuvassa 4.7 näkyy Python Tutorin tekemä visualisointi rekursiivisen funktion toiminnasta. Huomionarvoinen asia visualisoinnissa on se, että PT visualisoi yhden ohjelmakoodirivin kerrallaan, eikä visualisoi rivin sisäkkäisiä käskyjä. Esimerkiksi jos rivillä on yhdistetty input-funktio, sekä sen palauttaman merkkijonon muuttaminen int-funktiolla kokonaisluvuksi, eli "a = int(input("Anna luku: "))", visualisoinnissa kokonaisluku ilmestyy muistiin suoraan.

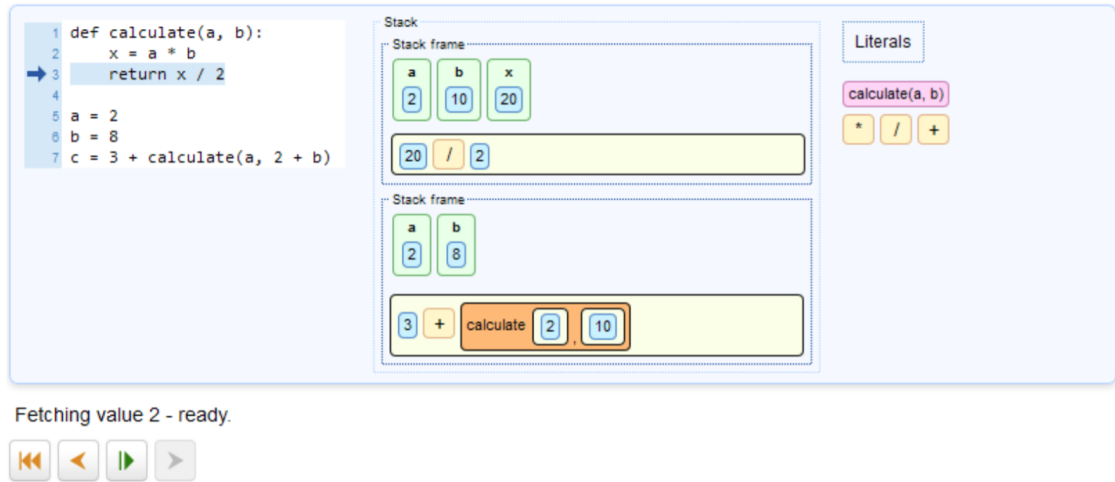


Kuva 4.7: Python Tutor -visualisaation

PT koostuu käyttöliittymästä sekä palvelinpuolesta, jossa visualisoitava ohjelmakoodi ajetaan standardilla Python-debuggerilla (bdb). Koska opiskelija voi myös syöttää omaa ohjelmakoodiansa Tutorin suoritettavaksi, täytyy ohjelmakoodi suoritaa turvallisessa ympäristössä, eli hiekkalaatikossa. Ohjelmakoodia ajetaan debuggerilla rivi kerrallaan ja tallennetaan muistiin erinäisiä asioita, kuten käskytyyppi, rivinumero jne. Lopuksi kaikki tallennettu data lähetetään JSON-muodossa käyttöliittymälle, jossa visualisointi tehdään JSON-tiedoston tarjoamalla parametreilla [17].

Python Tutor-visualisointi voidaan sisällyttää helposti tavalliselle internetsivulle käyttäen iFrame-tagia, joka on HTML-kielen keino sisällyttää toinen internetsivu toisen sisälle. Visualisaatio tehdään tällöin oikeasti Python Tutorin tekijöiden palvelimella, josta se kopioidaan halutulle sivulle [16]. Sisällyttämisen helppouden takia käytämme rakennettavassa järjestelmässä Python Tutoria visualisoinneissa aina kun mahdollista, eli kun yksittäiset ohjelmakoodirivit eivät sisällä monia sisäkkäisiä käskyjä. Tarkempaa visualisointia varten käytämme Jsveetä.

Jsvee on A+⁺:n tekijätiimin tekemä ohjelmavisualisaattori, joka esittää ohjelmakoodin toiminnan lausekkeittain, kun Python Tutor teki tämän riveittäin. Jsvee siis osaa ottaa huomioon kaikki toiminnot yksittäisellä rivillä, mikä auttaa oppilaita ymmärtämään ohjelman toimintaa paremmin. Ongelmana tässä kuitenkin on, että visualisaatiot voivat olla hyvinkin pitkiä ja liian yksityiskohtaisia [54]. Kuvassa 4.8 on esitelty Jsvee:n käyttöliittymä.



Kuva 4.8: Jsveen tekemä lausekepohjainen visualisointi [54, s.4]

Jsveen visualisointeihin voi lisätä animoituja tekstejä ja nuolia käyttäen lisäosaa nimeltä Kelmu. Tämä tarjoaa yksinkertaisen web-käyttöliittymän, jonka avulla animaatioita voi tehdä ilman ohjelmakoodiin koskemista.

Jsvee-visualisaation sisällyttäminen materiaaliin tapahtuu lisäämällä halutulle html-sivulle Jsveen tyyli- ja JavaScript-kirjastot sekä Python-ohjelmakoodista laskevat parametrit, joita Jsvee käyttää tehdäkseen animaation. Parametrit voidaan laskea syöttämällä visualisoitava ohjelmakoodi tekijätiimin sivuilla olevalle työkalulle nimeltä Jsvee transpiler [53]. Tämä prosessi on suhteellisen työläs verrattuna Python Tutorin tapaan sisällyttää visualisointi internetsivuille. Jsveessä tätä tapaa ei voida käyttää, koska tekijät eivät tarjoa samanlaista palvelua, josta ohjelmakoodin syöttäessä saisi vastauksena visualisaation. Tämän takia käytämme Jsveetä vain monimutkaisimpien ohjelmien visualisointiin.

4.7 Oppimisen seuranta

Edellisessä luvussa esitellylle Tin Can API-arkkitehtuurille on olemassa valmiita kirjastoja lukuisille ohjelmointikielille, kuten JavaScriptille, Pythonille ja Javalle [57]. Täten voimme suhteellisen helposti implementoida seuraamisjärjestelmän sekä IDEen, että LMS:sään. LRS:n taas voi hankkia avoimen lähdekoodin projektista, ostaa kaupallisena tai tehdä itse [14]. Asennetuin avoimen lähdekoodin LRS on Learning Locker. Se tarjoaa seurantakomponenteille helppokäyttöisen REST-rajapinnan OAuth-autentikoinnilla. Kerättyä dataa voidaan analysoida ohjelman omassa käyttöliittymässä tai siirtää muihin analytiikkaohjelmistoihin [31].

Emme tule tämän työn puitteissa implementoimaan opiskelijoiden seurantakomponenttia, mutta jatkokehityksen kannalta Tin Can API vaikuttaa lupaavalta tekniikalta. Sillä pystyy keräämään dataa monesta eri lähteestä, datan muoto on selkeä

(kts. 3.2.5) ja sen voi kerätä yhteen keskitettyyn paikkaan eli LRS:ään. Verrokkina Tin Can API:lle tutustumme vielä TMC-järjestelmässä olevaan oppilaidenseuranta-komponenttiin.

TMC ”vakoilupalvelin” kerää TMC IDE-pluginien ottamia kaappauksia oppilaiden ohjelmakoodeista. IDE-plugin kaappaa jokaisen näppäimenpainalluksen (sekä hiiren, että näppäimistön), joka IDE:n sisällä tehdään ja lisää tähän tilannevedoksi kutsuttuun lokitietoon oppilaan tiedot, lähdekoodin muutoksen edellisestä painalluksesta, aikaleiman sekä ratkaistavan tehtävän numeron. Lähdekoodin muutos tallennetaan diff-muodossa, minkä lisäksi tallennetaan tieto siitä, asetettiinkö, poistettiinkö vai liitettiinkö uutta tekstiä. Tämän seurantaominaisuuden tarkoitus on tarjota opettajille työkalu niiden ongelmien, joihin oppilaat törmäävät, näkemiseen ja analysoimiseen. Opettava voi esimerkiksi nähdä, mitä syntaksivirheitä oppilaat eniten tekevät [64].

Kyseinen seurantaominaisuus on NetBeans-pluginin sisäinen ominaisuus, eikä sitä saa suoraan siirrettyä muuhun komponenttiin, kuten LMS:sään. Täten tällä komponentilla ei onnistu materiaalin käytön seuranta.

4.8 Tehtävien ja materiaalin arkistointi

Ohitamme tässä työssä myös erillisen tehtävä- ja materiaaliarkiston implementoinnin, koska A+ ja Mooc-grader tarjoavat näihin ratkaisut. A+ tallentaa opiskelijoiden palautukset käyttämäänsä tietokantaan. Assistentit ja opettajat voivat katsoa näitä A+:n web-käyttöliittymän kautta. Tämä riittää yksittäisen toteutuskerran palautusten arkistointiin, mutta kurssin ollessa ohi nämä tulee kopioida erilliselle palvelimelle tai muuhun talletusmediaan. Varmuuskopiointi voidaan tehdä monella tavalla, joihin ei tässä työssä perehdytä.

Mooc-grader-komponenttia ajavaan palvelimeen voidaan tehdä materiaalit ja tehtävänannot sisältävä Git-kansio polkuun *mooc_graderin_asennuspolku/exercises*. Mooc-grader osaa poimia tämän kansion sisältämät materiaalit, kunhan ne on indeksoitu käyttämällä erityistä index.yaml-tiedostoa. Materiaalin teosta ja käytöstä kerrotaan tarkemmin seuraavassa luvussa.

4.9 Sosialisointi

Sosialisointikomponenteista tutustumme Piazza- ja Vanilla -nimisiin komponentteihin. A+:n kanssa on käytetty aiemmin Piazza-komponenttia, sekä se sisältää tuen LTI 1.0-standardille, joten valitsemme järjestelmäämme sen, mutta tutustumme vertailun vuoksi myös Vanillaan.

Piazza on keskustelualusta, johon oppilaat voivat lähettää omalla nimellään tai anonyymisti kysymyksiä joihin muut oppilaat ja assistentit voivat vastata. Kysymyksiä ja niihin tulleita vastauksia on helppo selata, koska kysymyksiin tulleista vastauksista näytetään vain yksi oppilaan ja yksi assistentin antama vastaus. Näytettävä vastaus valitaan sen saamista äänistä, joita osallistujat saavat antaa. Mitä parempi vastaus, sitä enemmän se saanee ääniä [74].

Keskustelut päivittyvät reaaliaikaisesti, eli heti kun viesti on lähetetty se näkyy muille. Lisäksi Piazzasta on tehty mobiililaitteille ohjelmat, joilla oppilaat voivat seurata keskustelua mistä vain [74].

Piazza voidaan integroida LMS:sään käyttäen LTI 1.0 -standardia, eli palveluun voidaan siirtyä LMS:stä siten, että käyttäjän tunnistustiedot lähtevät siirtymisen yhteydessä Piazzalle [2]. Tosin ensimmäisellä käyttökerralla opiskelijan täytyy antaa automaattisesti luodulle Piazza-käyttäjätunnukseensa salasana, jotta hän voi kirjautua siihen muualta, kuin A+:sta, esimerkiksi puhelinsovelluksesta.

Vanilla sisältää Piazzassa olevan kysymys-vastaus -tyylisen osion, minkä lisäksi siinä on mm. perinteinen keskustelufoorumi, mahdollisuus lähettää yksityisviestejä ja pelillistämisominaisuus. Oppilaat voivat viesteillään ansaita mitaleja ja arvonimiä, jotka taas oikeuttavat lisäämään viesteihin joitain lisäominaisuuksia, kuten allekirjoituksia [63].

Vanilla tukee SSO:ta, sekä tarjoaa helppokäyttöisen REST-rajapinnan, jonka avulla Vanilla voidaan integroida melko tiiviisti LMS:sään, mikä tosin vaatii paljon työtä. LTI-protokollaa varten Vanillalle on saatavilla plugin, joka tosin on melko vanhalle versiolle tehty, joten sen toimivuudelle ei ole takeita.

4.10 Tentti

Kappaleessa 2 mainittiin EXAM-niminen sähköinen tenttijärjestelmä. Kyseessä on REST-rajapintoja käyttävä palvelu, joka voidaan integroida yliopiston opintohallinnon järjestelmään. Järjestelmästä voidaan rajapinnan kautta hakea opintojakson tiedot, sekä lähettää sinne opiskelijan tentistä saama arvosana. Lisäksi EXAM tunnistaa käyttäjät yliopiston käyttäjänhallintajärjestelmän avulla [61].

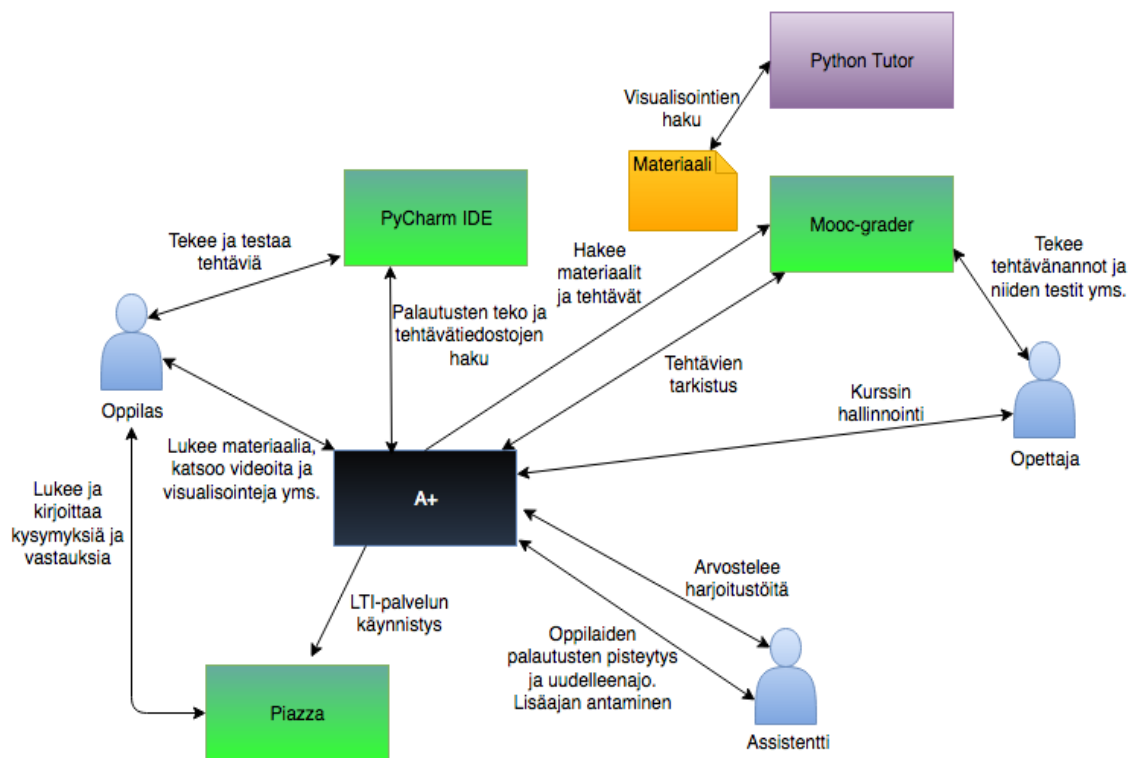
Tentin tehtävänannot voivat sijaita EXAMissa itsessään, mutta myös ulkoisten palveluiden käyttäminen tenttitilanteessa on mahdollista. Ulkoista palvelua ei tosin toistaiseksi voida linkittää EXAMiin, eli A+ ei voi lähettää tälle tenttituloksia, eikä esimerkiksi avata tehtävää näkyviin vain silloin, kun tenttiminen on käynnissä. Tenttimisen integrointi A+:aan täten sivuutetaan tässä työssä ja jätetään jatkokehityskohteeksi.

4.11 Lopullinen arkkitehtuuri

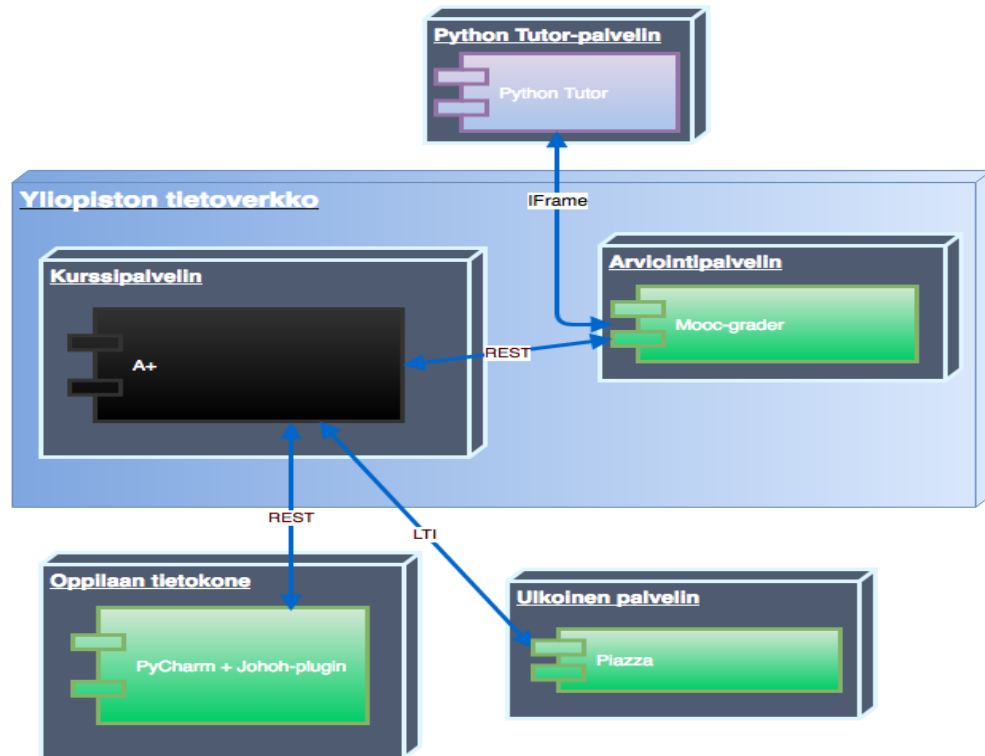
Luvussa 3 määriteltiin, miten komponentit voidaan liittää toisiinsa. Kun sijoitamme valitsemamme tuotteet arkkitehtuuriin ja rajaamme ylimääräiset komponentit pois, saamme kuvan 4.10 mukaisen lopputuloksen. Kerrataan kuitenkin vielä lyhyesti kuvan 4.9 avulla, mitä kukin käyttäjä tekee kullakin komponentilla.

Oppilas pystyy A+:ssa lukemaan oppimateriaalia, katsomaan videoita ja ohjelmavisualisointeja, tarkastelemaan tehtävänantoja ja etenemistänsä sekä siirtyä Piazzaan, jossa hän voi lukea ja kirjoittaa kysymyksiä ja vastauksia kurssin asioihin liittyen. PyCharm:illa hän taas pystyy ohjelmoimaan, testaamaan ja ajamaan tehtäviä. IDE:en tehtävä plugin hoitaa automaattisen palautusten teon A+:lle, sekä tehtävissä mahdollisesti tarvittavien ohjelmakoodipohjien haun ja projektien automaattisen luonnin.

A+:ssa näkyvät tehtävisivut sijaitsevat oikeasti Mooc-graderillä, mutta A+ hakee niiden lähdekoodit toimien eräänlaisena välityspalvelimena oppilaan ja graderin välillä. Tehtävisivut ovat tavallisia HTML-sivuja, joissa on samalla mahdollisesti opetusmateriaalia ennen tehtävien tekoa. Ohjelmavisualisoinnit, jotka sisällytetään materiaalisivuihin, tehdään käyttäen Python tutoria ja Jsvee:tä. Opetusvideot saadaan liitettyä sivuille yksinkertaisesti käyttämällä HTML5:n, eli HTML-kielen 5. version, video-elementtiä. Videon formaatti pitää tällöin olla joko mp4 tai webm [34].



Kuva 4.9: Järjestelmän arkkitehtuuri ja käyttäjät



Kuva 4.10: Järjestelmän arkkitehtuuri 4+1 -mallia mukaillen

Kun tehtävä on palautettu A+:aan, se lähtee Mooc-graderiin arvioitavaksi. Mooc-grader ajaa palautetun ohjelmakoodin suojatussa ympäristössä tehtävätyypin mukaisella tarkistusskriptillä. Lopuksi grader palauttaa A+:lle testitulokset. A+ tallentaa kaikki opiskelijan palautukset ja testitulokset, joten erillistä palautusarkistoa ei tarvita.

Opettaja pystyy tekemään tehtävänannot ja testiskriptit paikallisesti omalla tietokoneellaan ja tallentamaan ne esimerkiksi Git-tietovarastoon. Sieltä ne saadaan helposti siirrettyä Mooc-grader -palvelimelle ajamalla siellä olevassa git-instanssissa pull-komento. A+:ssa opettaja taas pystyy hallinnoimaan ja tarkastelemaan opiskelijoiden tietoja.

Assistenttien tulee joskus manuaalisesti antaa oppilaille pisteitä, ajaa testiajo uudestaan tai antaa tehtävään lisää aikaa. Nämä asiat ja harjoitustöiden manuaalisen arvioinnin hän voi tehdä A+:ssa.

5. JÄRJESTELMÄN TOTEUTUS

Tässä luvussa käsittelemme alaluvussa 4.11 esitellyn järjestelmän käytännön toteutusta. Pystymme tekemään määrittelemämme arkkitehtuurin mukaisen järjestelmän pitkälti valmiista komponenteista, mutta implementoitavaksi jää silti seuraavat asiat:

- A+:aan REST-rajapinta, jonka kautta plugin ja muut komponentit voivat kommunikoida A+:n kanssa (luku 5.1).
- PyCharm-plugin, jonka avulla PyCharm saadaan kommunikoimaan A+:n kanssa (luku 5.2).
- Mooc-graderin hiekkalaatikossa ajettavat testiskriptit, joilla oppilaan ohjelma testataan (luku 5.3).

Näiden osien toteutusta käsittelemme tarkemmin tulevissa aliluvuissa listassa kerrotussa järjestyksessä.

5.1 A+:n REST-rajapinta

A+:ssa valmiina oleva REST-rajapinta on tehty tekniikoilla, joita ei enää aktiivisesti kehitetä, joten teemme alusta alkaen uuden rajapinnan käyttäen kirjastoa nimeltä Django REST Framework (DRF) [10]. Kyseinen kirjasto valittiin, koska se on suosituin Django:n kanssa käytetty REST-kirjasto. DRF:n ominaisuuksiin kuuluu mm. automaattisesti luotava, web-selaimella selattava versio API:sta ja monia käyttäjän tunnistamiseen käytettäviä menetelmiä.

5.1.1 Autentikaatio ja auktorisointi

Käyttäjän tunnistamisella, eli autentikaatiolla tarkoitetaan prosessia, jossa varmistetaan, että käyttäjä on todellisuudessa se, joka väittää olevansa [35, s. 16]. Tunnistamisen voi tehdä monella tavalla, mutta tässä työssä käytämme tunnistustapana istuntopohjaista tunnistamista (session based authentication) ja API-avainperusteista tunnistamista (API-key based authentication), joista Django REST Framework tukee kumpaakin.

Istuntotunnistusta käytetään silloin, kun käyttäjä tarkastelee rajapintaa web-selaimen kautta. DRF:n istuntotunnistus käyttää hyväkseen Django:n käyttäjälle

tunnistamistilanteessa (selitettiin lyhyesti luvussa 4.1) luotavaa istuntoa. Käyttäjän istunto on niin kauan voimassa, kunnes tämä kirjautuu ulos tai poistaa evästeensä. API-avainta hyödyntävää tunnistusta käytetään, kun rajapintaa käyttää itsenäinen sovellus, meidän tapauksessamme IDE. API-avain -tunnistus toimii siten, että Django REST muodostaa jokaiselle käyttäjälle yksilöllisen merkkijonotunnisteen eli API-avaimen, jonka käyttäjän itsenäinen ohjelma lähettää rajapinnalle jokaisen tunnistautumista vaativan kyselyn mukana [11].

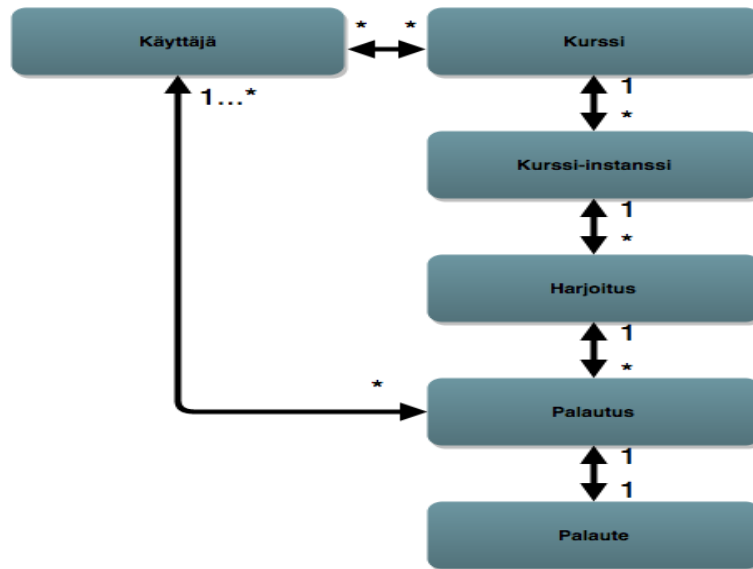
Autentikaatioon läheisesti liittyvä termi auktorisointi viittaa sääntöihin, jotka määräävät, ketkä käyttäjät pääsevät mihinkin resursseihin käsiksi ja mitä he voivat resursseilla tehdä. [35, s. 31] Esimerkiksi opettaja näkee kaikkien oppilaiden arvosanat ja voi muokata niitä, mutta oppilas näkee vain omansa, eikä voi muokata sitä. Määrittelemme säännöt luvussa 5.1.2, kunhan olemme ensin määritelleet itse resurssit.

5.1.2 Rajapintamäärittely

Luvussa 2.6.1 kerrottiin kuinka IDE:n pitää pystyä hakemaan A+:-sta kurssin tehtäväpohjat, sekä kurssirakenne, jotta tehtäväpohjat saadaan oikeisiin projektikansioihin. Lisäksi, oppilaan täytyy pystyä tekemään palautuksia A+:-aan IDE:n kautta. Rajapinnassa täytyy siis olla resursseina kurssi, harjoitus ja palautus.

A+ lähettää tehtävät arvioitavaksi Mooc-graderille, joka palauttaa arviointitulokset rajapinnan kautta takaisin A+:-lle, joten rajapinnassa täytyy olla resurssi palautteelle. Palautukset tulee linkittää oikeille käyttäjille, joten lisäämme rajapintaan vielä käyttäjä-resurssin.

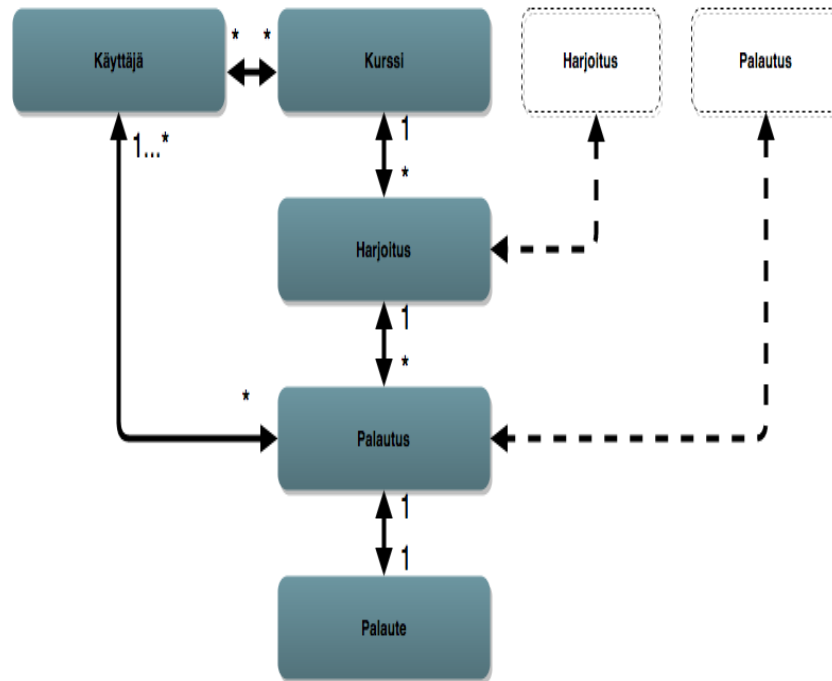
Resurssit muodostavat luonnollisen hierarkian (kuva 5.1): kurssilla on monta toteutuskertaa, joten sillä on monta instanssia, instanssilla on harjoituksia, harjoituksella on palautuksia ja palautuksella palaute. Lisäksi kurssit ja palautukset liittyvät käyttäjiin. Kuvassa 5.1 resurssien väliset lukumäärät on kuvattu UML-mallinnuksella, jossa asteriskilla tarkoitetaan lukumäärää, joka on nolla tai suurempi.



Kuva 5.1: REST-rajapinnan resurssit hierarkkisesti kuvattuna

REST-rajapinnoissa resurssien selaaminen URI:en kautta tehdään mahdolliseksi asettamalla samantyyppiset resurssit joukoiksi, joista voidaan yksittäisen resurssin kautta päästä siihen liittyviin resursseihin. Esimerkiksi joukko kurssi-resursseja on URI:ssa `/kurssit`, jonka sisältä voidaan valita jokin kurssi menemällä esimerkiksi URL:iin `/kurssit/1/`, jossa 1 on kurssin tunnustenumero. Tämän kurssin kurssi-instanssijoukkoon taas päästään URL:ista `/kurssit/1/instanssit/` ja täältä voidaan taas valita haluttu instanssi, esimerkiksi `/kurssit/1/instanssit/2`.

Ongelmana syvälle menevässä hierarkkisuudessa on, että resurssien muodostamat URI:t voivat olla hyvinkin pitkiä, esim. `api/v2/courses/johoh/instances/kk2016/exercises/1/submissions/1/feedback`. Toinen vaihtoehto olisi luoda tasainen (flat) rajapinta edellä esitellyn sisäkkäisen (nested) sijaan. Tasaisessa rajapinnassa resurssit tosin ovat yhdessä tasossa, eikä sisäkkäisyyttä esiinny, mikä on vastoin resurssien luonnollista hierarkkisuutta. Ratkaisemme sisäkkäisen rajapinnan ongelman jättämällä instanssi-resurssin pois ja lisäämällä oikopolut harjoituksiin ja palautuksiin kuvan 5.2 mukaisesti. Instanssi-resurssia ei tarvita, koska jokaisella tehtävällä, palautuksella ja palautteella on oma yksilöllinen tunnisteensa. Resurssit eivät siis sekoitu keskenään, vaikka järjestelmässä olisi monta saman kurssin toteutuskertaa.



Kuva 5.2: REST-rajapinnan resurssit korjattuna

Kuvassa 5.2 esitellystä resurssihierarkiasta voidaan johtaa URI:t resursseille. Rajapinnan URI:n juuressa on kuvan mukaisesti käyttäjä, kurssi, harjoitus ja palautus. Kurssi-resurssin kautta päästään hierarkiassa eteenpäin harjoituksiin, palautuksiin ja palautteeseen. Johdetut URI:t, niiden sallimat metodit ja käyttöoikeudet on listattu taulukossa 5.3. Submitter_stats on palvelimen luoma resurssi, josta nähdään tilastoja yksittäisen käyttäjän tiettyyn tehtävään tekemistä palautuksista. URI:sta /submission/<id>/feedback/ automaattinen arviointipalvelin voi asettaa palautuksille arvosanoja sekä hakea arviointia varten lisätietoa palautuksesta ja tehtävästä.

Resurssi	URI	Sallitut metodit ja oikeudet
API:n juuripolku	/api/v2/	GET (kaikilla kirjautuneilla käyttäjillä)
Lista käyttäjistä	users/	GET (opettajilla)
Yksittäisen käyttäjän tiedot	users/<user_id>/	GET (opettajilla kaikkiin, muilla omiin tietoihinsa)
Lista kurseista	courses/	GET (kaikilla kirjautuneilla käyttäjillä)
Yksittäisen kurssin tiedot	courses/<course_id>/	GET (kaikilla kirjautuneilla käyttäjillä)
Lista kurssin tehtävistä	courses/<course_id>/exercises/	GET (kaikilla kirjautuneilla käyttäjillä)
Kurssille liittyneet opiskelijat	courses/<course_id>/students/	GET (opettajilla)
Yksittäisen tehtävän tiedot	exercises/<exercise_id>/	GET (kaikilla kirjautuneilla käyttäjillä)
Tehtävään tehdyt palautukset	exercises/<exercise_id>/submissions/	GET, POST (opettajilla kaikkiin, muilla omiin tietoihinsa)
Oppilaan palautustilastot tehtävässä	exercises/<exercise_id>/submitter_stats/<user_id>/	GET (opettajilla kaikkiin, muilla omiin tietoihinsa)
Palaute palautukselle	submissions/<submission_id>/feedback/	GET (opettajilla, arviointipalvelimella), POST (arviointipalvelimella)
Palautuksen tiedot	submissions/<submission_id>/	GET (opettajilla kaikkiin, muilla omiin tietoihinsa)

Kuva 5.3: A+ :n REST API:n resurssit

Rajapinnassa käytetään ainoastaan GET ja POST -metodeja, GET:iä tiedon hakemiseen ja POST:ia uuden resurssin luomiseen. Kaikkiin kuvassa 5.3 esitettyihin

resursseihin voidaan lähettää GET-pyyntö. Automaattisen arviointipalvelimen tekemät arvioinnit ja palautukset tehdään POST-pyynnöllä. Oppilaan asiakasohjelma voi tehdä palautuksen lähettämällä tekemänsä tehtävän POST-pyynnöllä resurssiin `exercises/<exercise_id>/submissions/`. Rajapinnan kautta ei voi poistaa (DELETE) tai muokata (PATCH) mitään, koska millekään resurssille ei ole tarvetta tehdä näitä operaatioita, ainakaan rajapinnan kautta.

Käyttöoikeudet eli auktorisointi on yleisellä tasolla määritelty siten, että opettajilla on pääsy kaikkiin resursseihin käyttäen kaikkia metodeita, kun taas oppilailla on pääsy vain omiin tietoihinsa ja POST-oikeus ainoastaan palautuksen tekoon. Palautte on erikoistapaus, koska siihen voi ainoastaan Mooc-grader tehdä POST-pyynnön.

Kaikkiin URI:in, joista voidaan hakea yksittäisen opiskelijan tietoa, on myös mahdollisuus antaa käyttäjäID:n sijaan tunniste "me". Tällöin A+ tunnistaa kysyjän tämän lähettämän viestin otsakkeessa nimeltä "Authorization" olevan API-avaimen avulla. Esimerkiksi kun käyttäjä lähettää pyynnön URI:in `/api/v2/users/me`, saadaan listauksen 5.1 mukainen vastaus. Listauksesta nähdään, että vastaus sisältää opiskelijan tunnistenumeron, käyttäjänimen, listan kursseista, joihin hän on ilmoittautunut, opiskelijanumeron, nimen ja sähköpostiosoitteen.

Listaus 5.1: Opiskelijan tiedot vastausviestissä

```
{
  "url": "http://127.0.0.1/api/v2/users/me/",
  "user_id": 2,
  "username": "testihlo",
  "courses": [
    {
      "name": "Johdatus_Ohjelmointiin_KK2016",
      "id": 1
    }
  ],
  "student_id": "229656",
  "first_name": "Teemu",
  "last_name": "Testaaja",
  "email": "teemu@testaaja.com"
}
```

API:a voidaan selata resurssista toiseen tietämättä API:n rakenteesta mitään. Tämän mahdollistavat jokaisesta resurssista löytyvät linkit, jotka johtavat resurssiin liittyviin resursseihin. Esimerkiksi kurssi-resurssissa on linkit kurssiin liittyviin harjoitukset- ja opiskelijat -resursseihin, kuten listauksessa 5.2 on esitetty.

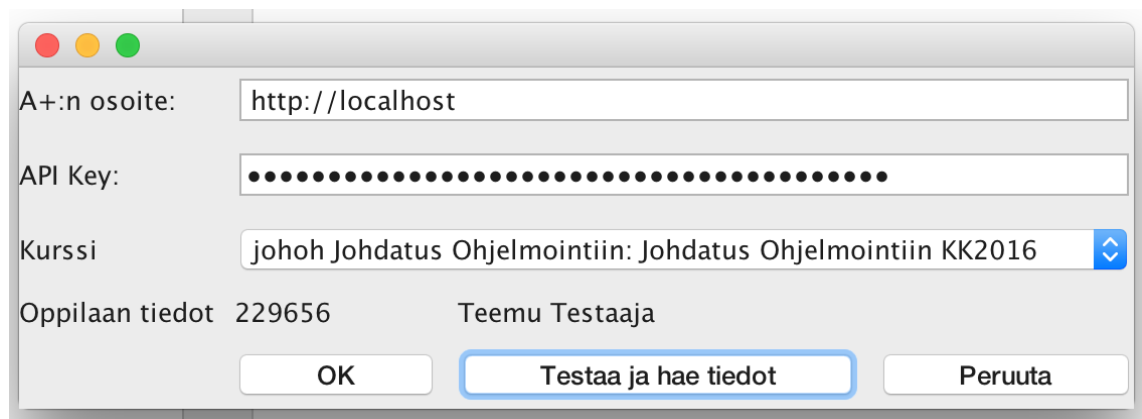
Lista 5.2: Kurssi-resurssin tiedot JSON-muodossa

```
{
  "id": 5,
  "url": "https://plus.cs.tut.fi/api/v2/courses/5/",
  "html_url": "https://plus.cs.tut.fi/johoh/S2016/",
  "code": "TIE-02100",
  "name": "Johdatus_ohjelmointiin",
  "instance_name": "Fall_2016",
  "language": "fi",
  "starting_time": "2016-08-28T21:00:00Z",
  "ending_time": "2016-12-21T22:00:00Z",
  "visible_to_students": true,
  "exercises": "https://plus.cs.tut.fi/api/v2/courses/5/exercises/",
  "students": "https://plus.cs.tut.fi/api/v2/courses/5/students/"
}
```

5.2 PyCharm-pluginin

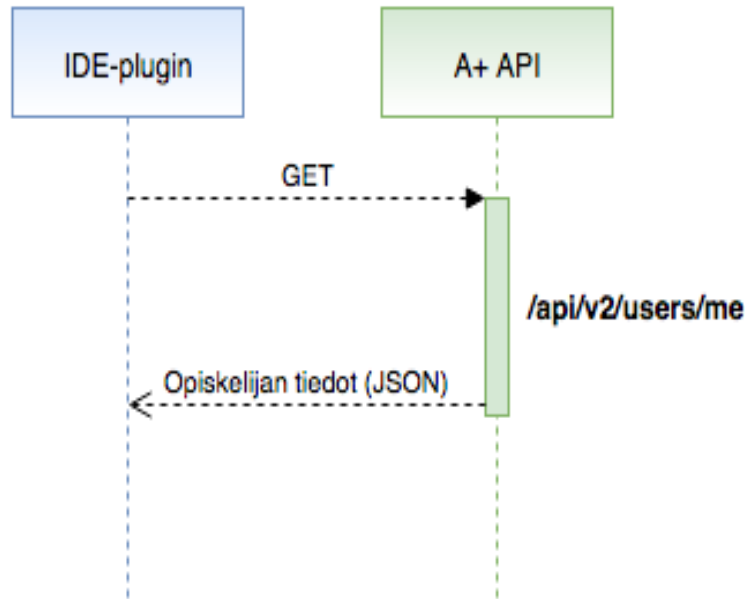
Pluginin tarkoitus on helpottaa ohjelmakoodipohjien hakemista sekä projektien luomista ja tehtävien palauttamista. Tutustumme näihin käyttötapauksiin kahdessa seuraavassa aluvuossa, mutta ensin käsittelemme pluginin käyttöönottoa.

Ensimmäisellä käyttökerralla oppilas antaa asetuskunassa (kuva 5.4) ohjelmalle A+:n luoman API-avaimen, jonka oppilas saa A+:ssa olevasta omasta tilinäkymästään. Tällä avaimella käyttäjä tunnistetaan ja palautukset menevät oikean käyttäjän nimissä. Lisäksi oppilas joutuu antamaan kurssilla käytettävän A+ -palvelimen osoitteen, koska tämäkin voi vaihdella kurssin toteutuskerran ja -paikan mukaan.



Kuva 5.4: PyCharm-pluginin asetuskunassa

Pluginin asetusikkunassa on nappula ”Testaa ja hae tiedot”, jota painamalla ohjelma lähettää annetulle serverille polkuun `/api/v2/users/me` GET-pyyynnön (kuva 5.5), jonka otsake sisältää API-avaimen. Serveri osaa avaimen avulla hakea sitä vastaavan käyttäjän nimen, opiskelijanumeron ja kurssit, joihin tämä on ilmoittautunut. Serveri antaa HTTP-vastauksessa nämä pluginille, joka näyttää tiedot käyttäjälle asetusikkunassa. Vastausviesti on listattu edellisessä kappaleessa listauksessa 5.1. Käyttäjä voi valita alasvetovalikosta kurssin, jonka tehtäviä aikoo hakea ja palauttaa.



Kuva 5.5: Sekvenssidiagrammi oppilaan tietojen hausta

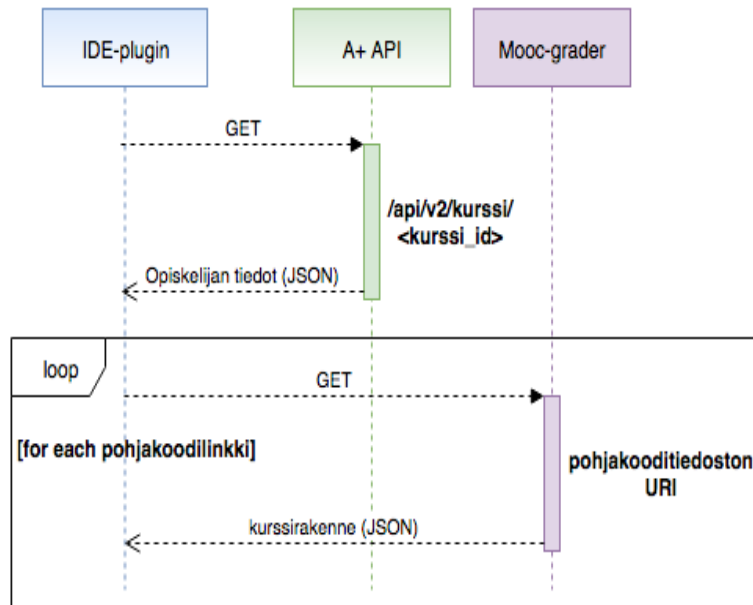
5.2.1 Kurssirakenteen haku

Valmiit ohjelmakooditiedostot oppilas voi hakea painamalla ”Hae tehtävät”-nappulaa. Tällöin plugin pyytää kurssin rakennetta ja tehtäväpohjia A+:lta, jolloin tämä lähettää pyytäjälle kurssin rakenteen JSON-tiedostona (yksinkertaistettu listaus 5.3), jonka sisällä on myös linkit ohjelmakoodipohjiin ja lähettämishetken aikaleima. PyCharm jäsentää saadun JSON:in ja tekee saatujen tietojen perusteella projektikansiot, tehtäväkansiot ja jokaiseen tehtäväkansioon asetustiedoston, joka sisältää tiedon mihin osoitteeseen tehtävä tulee palauttaa, sekä aikaleiman, joka kertoo, koska ohjelmakooditiedoston haku tehtiin. Lopuksi plugin hakee ohjelmakoodipohjat tehtäväkansioihin JSON:ista saatujen linkkien avulla.

Listaus 5.3: Kurssin JSON:in muoto

```
{
  "aikaleima": "2016-09-09T04:00:18Z",
  "tehtavat": [
    "viikko_x": [
      {
        "tehtavanimi": "moro",
        "ohjelmakoodipohja": "None",
        "palautusosoite": "http://.."
      },
      {
        "tehtavanimi": "hymiot",
        "ohjelmakoodipohja": "http://..",
        "palautusosoite": "http://.."
      }
    ],
    "viikko_x+1": [
      ...
    ],
  ]
}
```

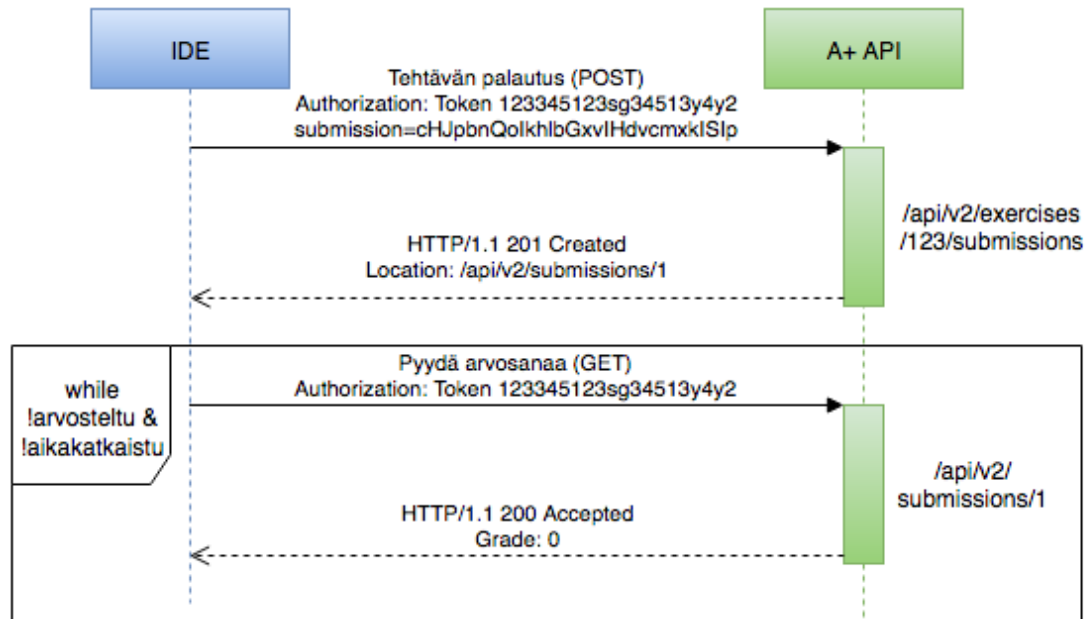
On mahdollista, että kesken kurssin opettaja muokkaa ohjelmakoodipohjia mahdollisen virheen tai parannusehdotuksen takia, jolloin oppilas joutuu hakemaan tehtäväpohjat uudestaan. Aikaleimasta nähdään, koska edelliset pohjat on haettu, jolloin tiedettäessä viimeisin ohjelmakoodipohjan muokkausajankohta voidaan päättää, korvataanko vanhat oppilaan koneella olevat tiedostot. Esimerkiksi, jos oppilas on hakenut ohjelmakooditiedostot 5.10 ja viimeisin muokkaus serverillä oleviin pohjakoodeihin on tehty 12.10, on oppilaan versio vanhentunut. Tällöin kysytään oppilaalta, ylikirjoitetaanko vanha ohjelmakooditiedosto vai ei, jos hän on mahdollisesti jo tehnyt muokkauksia ohjelmakoodiin.



Kuva 5.6: Sekvenssidiagrammi kurssirakenteen hausta

5.2.2 Tehtävän palautus

Kun käyttäjä on toteuttanut ohjelmakoodinsa valmiiksi, hän voi palauttaa parhailaan aktiivisena olevan tiedoston. Palautus tapahtuu painamalla ohjelman yläpalkin ”Johoh”-valikosta ”Palauta tehtävä”-nappulaa. Teknisesti palautus toimii siten, että plugin tekee POST-pyyntön A+:lle polkuun `/api/v2/exercises/<exercise_id>/submissions`. Pyyntö sisältää ohjelmakooditiedoston, sekä mahdolliset lisätiedostot mukana hyötykuormassa. Ohjelmakooditiedosto tulee koodata base64-koodauksella, koska se voi sisältää mitä tahansa merkkejä, jotka voivat sekoittaa lähetettävän JSON-tiedoston muodon. Base64-koodaus muuttaa koodattavan datan merkkijonoksi, jonka merkit ovat JSON-rakenteen kannalta turvallisia ASCII-merkkejä [56]. Palautuksen saatuaan A+ tarkistaa, voidaanko palautus tehdä ja onko käyttäjää olemassa. Jos kaikki on kunnossa, palvelin vastaa asiakkaalle ”created 201”-viestillä, jonka location -otsakkeessa on linkki palautusta vastaavaan resurssiin. Tästä osoitteesta asiakasohjelma pyytää määritellyn ajan välein tarkastuksen tulosta, kunnes saa sen tai aikakatkaisu tapahtuu. Määrittelemme pyyntöjen aikavälien kasvavan eksponentiaalisesti 2:n potenssissa, eli ensimmäinen pyyntö tapahtuu 1 sekunnin päästä, seuraava kahden, sitä seuraava neljän ja niin edelleen. Arvostelussa voi ruuhkaisena aikana mennä jopa 2 minuuttia, joten aikakatkaisu määritetään tapahtumaan 7. pyynnön, eli noin 127 sekunnin jälkeen. Esimerkki tästä osapuolten välisestä viestinnästä on esitetty kuvassa 5.7.

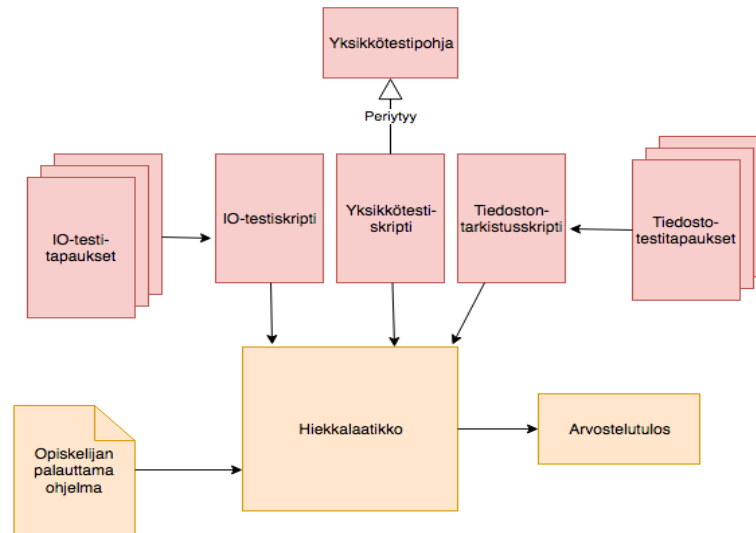


Kuva 5.7: IDE:n ja API:n kommunikointi tehtävän palautuksessa

5.3 Mooc-grader:in tarkistusskriptit

Mooc-grader tarjoaa hiekkalaatikon, jossa ohjelmia voi suorittaa turvallisesti. Lisäksi se osaa ottaa palautukset vastaan, laittaa ne tarkastusjonoon ruuhkan mukaan ja palauttaa HTML-muotoisen tuloksen opiskelijalle A+:aan. Grader ei kuitenkaan ota kantaa siihen, miten palautetut ohjelmakoodit testataan ja mitä palautteeksi annetaan, joten tehtäväksemme jää tehdä skriptit, jotka ajetaan hiekkalaatikossa oppilaan palautuksen kanssa. Vaikka hiekkalaatikko estää suurimmat tietoturvaongelmat ja Grader tekee aikakatkaistun testauksen mennessä pieleen, sen A+:aan antama virheilmoitus ei ole opiskelijalle ymmärrettävää. Testiskriptien täytyy siis tehdä oppilaan ohjelmakoodille tarkistuksia ja sanitointioperaatioita sekä antaa oppilaalle järkevää palautetta mahdollisissa virhetilanteista.

Arviointiprosessi (kuva 5.8) lähtee liikkeelle siten, että Grader ottaa tarkastusjonosta arvioitavan palautustiedoston ja kopioi sen hiekkalaatikkoon valittujen tarkistusskriptien kanssa. Tarkistusskriptejä on kolmenlaisia: IO-testiskripti, yksikkötestiskriptit ja tiedoston tarkistusskripti. IO-testin toimintaperiaate on, että testattavalle ohjelmalle annetaan tietty syötejoukko ja sen niillä antamaa ulostuloa verrataan oikeiksi tiedettyihin ulostuloihin. Yksikkötestit taas toimivat siten, että funktiolle tai metodille annetaan mahdolliset parametrit, se ajetaan ja lopuksi vertaillaan funktion paluuarvoja odotettuihin. Tiedoston tarkistusskripti tarkistaa, että oppilaan ohjelmakooditiedosto lukee ja kirjoittaa tiedostoja oikein. Yhdessä testiajossa voidaan käyttää vaikkapa kaikkia skriptityyppejä.



Kuva 5.8: Oppilaan palautuksen tarkistusprosessi

Ennen testien ajamista yksikkötestiskripti tekee oppilaan ohjelmakoodille tarkistuksen siitä, löytyykö testattava metodi tai funktio, ja poistaa main-funktion kutsun. Tämä täytyy tehdä siksi, että oppilaan koko ohjelmakooditiedostoa ei ajeta. IO-testiskripti sisältää ajonaikaisen varmistuksen siitä, että testattava ohjelma odottaa syötteitä oikeissa tilanteissa. Tällä varmistetaan se, että oppilaan ohjelma ei jää odottamaan aikakatkaisuun asti olematonta syötettä. Kummassakin testiskriptissä on myös aikakatkaisu sen varalta, että testattava ohjelma jää vaikkapa ikuisen silmukkaan.

Testiskriptit ajavat yleensä useamman testitapauksen, joiden onnistumisten mukaan oppilaalle annetaan pisteet ja palaute. Jokaisesta onnistuneesta testitapauksesta voidaan esimerkiksi antaa yksi piste ja lopussa pisteet lasketaan yhteen. Testitapauksella tarkoitetaan ohjelman tai funktion yksittäistä suorituskertaa tietyllä syöte- tai parametrijoukolla. Esimerkiksi funktiota, joka laskee sille parametrina annetun kokonaisluvun neliöjuuren, voidaan testata erikseen vaikkapa positiivisella ja negatiivisella parametrilla, jolloin saadaan kaksi testitapausta. IO- ja tiedoston tarkistusskriptit käyttävät erillisiä testitiedostoja testauksessa, kun taas yksikkötestiskriptit sisältävät testitapaukset, eli jokaista testitapausta varten täytyy tehdä uusi skripti. Tätä työtä tosin helpottaa Unittest-kirjastolla [60] tehty yksikkötestipohja, josta jokainen yksikkötesti periytetään.

Kun testit on ajettu ja pisteet laskettu, skripti tulostaa palautteen ja pisteet. Mooc-grader ottaa tämän tulostuksen vastaan ja muodostaa siitä HTML-sivun, joka palautetaan A+:-aan. Kuvassa 5.9 on kuvankaappaus testiskriptien ohjelmalle tuottamasta palautenäkymästä, joka on mennyt läpi yksikkötesteistä, mutta ei tuota oikeaa tulostetta. Vasemmalla näytetään, mitä ohjelma testattaessa tulosti ja oikealla se mitä sen odotettiin tulostavan. Alhaalla luetellaan yksikkötestien tulokset.

Jos yksikkötesti ei mennyt läpi, ohjelma kertoo oppilaalle testatun funktion nimen ja sille annetut parametrit.

Kokonaispisteet 2/3

Test Failed: Tehtävänannon esimerkki

Oma ohjelmasi

Malliohjelma

Syötä viestin tekstirivejä. Lopeta
Tässä tulee hirmuinen Rölli,
on mulla metsässä tölli.
Ja siellä mä asustelen,
ja sieltä mä hiippailen.

ROT13:
Gäffä ghyrr uvezhvara Eöyyv,
ba zhyyn zrgfäffä göyyv.
Wn fvryyä zä nfhfgryra,
wn fvrygä zä uvvccnvyra.

Pisteet 2/2

```
test_rivin_salauus (unittester.TestaaSalaa) ... ok
test_salaa (unittester.TestaaSalaa) ... ok
```

Ran 2 tests in 0.001s

OK

Kuva 5.9: Automaattisen skriptin antama palaute

5.4 Yhteenveto

Tässä luvussa suunnittelimme miten A+_n:n REST-rajapinnan, PyCharm-pluginin ja Mooc-graderin tarkistusskriptit voitaisiin toteuttaa. Rajapinnan resurssiksi valittiin käyttäjä, kurssi, harjoitus, palautus ja palaute. Näissä voidaan käyttää GET- ja POST-metodeja, GET:iä tiedon hakemiseen ja POST:ia palautusten ja palautteiden tekoon. Kurssivastuuhenkilöillä on oikeus nähdä kaikki resurssit, kun taas oppilailla vain omat tietonsa, palautuksensa ja tehtävänannot.

PyCharm-plugin osaa hakea kurssirakenteen ja tehtäväpohjat sekä palauttaa tehtävän A+_n:aan. Pluginin käyttäjän tunnistaminen tehdään yksilöllisellä API-avaimella, jonka A+ muodostaa jokaiselle käyttäjälle.

Mooc-graderin tarkistusskriptejä tehtiin kolmenlaisia: IO-, ja yksikkötestiskripti sekä tiedoston tarkistusskripti. Kyseiset skriptit ajetaan testattavan tiedoston kanssa hiekkalaatikossa ja testien suoriuduttua skriptit muodostavat palautteen ja arvostelun oppilaalle.

6. TOTEUTUKSEN ARVIOINTI

Luvussa 1.4 esiteltiin tutkimuskysymys: ”Miten pystymme rakentamaan yhtenäisen ohjelmoinnin oppimisympäristön? Mitä osia sen pitää sisältää ja miten näiden tulisi kommunikoida keskenään?”. Arvioimme tässä luvussa siis seuraavia asioita:

1. Onko lopputulos yhtenäinen? (luku 6.1)
2. Oliko potentiaalisten osien etsintä kattavaa? (luku 6.2)
3. Oliko valittu kommunikointitapa hyvä? (luku 6.3)

6.1 Yhtenäisyys

Yhtenäisellä ympäristöllä tarkoitetaan sitä, että se näyttäytyy opiskelijoille yksittäisenä ohjelmana, johon ei tarvitse kirjautua erikseen eri toimintoja varten. Oppimisympäristön toteutus onnistui tässä siten, että alkurekisteröintiä lukuunottamatta käyttäjän tulee kirjautua järjestelmään vain yhdestä paikasta eli A+:sta. Alkurekisteröinti täytyy tehdä Piazzassa ja PyCharm-pluginissa. Piazzassa oppilaan täytyy määrätä käyttäjätunnukseensa salasana ja pluginissa antaa tälle A+:n API-avain. Alkurekisteröintiä ei tarvitse toistaa enää myöhemmin, paitsi jos oppilas asentaa PyCharmin uudestaan tai esimerkiksi vaihtaa tietokonetta. Täten vaatimukseen yksittäisestä kirjautumisesta vastattiin.

Yhtenäisyyteen liittyvä erityistapaus oppimisympäristössämme on IDE:n ja LMS:n välinen integraatio. Tehtävien tekemisestä ja palauttamisesta haluttiin tehdä mahdollisimman mutkatonta. Pluginilla teimme mahdolliseksi automaattisen kansiorakenteiden teon ja ohjelmakoodipohjien haun palvelimelta sekä tehtävän palauttamisen ja siitä arvosanan saamisen kirjautumatta A+:aan. Tämä helpottaa opiskelijoiden oppimisprosessia huomattavasti edelliseen järjestelmään nähden, joten yhtenäisyyden vaatimus katsotaan saavutetuksi. Järjestelmää voisi tosin mahdollisesti vielä parantaa siten, että opiskelija saisi myös tehtävänannot PyCharmiin.

6.2 Osien etsintä

Vaatimusmäärittely aloitettiin tutkimalla vanhaa järjestelmää ja sen ominaisuuksia. Lisäksi kurssihenkilökunnan kesken epäformaalisti käytiin läpi erilaisia lisätoimintoja, joita järjestelmän pitäisi tarjota. Uudessa järjestelmässämme on kaikki ne

ominaisuudet, jotka vanha järjestelmä tarjosi, eli esimerkiksi automaattinen ja manuaalinen arviointi. Lisäksi järjestelmään saatiin uusiksi ominaisuuksiksi ohjelmavisualisointi ja IDE:n integraatio LMS:n kanssa.

Järjestelmä ei ole lukittu käytettäväksi vain valitsemillamme komponenteilla, vaan niitä voidaan vaihtaa toisiin. Myös täysin uusien komponenttien käyttöönotto on mahdollista. Näin ollen, kun jossain vaiheessa valitsemiamme komponentteja ei enää tueta ja kehitetä, voidaan ne kohtuullisella työmäärällä vaihtaa uusiin.

Saavutettujen ominaisuuksien ansiosta voimme pitää komponenttien hankintaa implementoidun järjestelmän osalta onnistuneena. Kaikkia alussa listattuja vaatimuksia emme kuitenkaan saaneet ratkaistua, koska emme löytäneet sopivaa vertaisarviointikomponenttia.

Lisäksi oppimisenhallintajärjestelmän valinta herättää kysymyksiä. Alkuvaatimuksena oli, että järjestelmässä käytetään A+ -oppimisenhallintajärjestelmää. Luvussa 4.1 esiteltiin Sakai-niminen LMS, joka sisältää valmiiksi REST-rajapinnan, sekä tuen LTI 2.0-protokollalle, joten tämä olisi voinut olla parempi valinta LMS:äksi.

6.3 Kommunikointitapa

Työssä ratkaistiin kommunikointikysymys tekemällä uusi REST-rajapinta A+:aan, koska tämä valikoitui keskuskomponentiksi, jonka kanssa muut komponentit kommunikoiivat. Vaihtoehtona itse tehdylle REST-rajapinnalle olisi ollut käyttää LTI 2.0-protokollaa. Tästä olisi ollut hyötyä siinä, että A+:aan olisi helpompi liittää uusia, tätä protokollaa noudattavia komponentteja. Nykyisessä tapauksessa komponenttia joutuu todennäköisesti muokkaamaan, että kommunikointi sen ja rajapinnan välillä olisi mahdollista. Toisaalta LTI 2.0 on raskas protokolla, joten sen implementointi olisi todennäköisesti ollut haastavampaa, ja rajapintaan olisi jouduttu implementoimaan sellaisia osia, joita ei koskaan tarvittaisi.

Tutkitaan vielä rajapintatoteutuksemme onnistumista. Käytämme tässä hyväksi Richardsonin kypsyysmallia [13] ja Microsoftin REST-rajapintaohjeistusta [39].

6.3.1 Richardsonin kypsyysmalli

Richardsonin kypsyysmalli jakaa REST-mallin tärkeimmät vaatimukset kolmeen osaan: resurssit, HTTP-metodit ja hypermedialla ohjattavuus. Kaikki kolme osaa on selitetty aiemmin kappaleessa 3.2.1. Se miten arvioitava rajapinta näihin vaatimuksiin vastaa, määrittelee sen kypsyystason. Richardsonin kypsyysmallissa on 4 tasoa lähtien tasosta 0. Tason 0 rajapinta tarjoaa asiakkaalle ainoastaan yhden päätepisteen, johon asiakas voi HTTP:n kautta tehdä etukäteen määriteltyjä toimintapyyntöjä. Tasossa 1 on otettu huomioon resurssirajoite, mutta ei ole käytetty HTTP-metodeja oikealla tavalla. Tasolla 2 metodeja on käytetty oikein, sekä

vastausviestien koodit ovat loogiset. Esimerkiksi tapahtuman onnistuessa rajapinta antaa vastaukseksi koodin 200 (OK) ja vaikkapa oikeuksien puuttuessa koodin 401 (Unauthorized). Tällä tasolla ei kuitenkaan tueta hypermedialla ohjattavuutta. Tasolla 3 on otettu hypermediakin huomioon, jolloin rajapintaa voidaan pitää REST-määritelmän mukaisena [13].

Jokaisella tasolla on selkeä hyöty. Taso 1 ensinnäkin selkeyttää rajapinnan rakennetta, kun kaikkia ominaisuuksia ei ole pakattu yhteen pisteeseen. Taso 2 helpottaa rajapinnan asiakassovellusten kehittäjien työtä, koska HTTP-metodit ovat standardeja käskyjä ja täten tutumpia kehittäjille, kuin vaikkapa rajapinnan kehittäjän itse keksimät metodit. Taso 3 myös helpottaa asiakassovelluksen tekijän urakkaa. Kun rajapinta on selattava, asiakkaan ei tarvitse etukäteen tietää rajapinnan resurssien osoitteita, vaan se voi liikkua niihin linkkien kautta. Täten asiakasohjelmaan ei tarvitse kovakoodata rajapinnan resurssien osoitteita, eikä näin ollen tarvitse huolehtia niiden rikkoutumisesta mahdollisten muutosten yhteydessä.

Olemme jakaneet rajapintamme loogisiin resursseihin, joten tason 1 vaatimus täyttyy. HTTP-metodeja on käytetty taulukon 3.1 määrittelyjen mukaisesti, eli tietoa hakiessa on käytetty GET:iä ja uuden resurssin teossa POST:ia. Täten tason 2 vaatimus täyttyy. Rajapinnan eri resurssien välillä pystytään siirtymään ilman aiempaa tietoa rajapinnan rakenteesta käyttäen hyperlinkkejä, kuten luvussa 5.1.2 määriteltiin. Täten vaatimus hypermedialla ohjattavuudestakin täyttyy, joten rajapinta on REST-määritelmän mukainen. PyCharm-pluginin ei tosin hyödynnä hypermediaa, vaan siihen on kovakoodattu resurssien osoitteet.

6.3.2 Microsoftin REST-ohjeistus

Microsoft on luonut palveluidensa kehittäjille ohjeistuksen, jolla palvelujen REST-rajapintoja pyritään yhdenmukaistamaan. Ohjeistus noudattaa teollisuudessa laajalti hyväksytyjä REST:in parhaita käytäntöjä ja pyrkii tekemään rajapinnoista helppokäyttöisiä niiden asiakkaille. Nostamme ohjeistuksesta esille muutamia kohtia ja vertaamme niitä omaan rajapintatoteutukseemme. Näin voimme varmistaa rajapintamme laadun suunnittelun osalta.

Ohjeistuksessa kerrotaan asiakasohjelmien käyttäytymisestä, johdonmukaisen rajapinnan pääpiirteistä, resurssijoukoista, JSON:n standardimuotoilusta, rajapinnan versioinnista, paljon aikaa vievistä kyselyistä, webhookeista, ei-tuetuista pyynnöistä, CORS:stä (Cross Origin Resource Sharing) ja deltakyselyistä. Ohitamme arvioinnissamme asiakasohjelmien käyttäymisösion, JSON:in standardimuotoilun, CORS:n ja deltakyselyt. Asiakasohjelmien käyttäytymistä emme tutki, koska aiemmin rajasimme tarkastelun A+:n REST-rajapintaan. JSON:n standardimuotoilun tarkastelun ohitamme siksi, että Django REST Framework tekee datan sarjallistamisen JSON-muotoon automaattisesti. Ei-tuetut pyynnön ohitamme samasta syystä, jos raja-

pintaan tulee pyyntö, jota tämä ei tue, DRF osaa antaa asiallisen virheilmoituksen asiakkaalle. CORS:llä tarkoitetaan sitä, että internetsivulla oleva JavaScript-ohjelma voi tehdä pyyntöjä toisella toimialueella (eng. domain) sijaitsevalle palvelulle, esimerkiksi meidän rajapintaamme [59]. Deltakyselyillä (eng. delta query) voidaan haakea asiakkaan viimeksi tekemän kyselyn jälkeen resursseissa tapahtuneet muutokset [39] [58]. Ohitamme myös nämä ominaisuudet, koska niille ei lyödetty tarvetta rajapinnassamme.

Ohjeistuksen kohdassa 7 käsitellään rajapinnan yhdenmukaisuutta. Ensimmäisenä vaaditaan, että ihmisen täytyy pystyä lukemaan ja muodostamaan rajapinnan polkuja. Kohdassa 7.3 kerrotaan, että rajapinnan URI:ssa voidaan käyttää oikopolkuja, kuten `"/my"`, jolla päästään käyttäjän omaan resurssiin, kuten määrittelimme luvussa 5.1.2. Kohdassa 7.4 käsketään käyttämään HTTP-metodeja asianmukaisesti, eli esimerkiksi GET-metodilla haetaan resurssin arvo ja POST-metodilla tehdään uusi resurssi. Erityisesti POST-metodista mainitaan, että tämän tulee palauttaa asiakkaalle Location-otsakkeella luodun resurssin osoite. Kohdassa 7.9 kerrotaan, että tietoturvallisuuden takia käyttäjän henkilökohtaisia tietoja ei saa lähettää URI-polkujen seassa parametreina, vaan ne tulee lähettää otsakkeissa. Kaikkiin edellämäinnittuihin vaatimuksiin rajapinta vastaa.

Kohdassa 7.10 esitellään normaaleja HTTP-koodeja tarkempi virheilmoitusformaatti. Rajapinnan antama virheen ilmoittava vastaus sisältää vikakoodin, selkokielisen vikailmoituksen ja mahdollisesti tarkempia tietoja. Lisäksi vastauksen otsakkeessa on `"Retry-after"` -niminen otsake, joka kertoo kuinka monta sekuntia asiakkaan täytyy odottaa ennen seuraavan kyselyn tekoa. Meidän rajapinnassamme käytetään tavallisia HTTP-koodeja ilmaisemaan tapahtuneita virhetilanteita. Tämä voi aiheuttaa joissain tapauksissa epäselvyyksiä, mutta työmäärään nähden uuden virheilmoitusformaatin implementointi ei liene kannattavaa. `"Retry-after"` -otsake tosin toisi joustavuutta pluginien arvosanan hakemiseen, kun palvelin voi kuormituksensa mukaan käskää asiakkaita odottamaan tietyn ajan ennen seuraavaa kyselyä.

Kohdassa 9 käsitellään resurssijoukkoja. Jos JSON-tiedostossa on lista resursseista, ne tulee olla taulukkorakenteessa, jokaisella resurssilla tulee olla yksilöllinen tunniste ja resurssijoukkoja saa olla resurssijoukkojen sisällä (nested resources). Arvioitava rajapinta noudattaa näitä ohjeita ja sisältää sisäkkäisiä resursseja. Suurien resurssijoukkojen esitys tulee jakaa useampaan pienempään JSON-tiedostoon (pagination), joiden välillä pystytään liikkumaan hyperlinkkien avulla. Jaon voi päättää joko serveri itsenäisesti tai asiakas voi esittää pyynnön, kuinka monessa osassa se resurssilistan haluaa. Arvioitava rajapinta tekee jaon itsenäisesti, eikä kuuntele asiakkaan ehdotuksia. Serveri voi tukea resurssilistojen suodatusta ja järjestelyä, jolloin asiakas voi URL-parametreina antaa erilaisia suodatus- tai järjestyskäskyjä. Esimerkiksi pyyntö `"/courses/?id=5"` palauttaa vain kurssin, jonka yksilöllinen tunniste on

5. Arvioitava rajapinta ei tue suodatusta, eikä järjestelyä.

Kohdassa 12 käsitellään rajapinnan versiointia. Version valinnan voi tehdä rajapinnassa kahdella tapaa: osana URI, kuten arvioitavassa rajapinnassa eli `"/api/v2/..."`, tai URL-parametrina, eli esimerkiksi `"/api/courses?api-version=2.0"`. Versionumeroa tulee kasvattaa silloin, kun rajapinnassa tapahtuu suuria muutoksia. Tällaisia muutoksia ovat sellaiset, jotka muuttavat rajapintasopimuksia ja aiheuttavat yhteensopivuusongelmia aiempiin versioihin nähden, esimerkiksi resurssien uudelleennimeäminen ja vastauskoodien muutokset. Arvioitavassa rajapinnassa on kaksi versiota: 1.0 ja 2.0, joista 2.0 on tässä diplomityössä toteutettu versio. Vanhaan rajapintaan nähden suurin osa resursseista ja toiminnallisuudesta uusittiin, joten uusi versionumero on perusteltu.

Kohdassa 13 kerrotaan paljon aikaa vievistä pyynnöistä ja näitä koskevista vaatimuksista. Tarkasteltavassa rajapinnassa ainoa mahdollisesti paljon aikaa vievä operaatio on tehtävän tarkistus. Ohjeistuksessa suositellaan prosessin tilan kertovan tiedon lisäämistä resurssiin, jonka operaatiot voivat kestää kauan. Esimerkiksi palautuksella voi olla vaikkapa tilat: `"tarkistuksessa"` ja `"tarkistettu"`. Tällöin asiakas näkee tarkastuksen tilan tekemällä resurssiin GET-pyyynnön. Juuri tällä tavoin tehtävien tarkastusoperaatio on arvioitavassa rajapinnassa toteutettu.

Kohdassa 14 kerrotaan tiedonvälitysmekanismista nimeltä `"webhook"`, joka sijaitsee rajapinnassa. Webhookia käytetään ilmoittamaan asiakkaalle, kun jokin asia palvelimella on valmistunut. Se on ns. käänteinen API, eli se lähettää ilmoituksen asiakkaalle sen sijaan, että asiakas lähettäisi kyselyjä palvelimelle. Tätä varten asiakkaalle pitää toteuttaa rajapinta ja sen osoite pitää näkyä julkisessa internetissä [49]. Arvioitavassa rajapinnassa olisi hyötyä webhookista, koska pahimmassa tapauksessa 500 opiskelijaa lähettää kyselyjä samanaikaisesti kurssipalvelimelle tarkastustulosten haussa, mikä voi aiheuttaa suorituskykyongelmia palvelinpäässä.

6.4 Yhteenveto

Tässä luvussa arvioimme, kuinka tekemämme oppimisympäristö onnistui vastaamaan tutkimuskysymyksiimme. Arvioimme järjestelmää yhtenäisyyden, osien etsinnän onnistuneisuuden ja kommunikointitapojen hyvyyden kannalta.

Vaatus yhtenäisyydestä todettiin saavutetuksi, koska oppilas pystyy käyttämään järjestelmän kaikkia ominaisuuksia yhdellä kirjautumisella ja IDE saatiin yhdistettyä LMS:ään. Myös osien valinta implementoidun järjestelmän osalta oli onnistunut, koska kaikki vanhan järjestelmän ominaisuudet saatiin toteutettua myös uuteen järjestelmään. Komponenttien välistä kommunikaatiota arvioitiin A+_n REST-rajapinnan onnistuneisuuden arvioinnilla. Tähän käytettiin Richardsonin kypsyydshallia ja Microsoftin REST-rajapintaohjeistusta. Rajapinnan todettiin olevan kypsyydshallin kolmannella, eli parhaalla tasolla ja Microsoftin ohjeistuksen useimmat vaatimukset täyttyivät, joten rajapintakin voidaan todeta onnistuneeksi.

7. YHTEENVETO JA JATKOKEHITYS

Tässä diplomityössä etsittiin vastausta tutkimuskysymyksiin: ”Miten pystymme rakentamaan yhtenäisen ohjelmoinnin oppimisympäristön? Mitä osia sen pitää sisältää ja miten näiden tulisi kommunikoida keskenään?”. Vastauksia etsittiin tapaustutkimuksen avulla, jossa toteutettiin oppimisympäristö Tampereen teknillisessä yliopistossa pidettävälle Johdatus ohjelmointiin -kurssille. Lähtökohtana ympäristölle oli, että se rakennetaan palveluorientoituneen A+-nimisen oppimisenhallintajärjestelmän ympärille. Ensimmäiseen tutkimuskysymykseen saammekin vastauksen palveluorientoituneisuudesta.

Tapaustutkimus lähti liikkeelle luvussa 2 kurssin ympäristölle kohdistamien vaatimusten kartoittamisella. Näiden vaatimusten pohjalta määrittelimme ympäristön käyttötapaukset eri käyttäjäryhmien osalta ja jaoimme ominaisuudet seuraaviin komponentteihin: LMS, IDE, automaattinen tarkistus, ohjelmavisualisointi, sosiaalisointi.

Komponenttien tunnistamisen jälkeen tarkastelimme luvussa 3 potentiaalisia integraatiovaihtoehtoja eri näkökulmista. Tässä käytettiin apuna sovellettua 4+1 -arkkitehtuurimallia. Looginen näkökulma rakennettiin KnowledgeTree-arkkitehtuurin avulla. Prosessinäkökulmaa tarkasteltaessa taas päädyttiin siihen, että REST ja LTI sopivat komponenttien välisiksi kommunikaatoratkaisuiksi. LTI on oppimistyökalujen välillä käytetty standardi kommunikointiratkaisu, joten monet valmiit komponentti-implementaatiot tukevat sitä. Tämä helpottaa uusien komponenttien integroimista järjestelmään. REST:illä puolestaan voidaan tehdä kommunikaatoratkaisuja, jotka vastaavat LTI:tä paremmin komponenttien kommunikointivaatimuksiin.

Luvussa 4 teimme kirjallisuuskatsauksen komponentti-implementaatioihin, jotka voidaan liittää A+:aan. Tässä valikoituivat käytettäväksi Mooc-grader, PyCharm, Python Tutor, Jsvee ja Piazza. Näin saimme vastauksen toisen tutkimuskysymyksen ensimmäiseen osaan. Piazza käyttää A+:n kanssa kommunikoidessa LTI-protokollaa. Mooc-grader ja PyCharm taas kommunikoivat A+:n kanssa käyttäen tämän REST-rajapintaa, jonka toteutuksesta kerrottiin luvussa 5. Nämä kommunikaatiovalinnat vastaavat toisen tutkimuskysymyksen toiseen osaan.

Työssä toteutettu järjestelmä on yhtenäinen kokonaisuus, jossa oppilas voi käyttää kaikkia toiminnallisuuksia yhden kirjautumisen kautta. Esiteltyä arkkitehtuu-

ria voidaan käyttää minkä tahansa komponentti-implementaatioiden kanssa. Tämän ansiosta työn tuloksia voidaan käyttää hyväksi ohjelmointikurssien kehittämisessä riippumatta niissä käytetyistä työkaluista.

Otimme uuden järjestelmän käyttöön Johdatus Ohjelmointiin -kurssin 11 viikkoa kestäväällä kesätoteutuskerralla vuonna 2016. Järjestelmä toimi kaiken kaikkiaan hyvin, joten sen käyttöä jatkettiin syksyn 2016 toteutuskerralla. Järjestelmä sisälsi alusta alkaen A+- ja Mooc-grader -web-palvelimet, joihin päivitettiin kurssimateriaaleja kurssin edetessä. Kurssimateriaalin teossa työtä tuottivat mahdollisimman helppokäyttöisen kansiorakenteen kehittäminen tehtäville, testiskriptien toteuttaminen, Mooc-graderin toimintaperiaatteen sisäistäminen ja muokkaaminen omiin tarpeisiin sopivaksi sekä testitapausten teko.

Diplomityön kirjoitushetkellä olemme aloittamassa PyCharm-pluginin käytön testausta assistenttien kesken, minkä jälkeen se annetaan oppilaiden käyttöön. Täten toteuttamamme järjestelmän käyttö ja kehitys ei pääty tähän.

Jatkokehitysmahdollisuudet

Jotta työ ei olisi kasvanut liian suureksi, karsittiin esille tulleiden vaatimusten joukkoa. Pois jätettiin A+:sta erillään oleva manuaalinen arviointikomponentti, vertaisarviointikomponentti, opiskelijoiden seuraaminen, tenttijärjestelmän integrointi A+:aan ja pelillistäminen.

Tärkein jatkokehityskohde on vaatimuksiimme vastaavan manuaalisen arviointikomponentin liittäminen järjestelmäämme. Työssä päädyttiin Rubyriin ja se on tarkoitus integroida järjestelmään mahdollisimman pian.

Toiseksi tärkein jatkokehityskohde on vertaisarviointikomponentin implementointi, koska vertaisarviointi tulee olemaan yksi kurssin osa-alue tulevilla toteutuskerroilla. Vertaisarviointi voidaan toteuttaa esimerkiksi erillisellä web-komponentilla, joka yhdistetään A+:aan käyttäen tämän REST-rajapintaa. Tätä varten toki joudutaan tekemään kyseiseen rajapintaan lisäominaisuuksia, joiden avulla kommunikointi onnistuu.

Ohjelmoinnin opetuksen kehittämiseen ja tutkimiseen tarvitaan opiskelijoiden käyttäytymiseen liittyvää dataa. Tätä dataa voidaan kerätä Tin Can API:lla ja analysoida esimerkiksi CodeBrowser-nimisellä ohjelmalla. Datasta voidaan analysoida esimerkiksi yleisimpiä virheitä, joita opiskelijat ohjelmoinnin alussa tekevät.

Tenttitulosten saaminen samaan järjestelmään muiden tietojen kanssa helpottaisi kurssivastuuhenkilöiden työtaakkaa arvosanojen kirjaamisessa, joten A+ halutaan yhdistää EXAM:iin.

Pelillistäminen voisi motivoida opiskelijoita hankkimaan saavutuksia ja lisäksi mahdollisesti tervettä kilpailullisuutta opiskelijoiden kesken. Tätä mahdollisuutta

pitää kuitenkin tutkia vielä tarkemmin ennen toteuttamista.

Esitellyistä jatkokehityskohteista tulemme suurella todennäköisyydellä liittämään Rubyricin osaksi järjestelmäämme ja kehittämään vertaisarviointikomponentin.

LÄHTEET

- [1] A REST API for Result Resources. <https://www.imsglobal.org/lti/model/uml/purl.imsglobal.org/vocab/lis/v2/outcomes/Result/service.html>. [Viitattu 14.08.2016].
- [2] Add Piazza to your LMS via LTI. <https://s3.amazonaws.com/piazza-materials/LTI-Setup/GetAProgrammerStartedWithLTI.pdf>. [Viitattu 30.07.2016].
- [3] K. Ala-Mutka. *Automatic assessment tools in learning and teaching programming*. Tampereen teknillinen yliopisto. Julkaisu. Tampere University of Technology, 2005. Awarding institution:Tampere University of Technology.
- [4] C. Alario-Hoyos, M. L. Bote-Lorenzo, E. Gómez-Sánchez, J. I. Asensio-Pérez, G. Vega-Gorgojo, and A. Ruiz-Calleja. Gluel: An architecture for the integration of external tools in virtual learning environments. *Computers & Education*, 60(1):122–137, 2013.
- [5] C. Alario-Hoyos and S. Wilson. Comparison of the main alternatives to the integration of external tools in different platforms. In *Proceedings of the International Conference of Education, Research and Innovation, ICERI*, pages 3466–3476. Citeseer, 2010.
- [6] T. Auvinen. Rubyric-a rubrics-based online assessment tool for effortless authoring of personalized feedback. Master’s thesis, Citeseer, 2009.
- [7] T. Auvinen. Rubyric - yksilöllistä sanallista palautetta massaopetuksessa. 2011.
- [8] P. Brusilovsky. Knowledgetree: A distributed architecture for adaptive e-learning. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 104–113. ACM, 2004.
- [9] D. Dagger, A. O’Connor, S. Lawless, E. Walsh, and V. P. Wade. Service-oriented e-learning platforms: From monolithic systems to flexible services. *IEEE Internet Computing*, 11(3):28–35, 2007.
- [10] Django REST Framework. <http://www.django-rest-framework.org/>. [Viitattu 31.08.2016].
- [11] Django Rest Framework - Authentication. <http://www.django-rest-framework.org/api-guide/authentication/>. [Viitattu 04.09.2016].

- [12] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.
- [13] M. Fowler. Richardson Maturity Model. <http://martinfowler.com/articles/richardsonMaturityModel.html>. [Viitattu 06.08.2016].
- [14] Get an LRS. <https://tincanapi.com/get-lrs/>. [Viitattu 28.06.2016].
- [15] Guided Video Tours of NetBeans IDE. <https://netbeans.org/kb/docs/intro-screencasts.html>. [Viitattu 12.03.2016].
- [16] P. Guo. Embedding Online Python Tutor visualizations. <https://github.com/pgbovine/OnlinePythonTutor/blob/master/v3/docs/embedding-HOWTO.md>. [Viitattu 28.07.2016].
- [17] P. J. Guo. Online Python Tutor: Embeddable web-based program visualization for CS education. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 579–584, New York, NY, USA, 2013. ACM.
- [18] L. Haaranen, P. Ihanola, and A. Hakulinen, Lasse ja Korhonen. How (not) to introduce badges to online exercises. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 33–38. ACM, 2014.
- [19] J. Hattie and H. Timperley. The power of feedback. *Review of educational research*, 77(1):81–112, 2007.
- [20] K. Heinonen, K. Hirvikoski, M. Luukkainen, and A. Vihavainen. Using code-browser to seek differences between novice programmers. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 229–234. ACM, 2014.
- [21] J. Helminen, P. Ihanola, and V. Karavirta. Recording and analyzing in-browser programming sessions. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, pages 13–22. ACM, 2013.
- [22] Ieee recommended practice for architectural description of software-intensive systems. *IEEE Std 1471-2000*, pages i–23, 2000.
- [23] IMS Learning Tools Interoperability A Briefing Paper. <http://publications.cetis.org.uk/wp-content/uploads/2012/05/LTI-Briefing-Paper.pdf>. [Viitattu 24.03.2016].

- [24] J. Isotalo. TMC apista. <https://gist.github.com/jamox/e6705a9b045369bc7a70189fde9db473#file-tmc-api-md>. [Viitattu 22.07.2016].
- [25] V. Karavirta and P. Ihanola. Automatic assessment of javascript exercises.
- [26] V. Karavirta, P. Ihanola, and T. Koskinen. Service-oriented approach to improve interoperability of e-learning systems. In *Advanced Learning Technologies (ICALT), 2013 IEEE 13th International Conference on*, pages 341–345. IEEE, 2013.
- [27] K. Koskimies and T. Mikkonen. *Ohjelmistoarkkitehtuurit*. Talentum, 2005.
- [28] P. Kruchten. Architectural blueprints—the “4+ 1” view model of software architecture. *Tutorial Proceedings of Tri-Ada*, 95:540–555, 1995.
- [29] J. Kurhila and A. Vihavainen. Management, structures and tools to scale up personal advising in large programming courses. In *Proceedings of the 2011 conference on Information technology education*, pages 3–8. ACM, 2011.
- [30] J. P. Leal, R. Queirós, and D. Ferreira. A contribution to the e-framework: a specification of a programming exercise evaluation service. 2010.
- [31] Learning Locker Features. <https://learninglocker.net/features/>. [Viitattu 15.08.2016].
- [32] T. Lehtinen. A+. <https://github.com/Aalto-LeTech/a-plus>. [Viitattu 16.04.2016].
- [33] T. Lehtinen. Mooc-grader. <https://github.com/Aalto-LeTech/mooc-grader>. [Viitattu 13.03.2016].
- [34] P. LePage. HTML5 Video. <http://www.html5rocks.com/en/tutorials/video/basics/>. [Viitattu 01.08.2016].
- [35] M. Linden. *Identiteetin- ja pääsynhallinta*. Tampere University of Technology. Department of Pervasive Computing. Report. Tampere University of Technology, 2015. 16.12.2015.
- [36] LTI Connector for WebPA 2. <http://www.spvsoftwareproducts.com/php/webpa-lti/>. [Viitattu 06.08.2016].
- [37] LTI2 Introduction. <https://www.imsglobal.org/lti-v2-introduction>. [Viitattu 24.03.2016].

- [38] M. Luukkainen. Ohje JUnit:in käyttöön. <https://github.com/mluukkai/OTM2012/wiki/Ohje-JUnit:in-k%C3%A4ytt%C3%B6%C3%B6n>. [Viitattu 26.07.2016].
- [39] Microsoft REST API Guidelines. <https://github.com/Microsoft/api-guidelines/blob/master/Guidelines.md>. [Viitattu 08.08.2016].
- [40] L. Mykkänen, Juhani ja Liukas. Koodi2016. 2014.
- [41] S. Ninoriya, P. Chawan, B. Meshram, and M. VJTI. Cms, lms and lcms for elearning. *IJCSI International Journal of Computer Science*, 8(2):644–647, 2011.
- [42] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson. A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin*, 39(4):204–223, 2007.
- [43] PyCharm Features. <https://www.jetbrains.com/pycharm/features/>. [Viitattu 27.07.2016].
- [44] Python support in NetBeans. <http://wiki.netbeans.org/Python>. [Viitattu 12.03.2016].
- [45] J. Pärtel, Martin ja Isotalo. TestMyCode. <http://testmycode.github.io/>. [Viitattu 11.03.2016].
- [46] J. Pärtel, Martin ja Isotalo. TestMyCode Usermanual. <http://testmycode-usermanual.github.io/>. [Viitattu 11.03.2016].
- [47] R. Queirós and J. P. Leal. Orchestration of e-learning services for automatic evaluation of programming exercises. *J. UCS*, 18(11):1454–1482, 2012.
- [48] R. A. P. Queirós and J. P. Leal. Petcha: a programming exercises teaching assistant. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, pages 192–197. ACM, 2012.
- [49] N. Quinlan. What’s a Webhook? <https://sendgrid.com/blog/whats-webhook/>. [Viitattu 07.09.2016].
- [50] Sakai Features. <https://www.sakaiproject.org/features-tools-functionality>. [Viitattu 28.06.2016].
- [51] J. A. Sant. Mailing it in: email-centric automated assessment. *ACM SIGCSE Bulletin*, 41(3):308–312, 2009.

- [52] C. Severance, T. Hanss, and J. Hardin. Ims learning tools interoperability: Enabling a mash-up approach to teaching and learning tools. *Technology, Instruction, Cognition and Learning*, 7(3-4):245–262, 2010.
- [53] T. Sirkiä. Jsvee. <https://github.com/Aalto-LeTech/jsvee>. [Viitattu 27.07.2016].
- [54] T. Sirkiä. Exploring expression-level program visualization in cs1. In *Proceedings of the 14th Koli Calling International Conference on Computing Education Research*, pages 153–157. ACM, 2014.
- [55] J. Stasko. *Software visualization: Programming as a multimedia experience*. MIT press, 1998.
- [56] The Base16, Base32, and Base64 Data Encodings. <https://tools.ietf.org/html/rfc4648>. [Viitattu 09.09.2016].
- [57] Tin Can API Client Libraries. <https://tincanapi.com/libraries/>. [Viitattu 28.06.2016].
- [58] P. Todd. Delta Queries (Part 1). <http://scn.sap.com/blogs/pault/2014/03/08/delta-queries-part-1>. [Viitattu 17.09.2016].
- [59] Understanding CORS. <https://spring.io/understanding/CORS>. [Viitattu 17.09.2016].
- [60] unittest — Unit testing framework. <https://docs.python.org/3.4/library/unittest.html>. [Viitattu 08.07.2016].
- [61] K. Uolia. EXAM. <https://confluence.csc.fi/display/SITNET/Rajapinnat+ja+tietointegraatiot>. [Viitattu 17.08.2016].
- [62] Using HTTP Methods for RESTful Services. <http://www.restapitutorial.com/lessons/httpmethods.html>. [Viitattu 07.06.2016].
- [63] Vanilla Features. <https://vanillaforums.com/features/user-experience>. [Viitattu 30.07.2016].
- [64] A. Vihavainen, J. Helminen, and P. Ihanola. How novices tackle their first lines of code in an ide: Analysis of programming session traces. In *Proceedings of the 14th Koli Calling International Conference on Computing Education Research*, pages 109–116. ACM, 2014.
- [65] A. Vihavainen, M. Luukkainen, and J. Kurhila. Multi-faceted support for mooc in programming. In *Proceedings of the 13th annual conference on Information technology education*, pages 171–176. ACM, 2012.

- [66] A. Vihavainen and M. Paksula, Matti ja Luukkainen. Extreme apprenticeship method in teaching programming for beginners. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 93–98. ACM, 2011.
- [67] A. Vihavainen, T. Vikberg, M. Luukkainen, and M. Pärtel. Scaffolding students' learning using test my code. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pages 117–122. ACM, 2013.
- [68] V. Vironmäki. Isotalo, Jarmo 2016. [Haastattelu] 27.07.2016. Haastattelijana Ville Vironmäki.
- [69] A. Virtanen. Konstruktiivinen tutkimusote. *Ammattikasvatuksen aikakauskirja*, 1(2006):47–49, 2006.
- [70] VPL, the Virtual Programming lab for Moodle. <http://vpl.dis.ulpgc.es/>. [Viitattu 05.07.2016].
- [71] WebPA - An Online Peer Assessment System for HE. <https://github.com/WebPA/Source>. [Viitattu 06.08.2016].
- [72] What is REST? <http://www.restapitutorial.com/lessons/whatisrest.html>. [Viitattu 07.06.2016].
- [73] What is the Tin Can API? <https://tincanapi.com/overview/>. [Viitattu 28.06.2016].
- [74] Why Piazza Works. <https://piazza.com/product/overview>. [Viitattu 30.07.2016].