OULUN AMMATTIKORKEAKOULU

Asko Mattila

OPINTOJEN TUNTISEURANNAN JÄRJESTELMÄ

OPINTOJEN TUNTISEURANNAN JÄRJESTELMÄ

Asko Mattila
Opinnäytetyö
Syksy 2020
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, Ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä(t): Asko Mattila
Opinnäytetyön nimi: Opintojen tuntiseurannan tietojärjestelmä
Työn ohjaaja(t): Eino Niemi
Työn valmistumislukukausi ja -vuosi: Syksy 2020 Sivumäärä: 31 + 14 liitettä

Opinnäytetyön tavoitteena oli toteuttaa prototyyppitasolla, REST-rajapinta, jolla opiskelija, opettaja ja oppilaitos voivat seurata opintoihin käytettyä todellista aikaa. Opiskelijalle ajanmukainen seuranta auttaa ajankäytönhallinnassa, opettajalle ja oppilaitokselle se antaa työkalun resurssien käytön seuraamiseen ja ohjaamiseen. Toteutuksellisesti valmis järjestelmä sisältää myös prototyypin asiakasohjelmasta yleisimmille mobiilikäyttöjärjestelmille. Asiakasohjelma käyttää järjestelmää varten suunniteltua ja toteutettua rajapintaa, joka tallentaa saamansa datan tietokantaan.

Asiasanat: Ajankäyttö, tietokantaohjelmat, ohjelmistosuunnittelu, rajapinnat

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Program development

---

Author(s): Asko Mattila
Title of thesis: Opintojen tuntiseurannan tietojärjestelmä
Supervisor(s): Eino Niemi
Term and year when the thesis was submitted: Fall 2020
Number of pages: 31 + 14 appendices

Goal of this thesis was to design and build Rest API providing way to students track their time usage on their studies, while also enabling school to use the collected data to plan their future studies. API is also designed so that teacher has possibility to use it as an electronic timesheet.

Keywords: API, database, REST, time management

SISÄLLYS

SANASTO

Anonyymi käyttäjätunnus = (AnonID) järjestelmän luoma satunnaisesti luotu käyttäjätunnus

Docker-compose	= Useamman Docker-kontin määrittelyyn tarvittava asetustiedosto
Dockerfile	= Dockerin käyttämä tekstitiedosto, jolla annetaan käännön aikaisia komentoja levykuvalle
Docker image	= Docker käyttämä levykuva, josta kontti luodaan
Golang	= GO-ohjelmointikieli
GORM	= Object-relational mapping for Golang -moduuli, mahdollistaa relaatio tietokannan käytön tietorakenteiden avulla.
HTTP	= Hypertext Transfer Protocol, TCP-yhteyttä käyttävä yhteyskäytäntö
Image	= Levykuva
Kontti	= Docker-ohjelmiston käyttämä kapseloitu ohjelmisto.
Kurssi	= Opintokokonaisuus, joka koostuu yhdestä tai useammasta osiosta.
Lint / Linting	= Työkalu, joka analysoi koodin virheiden varalta.
Merkintä, Sessio	= Kirjaus, jolla on alkamis- ja loppumisajankohta.
MUX	= Multiplekseri. RESTissä ohjaa HTTP-pyynnön funktioille, jonka halutaan käsittelevän pyynnön.
Osio, Segmentti	= Kurssin itsenäisesti seurattava osio.
Päätepiste	= Endpoint. URL-osoite, josta rajapinnan tarjoamaa palvelua käytetään.
Rajapinta	= API (Application Programming Interface). Rajapinnan kautta järjestelmän palvelut ovat käytettävissä asiakasohjelmille.
REST	= Representational State Transfer, HTTP:hen perustuva arkkitehtuurimalli
UI	= Käyttöliittymä
URL	= Uniform Resource Locator
UX	= Käyttäjäkokemus
WSL2	= Window Subsystem for Linux 2

1 JOHDANTO

Ajankäytön hallinta on osa opintojen onnistunutta suorittamista (Nyyti Ry 2020). Usein opiskelijat kokevat ajan subjektiivisesti ja se voi niin lentää kuin madellakin. Oppimalla suunnittelemaan omaa ajankäyttöä varmistaa opiskelun etenemisen ja oman jaksamisen. Mitkä osat opinnoista hyötyisivät suuremmasta ajallisesta panostuksesta? Ehkä on mahdollista löysätä tahtia ja varmistaa että jaksaa myös tulevaisuudessa? Myös lisääntyneet etäopiskelut ja -työt vaativat enemmän vastuuta ajan käytöstä ja tehtäviin tarvittavan ajan arvioinnista. Rutiini aikakirjanpidon tekemiseen olisi siis hyödyllistä oppia jo opintojen aikana.

Opiskelijat eivät ole ainoat, jotka hyötyisivät keskitetystä ajanseurannasta. Opetuksesta vastaava voi halutessaan käyttää järjestelmää työaikakirjanpitoon sekä läsnäolonkirjaukseen. Opetuksen suunnittelussa voidaan käyttää pitkän aikavälin tietojen kertymää työkaluna resurssien ohjauksessa.

Työelämän käyttöön on tarjolla useita ohjelmistollisia vaihtoehtoja. Löytyy myös erilaisia vaihtoehtoja henkilökohtaisen kirjanpidon laatimiseen. Henkilökohtaisen kirjanpidon ongelmana on, että kerääntynyt data ei ole oppilaitoksen saatavilla. Yrityksien käyttämät ratkaisut taas eivät välttämättä ole kustannustehokas ratkaisu, tai niiden tarjoamat ratkaisut eivät vastaa oppilaitoksen ja opiskelijoiden tarpeita. Erityisesti opiskelijoille ja oppilaitoksille suunnattujen ratkaisujen olemassaolosta en löytänyt viitteitä

Opinnäytetyön tarkoituksena on suunnitella ja toteuttaa prototyyppi järjestelmästä, joka mahdollistaisi ajanseurannan räätälöitynä vastaamaan niin oppilaitoksen kuin opiskelijoiden vaatimuksia. Käyttöönotettaessa se sisältää asiakasohjelman, jolla opiskelija voi luoda merkintöjä, muokata niitä sekä seurata edistymistään. Tärkein osa järjestelmää on kuitenkin REST-asiakasrajapinta, joka hoitaa tiedon tallentamisen, hakemisen ja käsittelyn tietokannan ja asiakasohjelman välillä. Rajapinnan käyttö mahdollistaa asiakasohjelmien itsenäisen kehittämisen ja niiden vaihtamisen käyttöympäristön ja tarpeen mukaan. Koska kyseinen rajapinta on kriittinen toimivalle järjestelmälle, pääpaino on työssä rajapinnan suunnittelulla ja toteutuksella.

2 JÄRJESTELMÄN TOTEUTUKSEEN KÄYTETYT TEKNOLOGIAT

Tämän opinnäytetyön tarkoitus ei ole syvällinen perehtyminen käytettyihin teknologoihin, siksi käydään ne tässä läpi vain lyhyesti.

2.1 REST-arkkitehtuurimalli

Järjestelmän rajapinta (API) päätettiin toteuttaa käyttämällä noudattamalla REST-arkkitehtuurimallia. Päätös perustuu sen suureen suosioon, jolloin sille on tarjolla paljon materiaalia ja työkaluja suunnitteluun sekä toteutukseen

REST on arkkitehtuurimalli, jonka määritteli Roy Fielding vuonna 2000 (Wikipedia Foundation 2020). Se on tilaton protokolla, eli jokainen kutsu käsitellään itsenäisesti ennalta määrättyjen toimenpiteiden mukaisesti. Fielding, kehitti RESTiä saamaan aikaan, kun oli mukana HTTP 1.1 (RFC2616) kehityksessä (Fielding 2006). REST käyttääkin HTTP:n metodeja, kuten GET ja POST.

2.2 GO-ohjelmointikieli

Gon julkaisi Google vuonna 2009 avoimen lähdekoodin ohjelmointikielenä (Kincaid 2009). Se sai alkunsa Googlella työskentelevien Robert Griesemerin, Rob Piken ja Ken Thompsonin turhautumisesta siihen aikaan käytettyjen kielien monimutkaisuudesta kehitettäessä palvelinohjelmistoja (The Go Project 2020). Sillä onkin mahdollista toteuttaa yksinkertainen nettipalvelin vain muutamalla rivillä koodia (The Go Project 2020). Go valittiin muista mahdollisista ohjelmointikielistä, joilla REST-rajapinnan olisi voinut toteuttaa, alalla työskentelevän ystävän suosituksesta.

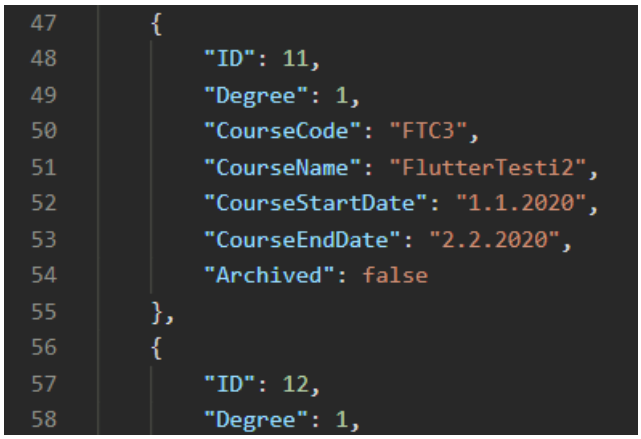
2.3 MySQL-relaatiotietokanta

Järjestelmän tietokanta käytetään MySQL 8 -relaatiotietokantaa. Sen kehitti vuonna 1994 ruotsalainen MySQL AB ja vuodesta 2010 lähtien sen on ollut Oracle Corporationin omistuksessa (European Commission 2010). Tutkimuksien mukaan se oli vuonna 2019 suosituin tietokanta (Explore Group 2019).

Käytetyksi tietokannaksi valittiin MySQL juuri sen suosion vuoksi, koska se kasvattaa saatavilla olevan tiedon ja yhteensopivien työkalujen määrä. Tarkoituksena on kuitenkin toteuttaa REST-rajapinta niin, että taustalla käytetyn tietokannan vaihto olisi kohtuullisen vaivatonta. Ei-relaatiotietokannan (NoSQL), kuten MongoDB tai Firestore. Niiden käyttöönotto on kuitenkin todennäköisesti vaikeaa niiden poikkeavan rakenteen vuoksi ja vaatii tietokantarakenteen uudelleen suunnittelua.

2.4 JSON

JSON (JavaScript Object Notation) on vuonna 2000 Douglas Crockfordin (Wikipedia Foundation 2020) määrittelemä ja vuonna 2013 standardoima avoin tiedostomuoto. JSON valittiin tiedonsiirtoon rajanpinnan ja asiakasohjelmien välillä koska se on yksi suosituimmista formaateista ja siihen löytyy valmis tuki useimmista ohjelmistoympäristöistä (kuva 1).



KUVA 1. Esimerkki järjestelmän palauttamasta JSON-vastauksesta POSTMAN-ohjelmassa.

2.5 Flutter-ohjelmistokehityspaketti

Asiakasohjelman prototyypin toteuttamiseen on tarkoitus käyttää Googlen avoimen koodin Flutter UI -ohjelmistokehityspakettia (Google, LLC 2020). Google julkaisi Flutterin 2017, ja kuten React Native se mahdollistaa ohjelmien kehittämisen useammalle alustalle samanaikaisesti. Tällä hetkellä Flutterilla tehtyä koodia voidaan suorittaa Android ja iOS-laitteilla, nettisivu ja työpöytätuki ovat kuitenkin kehitteillä. Flutter valittiin, kypsemmän ja suositumman React Nativen sijaan, koska ohjelmointiympäristönä voidaan käyttää Android Studiota ja sen käytöstä oli jo aikaisempaa kokemusta.

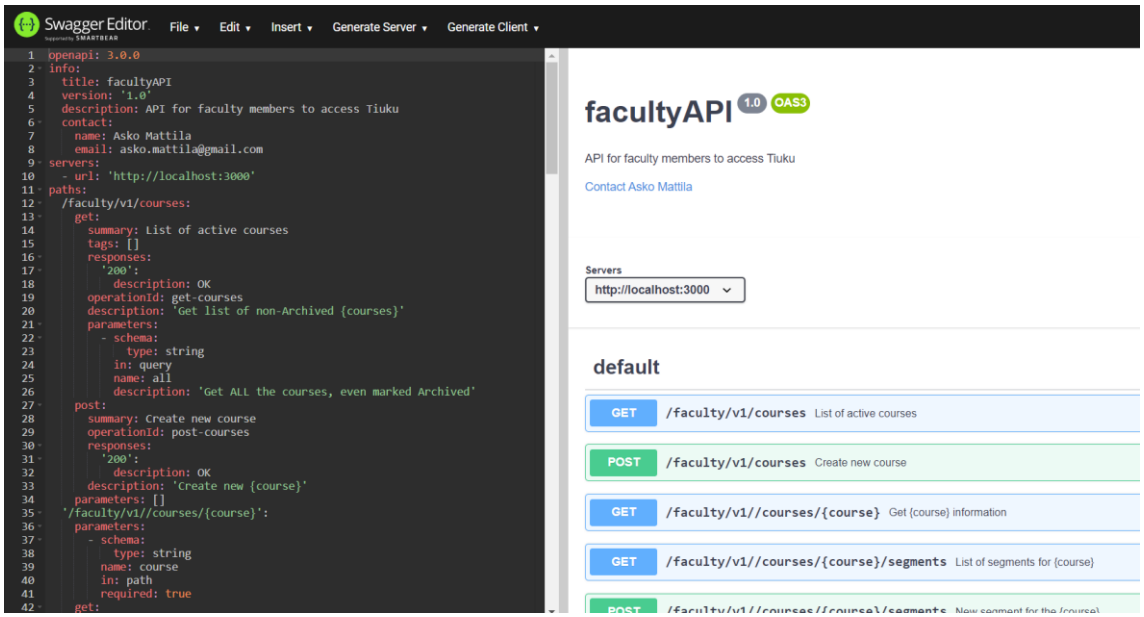
2.6 Docker

Docker on vuodesta 2013 (Wikipedia Foundation 2020) asti julkaistu kokoelma käyttöjärjestelmätason virtualisoinnin mahdollistavia tuotteita. Docker-kontti poikkeaa virtuaalikoneesta huomattavasti pienemmällä koolla ja vähempien resurssien tarpeella. Konteilla ei ole omaa käyttöjärjestelmää ja ne ovat tarkoitettu yleensä vain yhden tehtävän suorittamiseen ja sisältää sen suorittamiseen vaadittavat resurssit eikä mitään ylimääräistä. Niiden avulla ohjelmiston siirtäminen kehitysympäristöstä tuotantoon on nopeampaa, helpompaa ja varmempaa. Myös ongelmatilanteista konttien vaihtaminen tai palauttaminen aikaisempaan tilaan on nopeaa.

3 TYÖKALUT, KEHITYSYMPÄRISTÖT JA NIIDEN KÄYTTÖÖNOTOT

3.1 REST-rajanpinnan suunnittelu

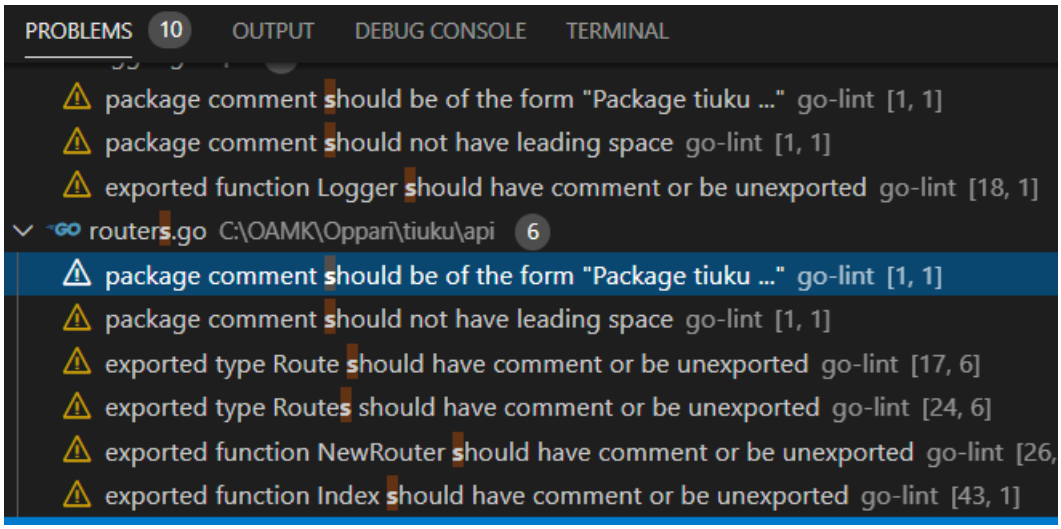
RESTin suunnittelussa tärkeää on rajapinnan tarjoamat päätepisteet. Niiden alustavassa suunnittelussa käytettiin kynää ja paperia. Tarkempi suunnittelu tehtiin käyttäen ilmaisia ohjelmia Stoplight Studio (Spotlight, Inc. 2020) ja editor.swagger.io (SmartBear Software 2020) (kuva 2). Stoplight Studiolla oli helppo luoda päätepiste ja asettaa sille palveluja. Havaittiin kuitenkin, että jälkimmäisellä on helpompi huomata puutteet ja mahdolliset ongelmat päätepisteissä. Se myös tarjoaa mahdollisuuden generoida valmista multiplekseri-koodia palvelimille ja asiakasohjelmille. Tätä ominaisuutta käytettiinkin hyväksi rajapinnan toteutuksen alkuun saattamisessa. Päätepisteiden testausta kehityksen aikana tehtiin Postmanilla (Postman, Inc 2020), joka mahdollisti HTTP-kutsujen helpon luomisen ja tallentamisen sekä nopean uudelleen lähettämisen.



KUVA 2. Rajapinnan suunnittelua Swagger.editor.io-ohjelmalla.

3.2 Go-ohjelmointiympäristö

Gon asentamisen lisäksi (The Go Project 2020) asennettiin myös Visual Studio Code (Microsoft Corporation 2020), joka on suosituimpia editoreita Go-ohjelmointiin. Sille löytyykin monia ohjelmointia helpottavia liitännäisiä ja laajennuksia, joiden asentamista se osaa tarpeen mukaan myös ehdottaa. Tämän työn kannalta tärkein laajennus on Go, jonka on tehnyt Go Team at Google. Linting-työkalu, joka tarkistaa koodin mahdollisten virheiden varalta, ja oletuksena on golint, vaihdettiin asetuksissa golangci-lintiin. Tämä tehtiin, koska golint antoi varoitusilmoituksia kommentoinnin puutteesta, ja vei huomion kehitystyön kannalta kriittisemmiltä virheiltä (kuva 3).



KUVA 3. Golintin virheilmoituksia kommentoinnin puutteesta.

Golle asennettiin moduuleita helpottamaan kehitystyötä. Yksi näistä oli gorilla/mux (Gorilla Toolkit 2020), HTTP-pyynnöistä vastaava multiplekseri, joka ohjaa pyynnöt oikeisiin päätepisteisiin. Se helpottaa huomattavasti päätepisteiden ohjelmointia ja se on myös vaatimuksena editor.swagger.io-sivun generoiman koodin käyttöön.

Toisen asennetun moduulin tarkoituksena on helpottaa tietokannan käyttöä. GORM (Jinzhu 2020) mahdollistaa tietokannan käsittelyn tietorakenteilla sen sijaan, että annettaisiin yksittäisiä SQL-käskyjä. GORMista on myös kehitteillä versio 2, mutta sen käytöstä oli tällä hetkellä heikommin materiaalia kuin versio 1:een.

Näiden moduulien lisäksi käytettiin moduulia uniuri (Chestnykh 2015), joka loi satunnaisluvut, joita käytettiin Anonyymien tunnuksien luomisessa.

3.3 Docker

Koska Docker on Linux-pohjainen, ja käytössä oleva tietokone on Windows 10 Home, on sen käyttöönotossa useita vaihtoehtoja.

Yksi olisi asentaa tuettu Linux-jakeluversio kuten Ubuntu, joko Windowsin tilalle tai rinnalle. Ensimmäinen ei ollut realistinen vaihtoehto, koska se poistaisi Windowsin kokonaan ja käytettyä tietokonetta käytetään myös opinnäytetyön ulkopuolella. Se voi myös rajoittaa käytettävissä olevien ohjelmistojen määrää.

Toinen vaihtoehto olisi asentaa rinnalle asentaminen, niin sanottuun kaksoiskäynnistykseen, jolloin voidaan tietokonetta käynnistäessä valita haluttu käyttöjärjestelmä. Silloin halutessa siirtyä käyttöjärjestelmästä toiseen pitää tietokone sammuttaa ja käynnistää joka kerta uudestaan, jolloin kehitystyö voi olla hitaampaa ja vaikeampaa kuin muissa vaihtoehdoissa. Mahdolliseksi ongelmaksi voi muodostua myös, ettei käytetyn tietokoneen yhteensopivuus Linux-käyttöjärjestelmien kanssa ole tiedossa. Tämä olisi kuitenkin ollut vaihtoehto, jos muut eivät toimi odotusten mukaisesti.

Alkuperäinen suunnitelma oli luoda Linux-virtuaalikone, johon Docker voidaan asentaa. Tämä suunnitelma ei kuitenkaan vaikuttanut järkevältä rajallisten tietokoneresurssien käytöltä ja toinen käyttöjärjestelmä kerros voi aiheuttaa ennalta arvaamattomia ongelmia.

Lopulta ratkaisu oli Windows Subsystem for Linux 2. WSL2-versiossa tulleet muutokset mahdollistavat Dockerin (Docker, Inc. 2020) ja Windows 10 Home 2004sen (Loewen 2020) käytön kehitysympäristönä. Tämän jälkeen WSL2 on tullut takautuvasti myös Home-versioihin 1903 ja 1909 (Citrin 2020). Koska käytetyssä tietokoneessa ei ollut asennettuna versio 2004, oli ensimmäinen tehtävä päivittää siihen. Päivityksen jälkeen asennettiin WSL2 käyttäen Microsoftin sivuilta löytyvää ohjetta (Microsoft Corporation 2020). Asennus ei kuitenkaan jostain syystä onnistunut, ennen kuin kohta ”/norestart” jätettiin pois. Tämän jälkeen Docker saatiin käyttöön yksinkertaisesti asentamalla Docker Desktop (Docker, Inc. 2020).

3.3.1 Docker-kontit

Docker-kontit käyttävät pohjanaan imageja, jotka sisältävät ohjeet niiden luomiseen. Imagen voi luoda itse tai kopioida valmiina Dockerhubista osoitteesta hub.docker.com, josta myös tämän opinnäytetyön imaget saatiin. Rajapintaa varten ladattiin Golang-image versio 1.15.1, joka oli uusin versio Go-kielelle, ja tietokantaa varten ladattiin image mysql:8.

3.3.2 Docker-compose

Kun halutaan, että kontit pystyvät kommunikoimaan keskenään, käytetään docker-composea. Käytännössä tämä tapahtuu luomalla tiedosto nimeltään docker-compose.yml (liite1). Tiedostossa voi määritellä, mitkä portit ovat avoinna ulkopuolelta tulevalle tietoliikenteelle, ja mitä portteja käytetään konttien väliseen tiedonsiirtoon. Tiedostossa asetetaan myös käyttäjänimi ja salasana tietokannalle.

Koska kontit eivät tallenna mitään vaan palautuvat sammutettaessa aina alkuperäiseen tilaan, määritellään tiedostossa lisäksi ”volumet”, joiden avulla yhdistetään konttien hakemistoja isäntäkoneen hakemistoihin. Näin saadaan tietokantaan tallennettu tieto pysymään tallessa sammutusten välissä, ja rajapinnan ohjelmoinnissa tämä tarkoittaa, että käytetty golang-kontti sisältää aina ajan tasalla olevan koodin. Kontit myös sammuvat automaattisesti suoritettuaan tehtävänsä. Joka golang-kontin kohdalla on ongelma, koska go-komento pitää pysäyttää ja suorittaa uudestaan, kun halutaan muutoksien tulevan voimaan. Tämä estettiin antamalla kontille käynnistyksessä käsky ”tail -fn0 /dev/null” ja manuaalisesti suorittamalla ”go run main.go” kontin sisällä. Näin pystyttiin testaamaan koodia paljon nopeammin kuin sammuttamalla ja käynnistämällä kontti aina, kun koodiin on tehty muutoksia.

3.4 Tietokanta

Tietokannan suunnittelu aloitettiin kynällä ja paperilla, josta jatkosuunnittelu ja visualisointi tehtiin Draw.io:lla (nykyään diagrams.net) (diagrams.net 2020). Visualisoinnin aikana alkuperäiset suunnitelmat saattoivat muuttua, kun huomattiin niiden heikkoudet tai tarpeettoman monimutkaiset rakenteet. Tietokannaksi seurattiin ja hallittiin kehitystyön aikana MySQL Workbench -ohjelmalla (Oracle Corporation 2020).

4 SUUNNITTELU

4.1 Laadulliset vaatimukset

Koska opinnäytetyöllä ei ollut tilaajaa tai toimeksiantajaa, laadulliset vaatimukset määriteltiin suunnitteluvaiheessa itse. Näin saatiin periaatteet, joita noudattaa suunnittelun ja toteutuksen aikana.

4.1.1 Helppous

Järjestelmässä, jonka toiminta on riippuvainen käyttäjien luomasta datasta, tässä tapauksessa kirjauksista, on hyvin tärkeää helppous, nopeus ja välittömyys. Onneksi älypuhelinten ja internet-yhteyksien yleisyys helpottaa tätä ongelmaa (Tilastokeskus 2019). Älypuhelin on käytössä lähestulkoon kaikilla opiskelijoilla ja melkein aina mukana. Siksi asiakasohjelmat yleisimmille älypuhelin alustoille, vuonna 2020 Android ja iOS (Statcounter 2020), on tehokas tapa kattaa mahdollisimman paljon käyttäjäkuntaa. Asiakasohjelmansuunnittelussa ja toteutuksessa on myös hyväksi tavoitella helppoa käytettävyyttä.

Vaikka rajapinta ei ole tarkoitettu tavallisten käyttäjien suoraan käytettäväksi, on sen suunnittelussa myös pidettävä tavoitteena selkeyttä ja loogisuutta. Näin ei pelkästään huomioida rajapintaa käyttävien asiakasohjelmien kehitystyötä tekeviä, vaan myös vähennetään suunnitteluvirheiden ja ei-toivottujen ominaisuuksien ilmenemistä.

4.1.2 Luotettavuus

Käyttäjien on pystyttävä luottamaan, että järjestelmän tarjoamat palvelut ovat saatavilla ja tieto tallessa. Tiedot tallennetaan oppilaitoksen tietokantaan, jolloin sen pysyvyys voidaan paremmin varmistaa. Palvelujen toimivuus myös poikkeustilanteiden aikana pitää ottaa huomioon. Esimerkiksi internet-yhteyden puuttumisen tai järjestelmän ylläpidon aikana rajapinta ja asiakasohjelma voivat puskuroida siirrettäviä tietoja, kunnes tilanne on ohi. Tämä vaatii puolestaan keinot ratkaista mahdollisesti muodostunut ristiriitainen data.

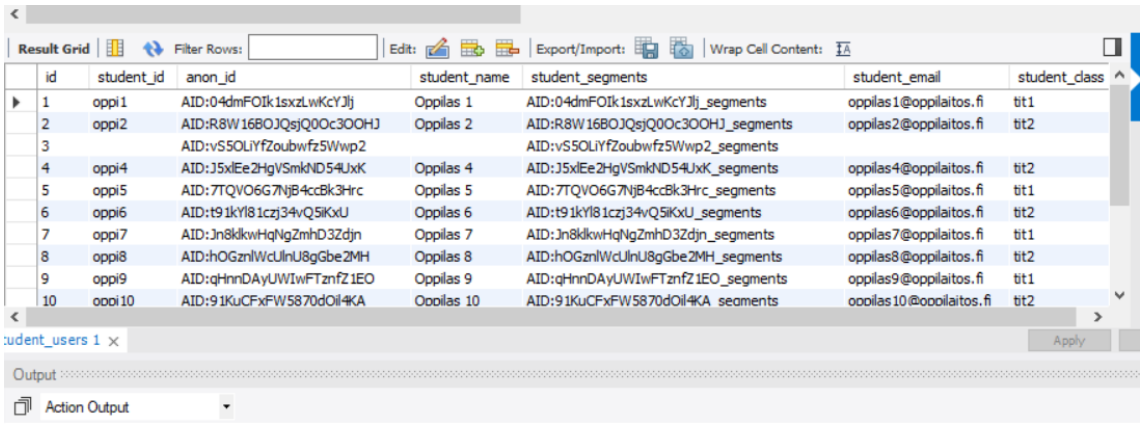
4.1.3 Joustavuus

Mukautuvuus muuttuviin ympäristön vaateisiin, kuten skaalautuminen muuttuviin käyttäjämääriin, vaihtuvaan laitteistoympäristöihin ja yhteysnopeuksiin, huomioidaan teknologiavalinnoilla ja rakenteen suunnittelussa.

4.1.4 Yksityisyys

Suunnittelussa on otettava huomioon käyttäjien yksityisyys. Järjestelmän käyttäjien pitää hyväksyä anonyymin datan tallentaminen käyttöönotossa. Kun anonyymius ei ole mahdollista, esimerkiksi kun vaaditaan seurattavaa tuntikirjanpitoa, käyttäjän pitää sallia se erikseen. Silti kyseistä tietoa pitää käsitellä niin, ettei kyseinen tieto näy käyttäjille, joilla ei ole siihen lupaa.

Oletuksena tieto kerätään pseudo-anonyyminä, niin että käyttäjätunnuksen tilalla on anonyymi käyttäjätunnus. Tätä tunnusta järjestelmä käyttää yhdistääkseen käyttäjän kirjaukset tilastojen luontia varten, eikä se saa kulkeutua järjestelmän ulkopuolelle. Kun käyttäjätunnus poistetaan käytöstä, poistetaan myös kaikki identifioiva data. Ainoastaan anonyymi käyttäjätunnus jätetään talteen, jotta vältetään sen uudelleen käyttö myöhemmin (kuva 4).



KUVA 4. Käyttäjätunnus ID 3 on poistettu järjestelmästä.

4.2 Toiminnalliset vaatimukset

Kuten laadullisten vaatimuksien kanssa, toiminnalliset vaatimukset määriteltiin suunnitteluvaiheessa itse.

4.2.1 Oppilaskäyttäjä

Oppilaskäyttäjän pitää pystyä tekemään vähintään seuraavat asiat:

- lisätä, muokata, poistaa ja selata merkintöjä
- liittyä ja poistua aktiivisen kurssin osiosta
- poistaa käyttäjätunnuksensa ja siihen liittyvä identifioiva data.
- Mahdollisena rajanpinnan lisäominaisuutena nähdä merkinnöistään luotuja tilastoja ja ennusteita.

Viimeisin voidaan tarjota myös asiakasohjelmassa, jolloin niiden tarjoaminen rajapinnan palveluna ei välttämättä ole tarpeen tai edes toivottua.

4.2.2 Opettajakäyttäjä

Opettajakäyttäjillä on mahdollisuus nähdä kurssin osion tuntikertymää ja kategorioita, luoda uusia kategorioita ja sulkea niitä. Opettajakäyttäjille olisi hyvä tarjota lisäominaisuutena tilastojen ja ennusteiden luontia. Toisin kuin oppilaskäyttäjät, näiden luomisessa käsitellään useita tauluja ja turvallisuuden vuoksi voi olla parempi käsitellä tämä data rajapinnassa.

4.2.3 Opetuksen suunnittelija

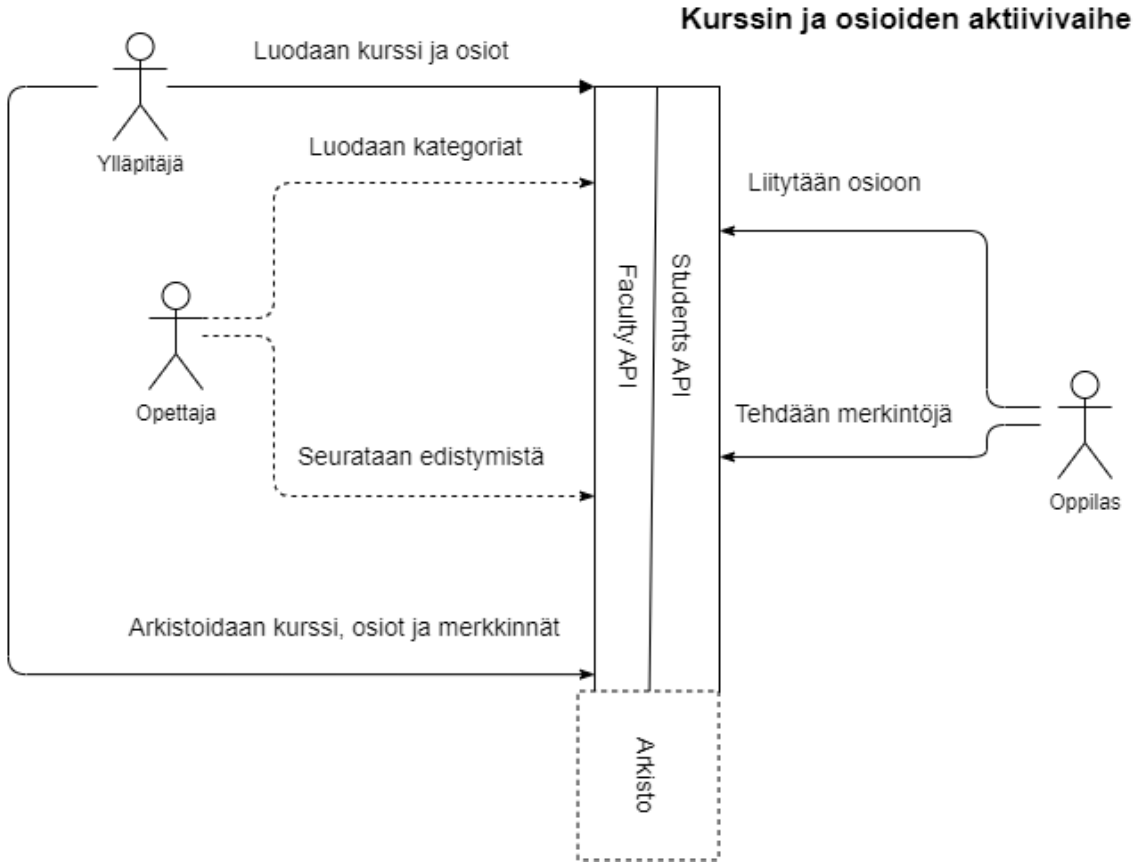
Opetuksen suunnittelija voi luoda ja arkistoida kursseja, lisätä osioita kursseihin ja määritellä niille opettajat ja luoda kattavia tilastoja arkistoidusta datasta.

4.3 Rakenteellinen suunnittelu

Järjestelmä on jaettu kolmeen osaan, joista jokaista voidaan kehittää ja vaihtaa toisistaan riippumatta. Asiakasohjelma käyttää rajapintaa, jolloin rajapinnalle ei ole merkitystä, millä alustalla ja miten se on toteutettu. Kunhan rajapinnan tarjoamat palvelut ja niiden osoitteet eivät muutu, asiakasohjelmalle taas ei ole merkitystä, miten rajapinta on sisäisesti toteutettu. Asiakasohjelma ja tietokanta eivät kommunikoi keskenään, ainoastaan rajapinnan kautta. Tietokannan vaihto toiseen voi kuitenkin vaatia muutoksia rajapinnan sisäisessä toiminnassa. Muutoksien määrä on todennä-

köisesti vähäistä, jos vaihdetaan johonkin toiseen GORM-moduulin tukemaan tietokantaan. Toisaalta vaihtaminen relaatiotietokannasta noSQL-tietokantaan, vaatii tietokannan rakenteen suunnittelemista uudelleen.

Rajapinta on ulkoisesti jaettu kahteen osaan. Students API on rajapinta oppilaskäyttäjille ja Faculty API on opettajille ja opetuksen suunnittelijoille. Sisäisesti ne kuitenkin jakavat koodia. Tarkoituksena on mahdollistaa kummankin rajapinnan tarjoamien palveluiden kehittäminen erillään, mutta jakaa ominaisuuksia, silloin kun se on mahdollista. Erilliset rajapinnat helpottavat myös suunnittelussa koska käyttäjäryhmillä on hyvin erilaiset tarpeet datan käsittelyssä ja näkemisessä (kuva 5).

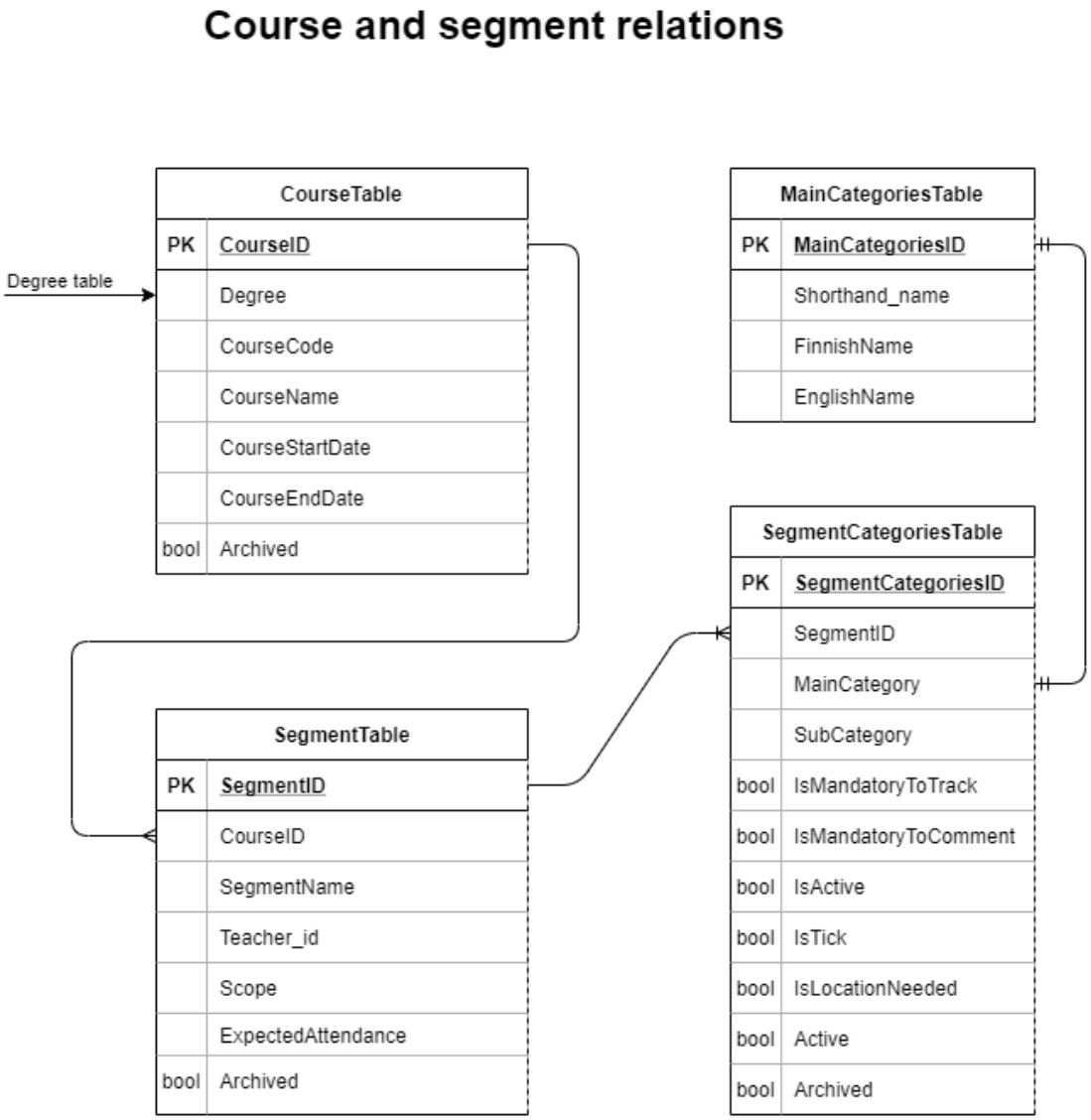


KUVA 5. Yksinkertaistettu kuva kurssin ja osion aktiivivaiheesta.

Tietokannan suunnittelussa oli kaksi suurempaa haastetta. Ensimmäisenä on, miten kursseista saataisiin oppilaitokselle hyödyllisempää tietoa kuin pelkästään kokonaistuntimäärät. Tämä ratkaistiin määräämällä, että jokaisella merkinnällä pitää olla kategoria ja jokaisella kategorialla pitää olla

pääkategoria. Näin voidaan luoda useamman kurssin ja osion sisältäviä tilastoja, ilman että rajoitetaan opettajan mahdollisuutta luoda omia alikategorioita, jotka vastaavat paremmin kurssin tarpeita. Pääkategorioiksi valittiin ”Lähiopetus”, ”Etäopetus” ja ”Itsenäinen työskentely”, niiden erilaisten resurssitarpeiden mukaan. Ne toimivat myös pohjana kolmelle alikategorialle, jotka ovat käytettävissä kaikissa osiossa. Tällä tavalla haluttiin varmistaa, että oppilas voi tehdä kaikkiin osioon merkintöjä ja seurata ajankäyttöään, vaikka sitä ei kyseisessä osiossa koettaisi opetuksen kannalta tarpeelliseksi.

Ehdottomasti suurin haaste tietokannan suunnittelussa oli, että kurssi voi sisältää useita opetuksellisesti erillisiä kokonaisuuksia ja jokaista näistä pitää pystyä seuraamaan itsenäisesti. Lopulta ratkaisuksi päädyttiin jakamaan kurssi kahteen osaan. Ensimmäinen osa on kurssi, joka sisältää tiedot, jotka ovat kaikille osioille samat. Sekä osioihin (segmentti), jotka sisältävät vain sitä osaa koskevat tiedot (kuva 6). Ratkaisu on joustava ja mahdollistaa sen, että kurssilla voi olla yksi tai enemmän osioita, ja niiden käytön ryhmäkohtaiseen jaotteluun.



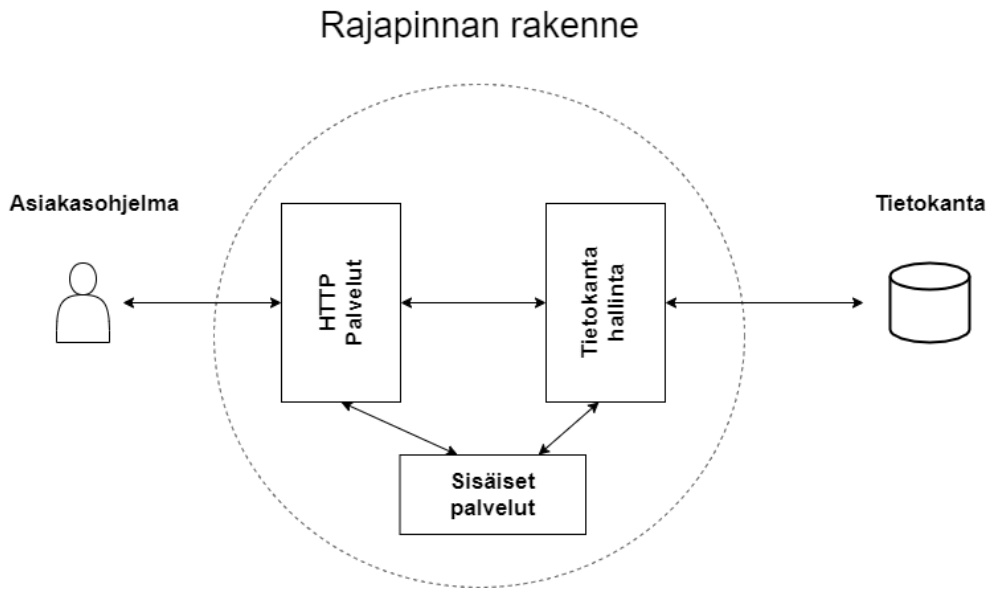
KUVA 6. Kurssi-, osio- ja kategoria -taulujen riippuvuussuhteet.

5 TOTEUTUS

Tämä opinnäytetyö aloitettiin miettimällä käyttäjätarinoita. Niiden avulla pystyttiin paremmin hahmottamaan toteutettava järjestelmä ja rajaamaan siitä pois toissijaisia ominaisuuksia. Alussa mietittiin myös laadulliset vaatimukset ja mitä ne käytännössä voisivat tarkoittaa. Koska aikaisempaa kokemusta käytetyistä teknologioista ja ohjelmistoista ei ollut, vaati jokainen vaihe myös paljon uuden opettelua. Kuitenkin pääsääntöisesti materiaalia, dokumentaatiota ja esimerkkejä löytyi hyvin ja lyhyen tutustumisvaiheen jälkeen pystyi jo siirtämään painotuksen kehitykseen ja opettelemaan lisää tilanteen niin vaatiessa.

5.1 Rajapinta

Koska työn tärkein osa on rajapinta, sen suunnitteluun ja toteutukseen käytettiin työssä paljon aikaa. Aluksi eteneminen tuntui hitaalta ja oli haasteellista tietää, mistä aloittaa. Usein toteutuksen aikana sai huomata, että se vaatii ensin muiden ominaisuuksien toteuttamista. Erityisen ongelmallinen oli kolmannen osapuolen GORM-moduuli, jonka heikko dokumentaatio ja rajoittunut käytäntö ilmoittaa virhetilanteista aiheutti viivästyksiä tarpeettoman paljon. Alun vaikeuksien jälkeen kuitenkin asiat helpottuivat. Koodia jaettiin mahdollisuuksien mukaan pieniin kokonaisuuksiin, tavoitteena tehdä siitä helpommin luettavaa ja parantaa uudelleen käytettävyyttä. Alkuperäisiä päätepisteitä jätettiin toteuttamatta, jos niiden tämänhetkinen toteutus ei ollut hyödyllistä, ja uusia luotiin tarpeen mukaan. Työssä toteutettiin yhteensä 48 päätepistettä, joista 22 opiskelija käyttäjille ja 26 opettajille ja ylläpidolle (liite 2). Sisäisesti rajapinta jaettiin kolmeen osaan. Yksi osa hoitaa asiakasohjelmien kanssa kommunikoinnin, toinen keskittyy tietokannan hallintaan ja kolmas pääasiassa tarjoamaan rajapinnan sisäisiä palveluja (kuva 7). Sisäinen tiedonsiirto osien välillä toteutettiin käyttämällä joko muuttujia tai tietorakenteita. Tällä haluttiin lisätä koodin muokattavuutta.



KUVA 7. Yksinkertaistettu rajapinnan rakenne.

5.2 Tietokanta

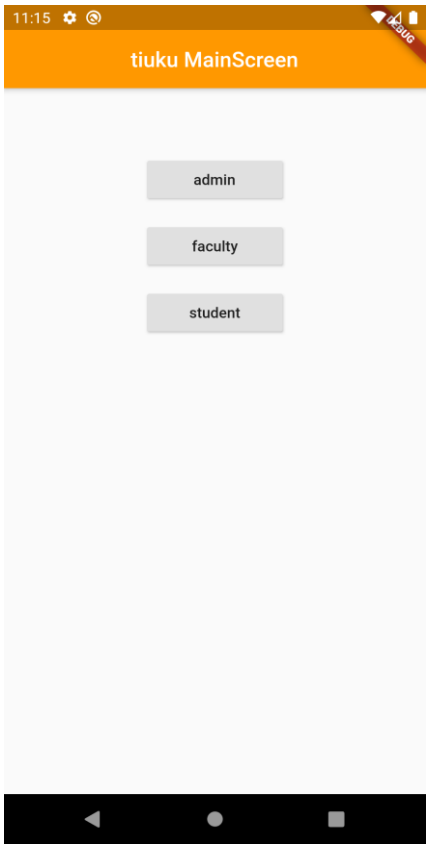
Tietokannan suunnittelu oli yksi ensimmäisistä toteutuksista, koska haluttiin hahmottaa, mitä tietoja järjestelmässä halutaan käsitellä. Tiedot pysyivät hyvin pitkälti suunnitelmien mukaisina, mutta taulujen määrä muuttui työn edetessä. Muun muassa huomattiin, että jossain tapauksissa yksi yhteinen taulu on paremmin hallittavissa kuin suuri määrä pieniä (liite 3). Näihin päätöksiin vaikutti, kenen käytettävissä taulun pitää olla ja kuinka usein sitä on tarvetta käsitellä. Koska käytettiin valmista Docker-konttia tarjoamaan tietokantapalvelua, ja siihen lukeminen ja kirjoittaminen oli GORM-moduulin vastuulla, vaati se hyvin vähän suoraa tietokannan käsittelyä.

Joitain ongelmia kuitenkin ilmaantui GORMin käytöstä. Sen ”assosisaatio”-ominaisuus vaikutti aluksi lupaavalta, mutta toiminta jäi käytännössä epäselväksi ja lopulta se poistettiin käytöstä. Ongelmia aiheutti myös, ettei GORM aina ilmoittanut mahdollisista virheistä selkeästi. Tietokantaa pitikin jatkuvasti seurata MySQL Workbench-ohjelman avulla, jotta pystyttiin näkemään GORMin tekemät muutokset tai niiden puutteet. Se ei myöskään hae tyhjiä tietueita, jotka valitettavasti olivat toiminnallisesti tärkeitä järjestelmässä, ja asia piti kiertää käyttämällä merkkijonoa merkitsemään tyhjää tietuetta.

Suuri poikkeama alkuperäiseen suunnitelmaan oli niin sanotun ”arkisto”-taulun lisääminen. Kun kurssi arkistoidaan, kopioidaan oppilaiden merkinnät anonyyminä tauluun ja se sisältää kaiken tarvittavan tiedon kattavien tilastojen luomiseen. Näin myös varmistetaan, ettei tietoa menetetä, vaikka alkuperäiset taulut poistetaan.

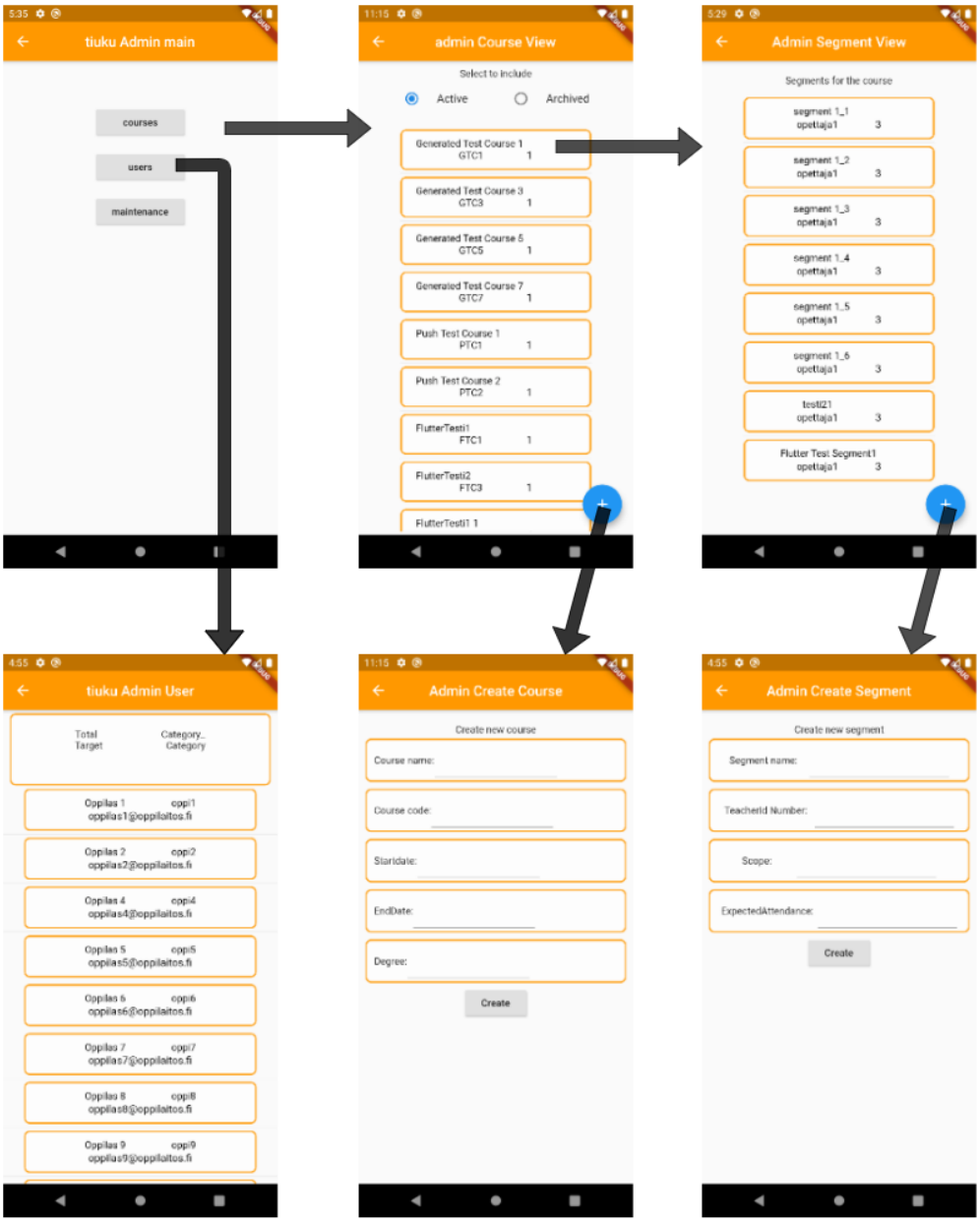
5.3 Asiakasohjelma

Järjestelmän ominaisuuksien esittelyyn tarkoitettu asiakasohjelma toteutettiin viimeisenä. Koska sen alustavaa suunnittelua oli tehty jo aikaisemmissa työvaiheissa, niin sen toteuttamiseen päästiin aika nopeasti. Koska aikaisempaa kokemusta Flutterista ei kuitenkaan ollut, vaati se opettelua niin ennen toteutuksen aloitusta kuin sen aikanakin. Työn painotus on kuitenkin rajapinnassa ja sen toteutuksessa, niinpä asiakasohjelmassa haluttiin esitellä vain joitain perusominaisuuksia ja ajatuksia, miten mahdolliset asiakasohjelmat voitaisiin toteuttaa. Asiakasohjelmassa esitellään kolmen erityyppisen käyttäjän tietonäkymät (kuva 8).



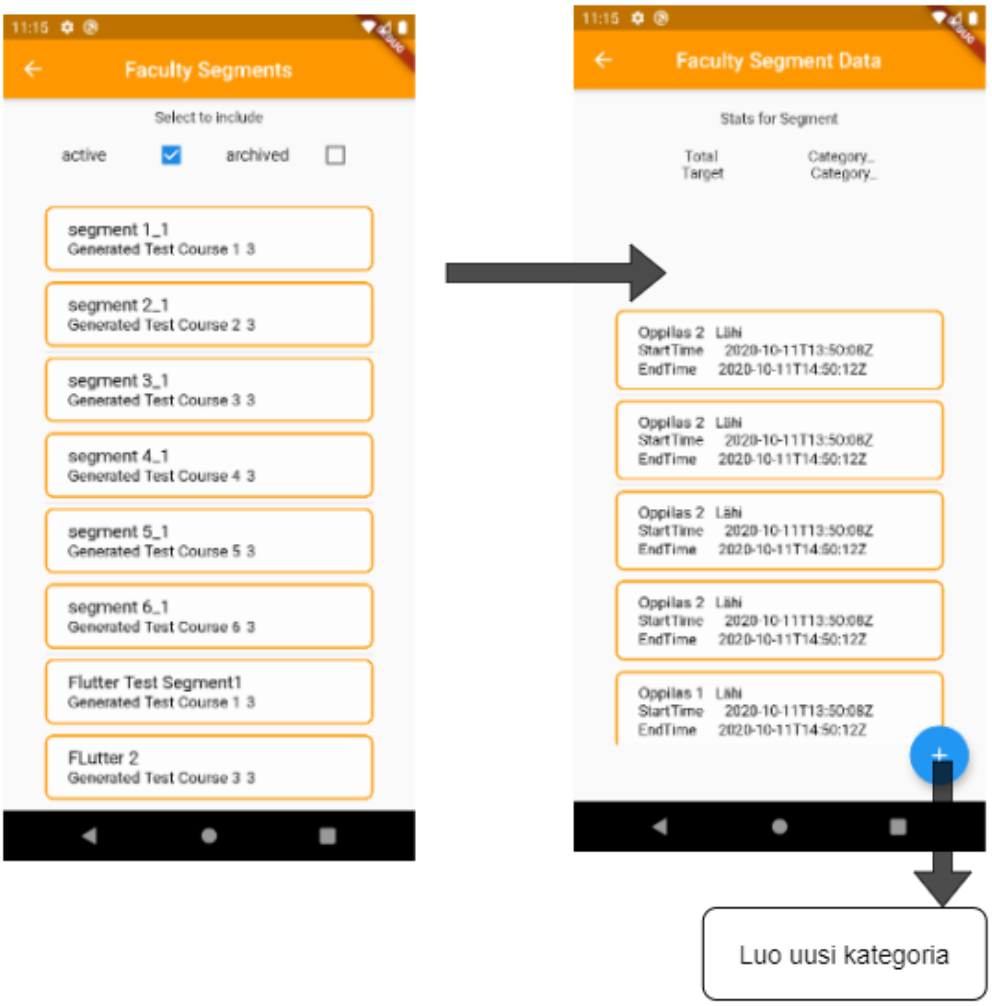
KUVA 8. Eri testikäyttäjät asiakasohjelmassa.

Ylläpitäjän näkymässä voi luoda kursseja ja osioita, sekä hallinnoida niitä (kuva 9).



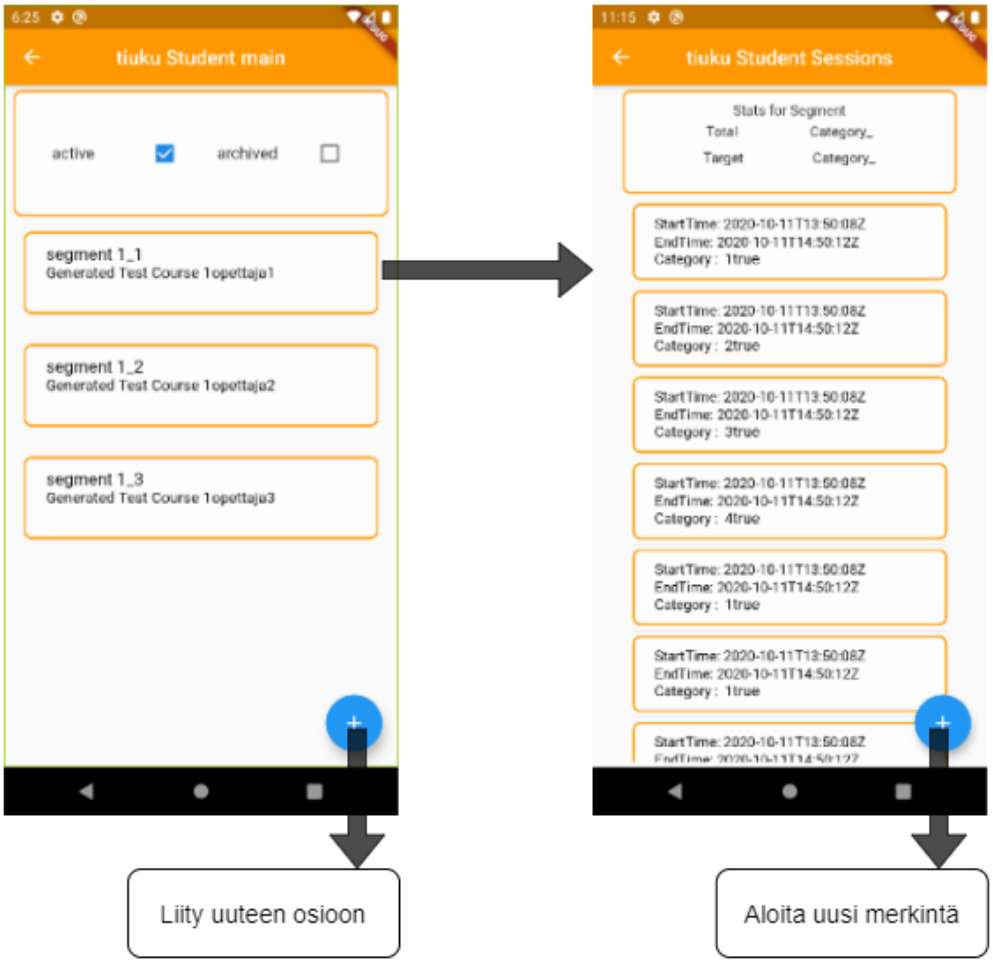
KUVA 9. Ylläpitäjän näkymiä asiakasohjelmassa.

Opettaja voi nähdä osiot, joille on merkitty opettajaksi, ja hallinnoida niitä (kuva 10).



KUVA 10. Opettajan näkymiä asiakasohjelmassa

Oppilas näkee osiot, joihin osallistuu ja niihin tekemänsä merkinnät (kuva 11).



KUVA 11. Oppilaan näkymiä asiakasohjelmassa.

6 YHTEENVETO

Jo opinnäytetyön aiheen valinnan aikaan oli hyvin selvää, että aiheen laajuuden vuoksi on tarpeen rajoittaa, mitä toteutetaan. Aika ja rajalliset resurssit, erityisesti aikaisemman kokemuksen ja osaamisen suhteen, määrittelivät toteutuksen pääpainon rajapinnan suunnitteluun ja toteutukseen. Ja rajoittaa tavoitteet siinä ominaisuuksiin, joita asiakasohjelman toiminta vaatii. Tässä tavoitteessa onnistuttiin, ja rajapinta saavutti pisteen, jossa sen jatkokehittäminen ilman käyttäjäpalautetta, ahkeraa systemaattista testaamista sekä kokeneempien kehittäjien opastusta olisi hidasta, jos ei mahdotonta.

Jatkokehityksessä tärkeimpiä ominaisuuksia olisi toteuttaa tilastot ja niiden automaattinen luonti. Tämän kehittäminen kuitenkin vaatii tilastomatemaattista osaamista ja kohderyhmien kanssa suunnittelua, että pystytään näyttämään heille oikeasti hyödyllistä tietoa. Lisäksi harkitsemisen arvoisia lisättäviä ominaisuuksia oli sijaintitiedot sekä arvosanat. Ensimmäisellä voitaisiin vaatia sijainnin tallentaminen esimerkiksi lähiopetus-kategoriassa. Koetaanko ominaisuus tarpeelliseksi ja voidaanko toteuttaa luotettavasti, ovat kysymyksiä, jotka vaativat tutkimusta. Arvosanojen merkitsemisellä osion ja kurssin lopuksi voisi kertoa, millainen suhde arvosanalla on osioon käytettyyn aikaan.

Kokonaisuutena opinnäytetyö oli hyvin opettavainen ja antoisa kokemus. Käytössä oli paljon nykyaikaisia ja uusia teknologioita ja työkaluja, joista on hyötyä tulevassa työelämässä. Työ antoi myös vahvaa kokemusta itsenäisestä työskentelystä, uuden opiskelusta, ongelmanratkaisusta sekä ajankäytöstä. Haasteena oli opinnäytetyön itsenäinen tekeminen, eikä ollut mahdollisuutta kysyä neuvoa tai mielipidettä, kun oli niihin tarvetta.

LÄHDELUETTELO

Chestnykh, D. 2015. Package uniuri. Viitattu 16.10.2020, https://github.com/dchest/uniuri.

Citrin, T. 2020. WSL 2 Support is coming to Windows 10 Versions 1903 and 1909. Microsoft Corporation. Viitattu 14. 9 2020, https://devblogs.microsoft.com/commandline/wsl-2-support-is-coming-to-windows-10-versions-1903-and-1909/.

diagrams.net 2020. diagrams.net. Viitattu 15.9.2020, https://www.diagrams.net/index.html.

Docker, Inc. 2020. Docker Desktop WSL 2 backend. Viitattu 14.9.2020, https://docs.docker.com/docker-for-windows/wsl/.

Docker, Inc. 2020. Install Docker Desktop on Windows Home. Viitattu 14.9.2020, https://docs.docker.com/docker-for-windows/install-windows-home/.

European Commission 2010. Mergers: Commission clears Oracle's proposed acquisition of Sun Microsystems. Viitattu 10.9.2020, https://ec.europa.eu/commission/presscorner/detail/en/IP_10_40.

Explore Group 2019. The Most Popular Databases 2019. Viitattu 10.9.2020, https://www.explore-group.com/blog/the-most-popular-databases-2019/bp46/.

Fielding, R. T. 2006. web.archive.org. Viitattu 10.9.2020, https://web.archive.org/web/20091111012314/http://tech.groups.yahoo.com/group/rest-discuss/message/6757.

Google, LLC. 2020. Flutter. Viitattu 8.11.2020, https://flutter.dev/.

Gorilla Toolkit 2020. Gorilla web toolkit. Viitattu 14. 9.2020, https://github.com/gorilla/mux.

Jinzhu 2020. The fantastic ORM library for Golang. Viitattu 14.9.2020, http://v1.gorm.io/.

Kincaid, J. 2009. Google's Go: A New Programming Language That's Python Meets C++. Viitattu 10.9.2020, https://techcrunch.com/2009/11/10/google-go-language/.

Loewen, C. 2020. WSL 2 will be generally available in Windows 10, version 2004. Viitattu 14.9.2020, https://devblogs.microsoft.com/commandline/wsl2-will-be-generally-available-in-windows-10-version-2004/.

Microsoft Corporation 2020. Visual Studio Code. Viitattu 14.9.2020, https://code.visualstudio.com/.

Microsoft Corporation 2020. Windows Subsystem for Linux Installation Guide for Windows 10. Viitattu 14.9.2020, https://docs.microsoft.com/en-us/windows/wsl/install-win10.

Nyyti Ry. 2020. AJANHALLINTA. Viitattu 10.9.2020, https://www.nyyti.fi/opiskelijoille/opi-elamantaitoa/hallittua-ajankayttoa/.

Oracle Corporation 2020. MySQL Workbench. Viitattu 4.11.2020, https://www.mysql.com/products/workbench/.

Postman, Inc. 2020. Postman. Viitattu 24.9.2020, https://www.postman.com/.

SmartBear Software 2020. Swagger Editor. Viitattu 14.9.2020, https://editor.swagger.io/.

Spotlight, Inc. 2020. Effortless API Design. Viitattu 14.9.2020, https://stoplight.io/studio/.

Statcounter 2020. Mobile Operating System Market Share Worldwide. Viitattu 16.9.2020, https://gs.statcounter.com/os-market-share/mobile/worldwide.

The Go Project 2020. golang.org. Viitattu 14.9.2020, https://golang.org/.

The Go Project 2020. Origins. Viitattu 10.9.2020, https://golang.org/doc/faq#Origins.

The Go Project 2020. Writing Web Applications. Viitattu 10.9.2020, https://golang.org/doc/articles/wiki/#tmp_3.

Tilastokeskus 2019. Suomalaisten internetin käyttö 2019. (Tilastokeskus) Viitattu 16.9.2020, https://www.stat.fi/til/sutivi/2019/sutivi_2019_2019-11-07_kat_001_fi.html.

Wikipedia Foundation 2020. Docker (software). Viitattu 24.11.2020, https://en.wikipedia.org/wiki/Docker_(software).

Wikipedia Foundation 2020. JSON. Viitattu 12.11.2020, https://en.wikipedia.org/wiki/JSON.

Wikipedia Foundation 2020. Representational state transfer. Viitattu 23.11.2020, https://en.wikipedia.org/wiki/Representational_state_transfer.

LIITTEET

Liite 1. Docker-compose.yml

Liite 2. Rest API Endpointit

Liite 3. Tietokannan taulut

DOCKER-COMPOSE.YML LIITE 1

```yaml
version: "3.8"

services:
    api:
        container_name: api
        ports:
        - "8080:8080"
        depends_on:
            - "db"
        working_dir: /tiuku
        volumes:
            - /c/Oamk/Oppari/tiuku:/tiuku
        build: .
        #command: go run main.go
        command: tail -fn0 /dev/null

    db:
        container_name: db
        image: mysql:8
        ports:
        - "3306"
        - "3306:3306"
        environment:
        - MYSQL_ROOT_PASSWORD=tiukusql
        - MYSQL_USER=testiuser
        - MYSQL_PASSWORD=testipassword
        - MYSQL_DATABASE=tiukuDB
        volumes:
        - /c/Oamk/Oppari/tiuku-mysql-data:/var/lib/mysql

volumes:
    tiuku-mysql-data:
    tiuku:
```

// Student v1 RAW Routes

// STUDENT DELETEs

- Poista osallistuminen {Segment}:lle
  "/students/v1/courses/{course}/segments/{segment}",
- Poista {Session} - merkintä.
  "/students/v1/segments/{segment}/sessions/{session}",

// STUDENT GETs

- Saa kaikki kertyneet käynnissä olevien segmenttien Sessionit.
  "/students/v1/sessions",
- Saa viimeisin Sessioni.
  "/students/v1/sessions/last",
- Saa lista kouluista
  "/students/v1/schools",
- Saa lista kampuksista
  "/students/v1/campuses",
- Saa lista yksiköistä
  "/students/v1/apartments",
- Saa lista tutkinnoista
  "/students/v1/degrees",
- Saa kaikkien henkilökunta käyttäjien tiedot
  "/students/v1/faculty"
- Saa {faculty} henkilökuntakäyttäjän tiedot
  "/students/v1/faculty/{faculty}"
- Saa lista aktiivista kursseista
  "/students/v1/courses",
- Saa {course} kurssin tiedot
  "/students/v1/courses/{course}",
- Saa {course} kurssin segmentit
  "/students/v1/courses/{course}/segments",
- Saa {course} kurssin {segmentti}

"/students/v1/courses/{course}/segments/{segment}",

- Saa {course} kurssin {segmentin} kategoriat

"/students/v1/courses/{course}/segments/{segment}/categories",

- Saa {segmentin} sessionit

"/students/v1/segments/{segment}/sessions",

- Saa {segmentin} {session}

"/students/v1/segments/{segment}/sessions/{session}",

- Saa segmentit joihin olet osallistunut, oletuksena näyttää aktiiviset

parametreillä archived=yes, näyttää myös arkistoidut. archived=only vain arkistoidut.

"/students/v1/segments",

// PATCHs STUDENT

- Lopeta Segment

"/students/v1/segments/{segment}/sessions/{session}",

// STUDENT POSTs

- Liity {course} kurssin {segmenttiin}

"/students/v1/courses/{course}/segments/{segment}",

- Aloita uusi session tai lähetä valmiin tiedot

"/students/v1/segments/{segment}/sessions",

// STUDENTS PUTs

- Korvaa/päivitä {segmentin} {session} uudella datalla.

"/students/v1/segments/{segment}/sessions/{session}",

// FACULTY v1 RAW Routes

// FACULTY DELETEs

- Poista opiskelija käyttäjä

"/faculty/v1/students/{student}",

// FACULTY GETs

- Saa oppilas-lista

"/faculty/v1/students",

- Saa {student} oppilaan tiedot

"/faculty/v1/students/{student}",

- Saa henkilöstökäyttäjä-lista

"/faculty/v1/faculty",

- Saa {faculty} henkilökuntakäyttäjän tiedot

"/faculty/v1/faculty/{faculty}"

- Saa koulut

"/faculty/v1/schools",

- Saa kampuksessa

"/faculty/v1/campuses",

- Saa yksiköt

"/faculty/v1/apartments",

- Saa tutkinto

"/faculty/v1/degrees",

- Saa henkilökuntakäyttäjän aktiiviset segmentit, oletuksena aktiiviset,

parametreillä archived=yes, myös arkistoidut

archived=only, vain arkistoidut

"/faculty/v1/segments",

- Saa kurssit, oletuksena aktiiviset, parametreillä archived=yes, myös arkistoidut

archived=only, vain arkistoidut

"/faculty/v1/courses",

- Saa {course} kurssin tiedot

"/faculty/v1/courses/{course}",

- Saa {course} kurssin segmentit

"/faculty/v1/courses/{course}/segments",

- Saa {course} kurssin {segmentin} tiedot

"/faculty/v1/courses/{course}/segments/{segment}",

- Saa {course} kurssin {segment}:lle ilmoittautuneet opiskelijat

"/faculty/v1/courses/{course}/segments/{segment}/students",

- Saa {course} kurssin {segment}:lle kirjatut sessionit

"/faculty/v1/courses/{course}/segments/{segment}/sessions",

- Saa {course} kurssin {segment}:n {kategoria}:n tiedot

"/faculty/v1/courses/{course}/segments/{segment}/categories/{category}",

- Saa {course} kurssin {segment}:n kategoriat

"/faculty/v1/courses/{course}/segments/{segment}/categories",

// FACULTY PATCHs

- Päivitä opiskelija käyttäjän tietoja.

  "/faculty/v1/students/{student}",

- Päivitä henkilökuntakäyttäjän tietoja.

  "/faculty/v1/faculty/{faculty}",

- Päivitä {course} kurssin tietoja, arhived=true arkistoi kurssin.

  "/faculty/v1/courses/{course}",

// FACULTY POSTs

- Lisää uusi opiskelija

  "/faculty/v1/students",

- Lisää uusi henkilökuntakäyttäjä

  "/faculty/v1/faculty",

- Lisää uusi kurssit

  "/faculty/v1/courses",

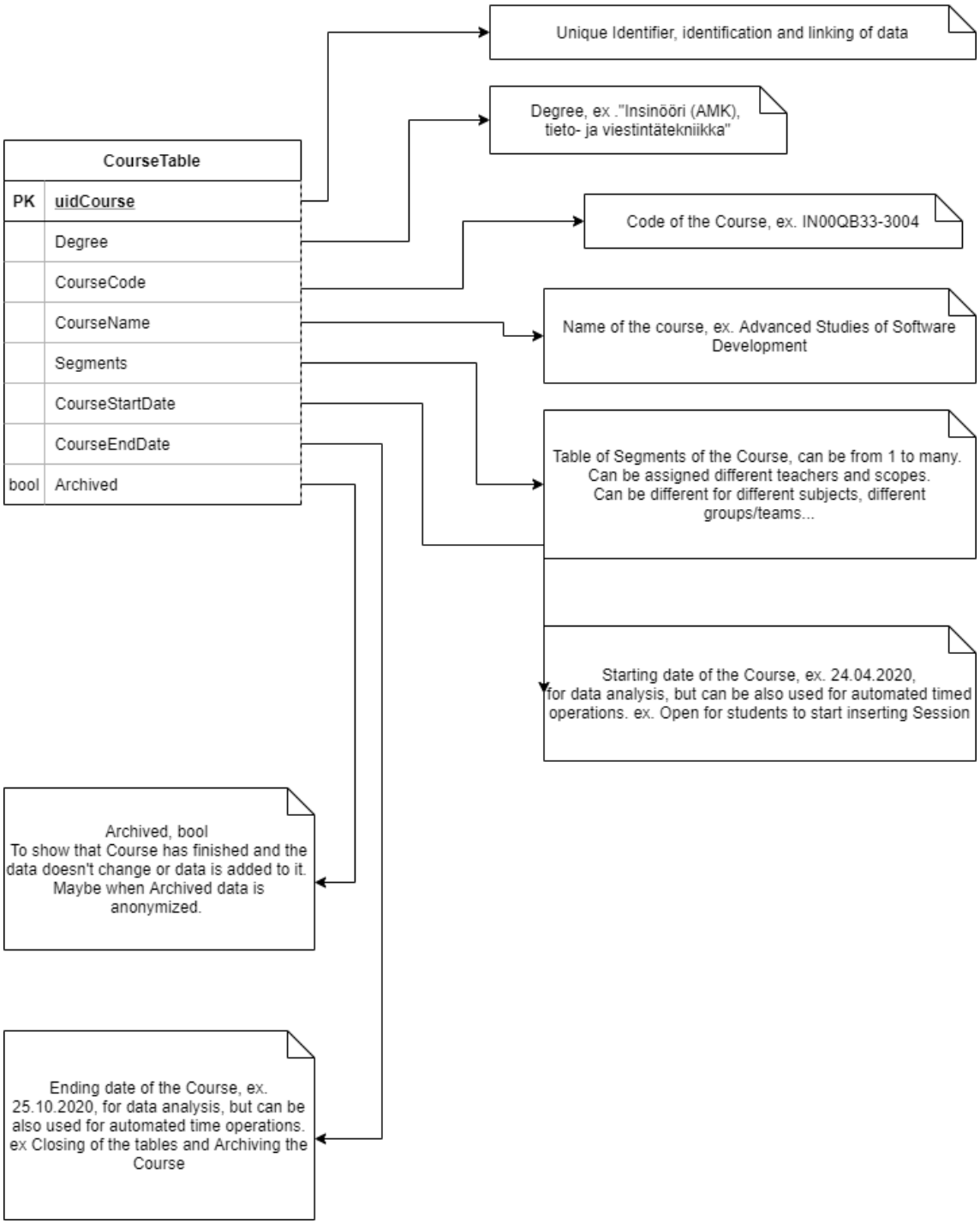
- Lisää uusi segmentti {course} kurssiin

  "/faculty/v1/courses/{course}/segments",

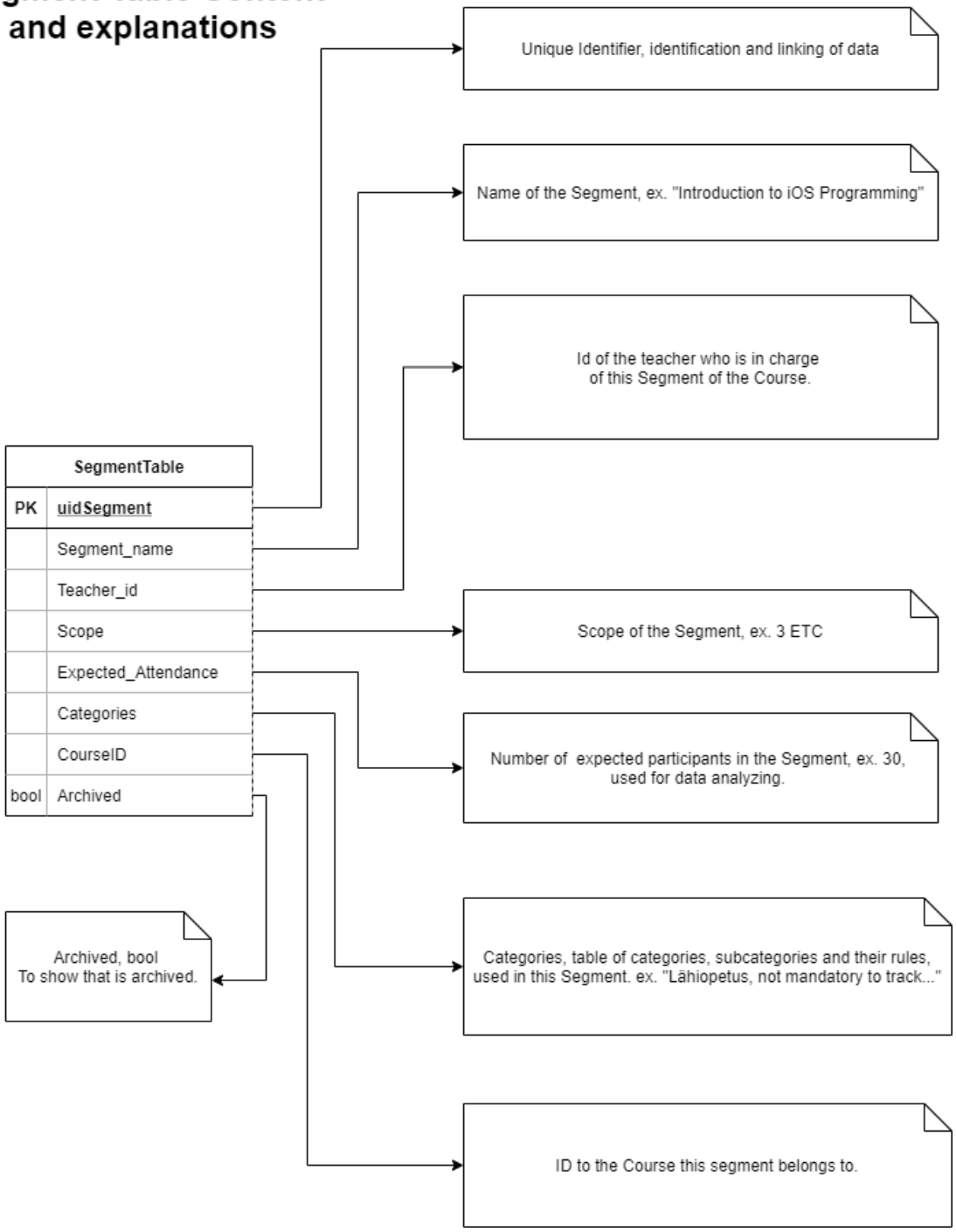
- Lisää uusi kategoria {course} kurssin {segmenttiin}

  "/faculty/v1/courses/{course}/segments/{segment}/categories",

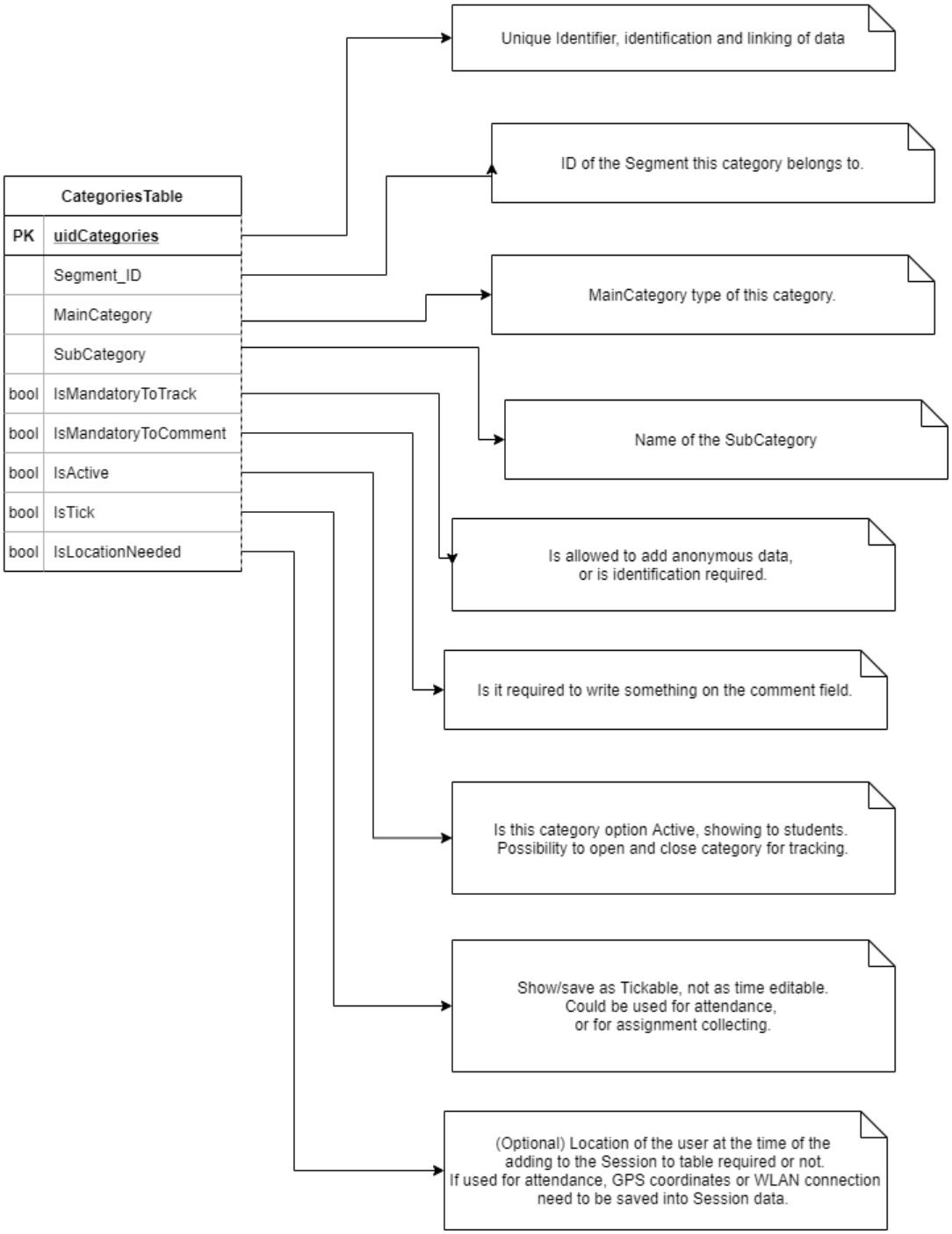
CourseTable Content and explanation

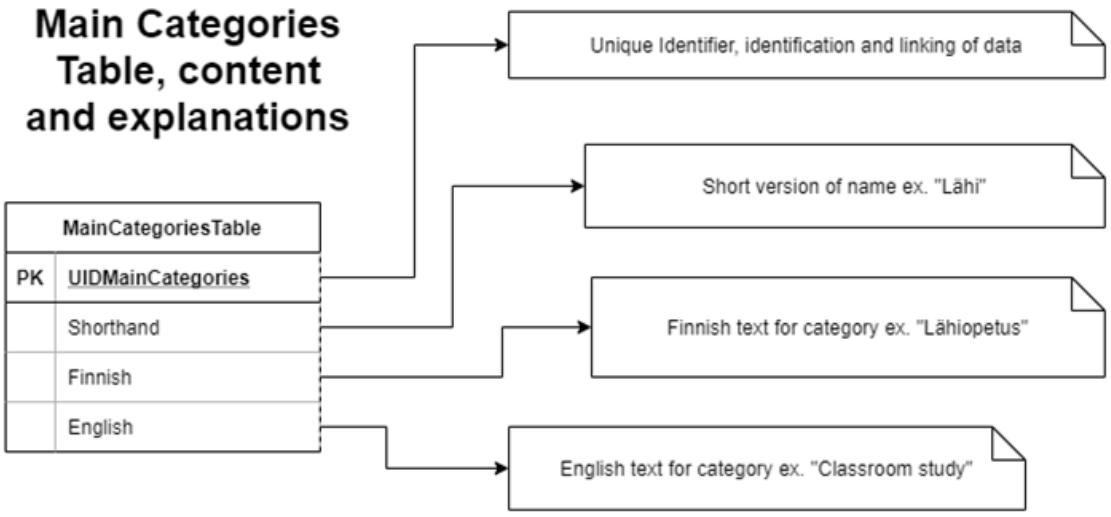


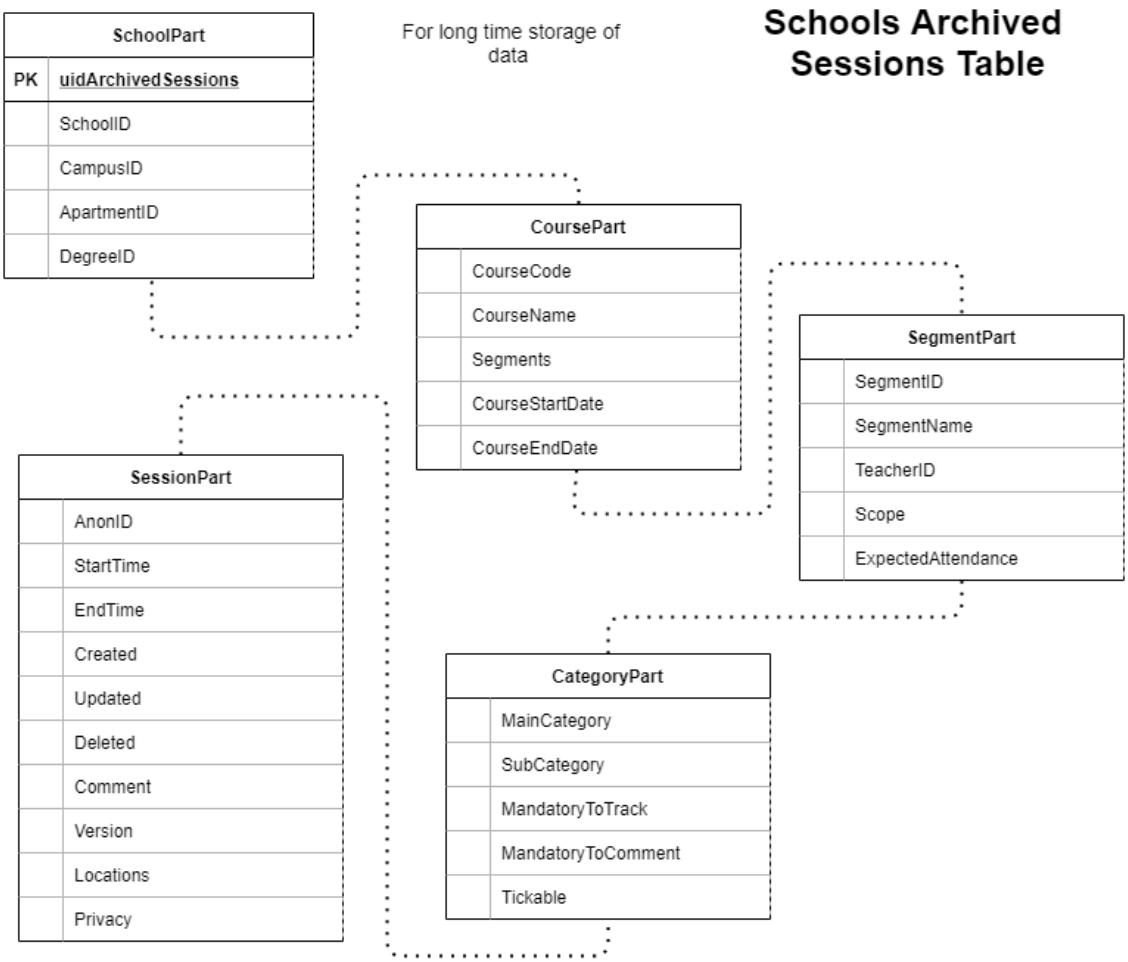
Segment Table Content and explanations



CategoriesTable, content and explanations

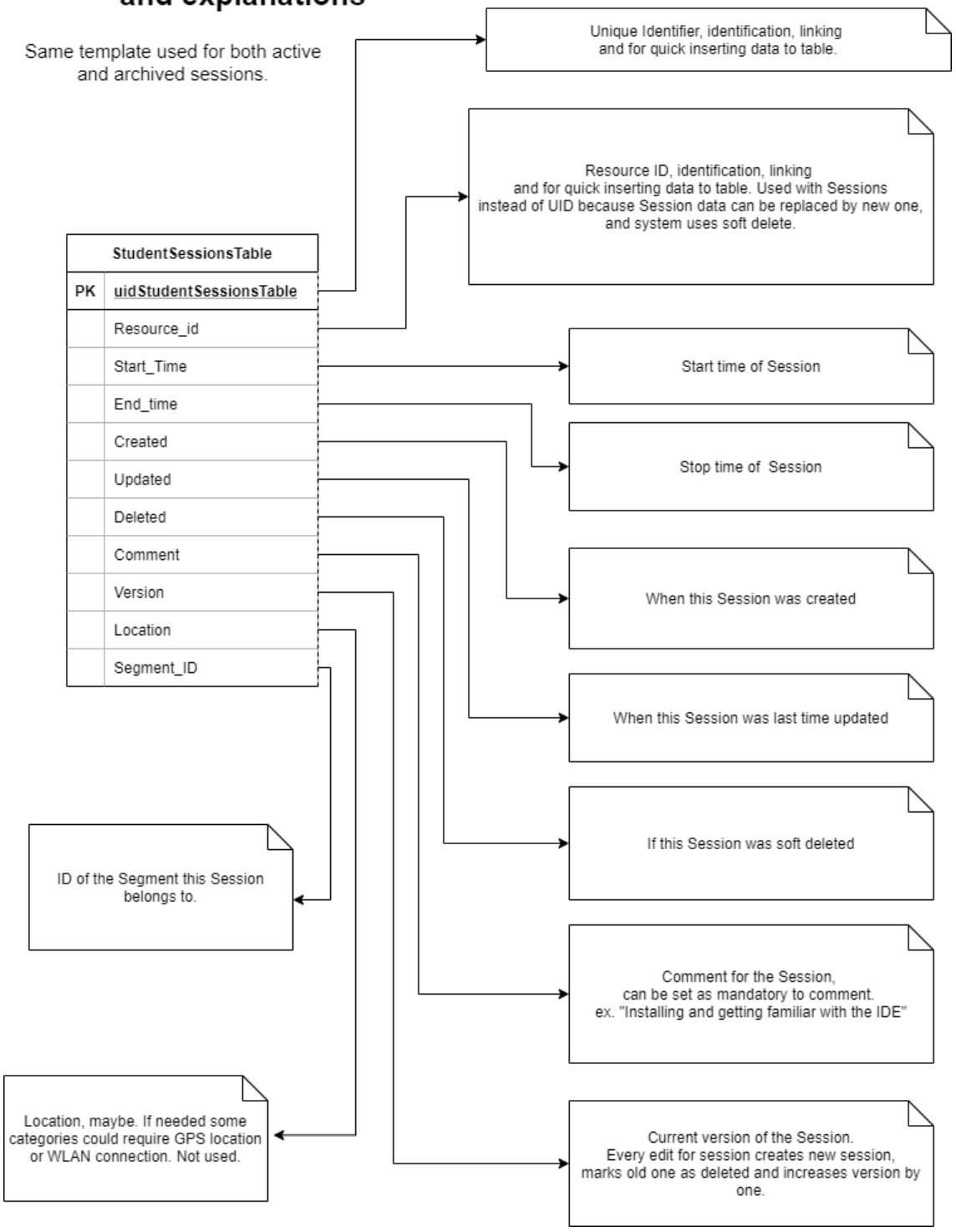



Main Categories Table, content and explanations
Unique Identifier, identification and linking of data
Short version of name ex. "Lähi"
Finnish text for category ex. "Lähiopetus"
English text for category ex. "Classroom study"
MainCategoriesTable
PK
UIDMainCategories
Shorthand
Finnish
English


Schools Archived Sessions Table
For long time storage of data
SchoolPart
PK
uidArchivedSessions
SchoolID
CampusID
ApartmentID
DegreeID
CoursePart
CourseCode
CourseName
Segments
CourseStartDate
CourseEndDate
SegmentPart
SegmentID
SegmentName
TeacherID
Scope
ExpectedAttendance
SessionPart
AnonID
StartTime
EndTime
Created
Updated
Deleted
Comment
Version
Locations
Privacy
CategoryPart
MainCategory
SubCategory
MandatoryToTrack
MandatoryToComment
Tickable

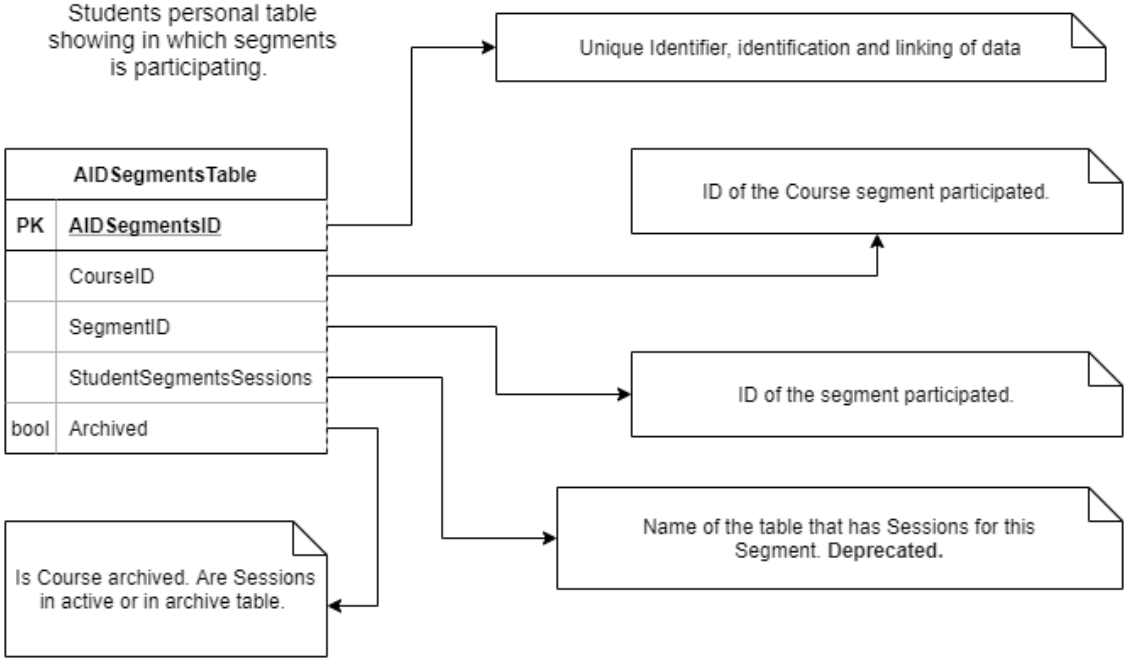
StudentSessionsTable content and explanations

Same template used for both active and archived sessions.



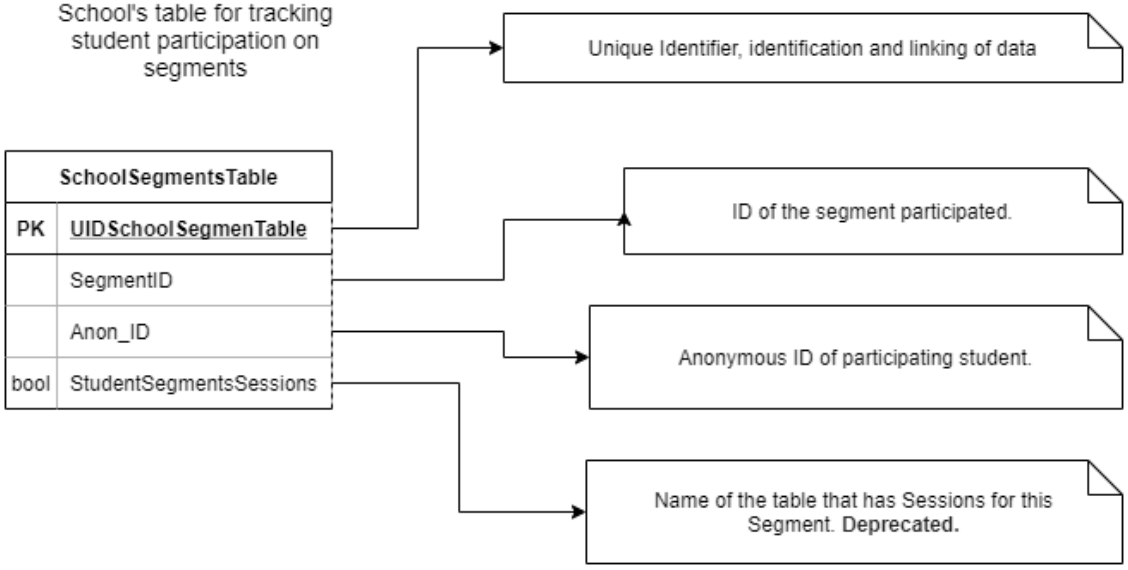
Students Segments Table, content and explanations

Students personal table showing in which segments is participating.

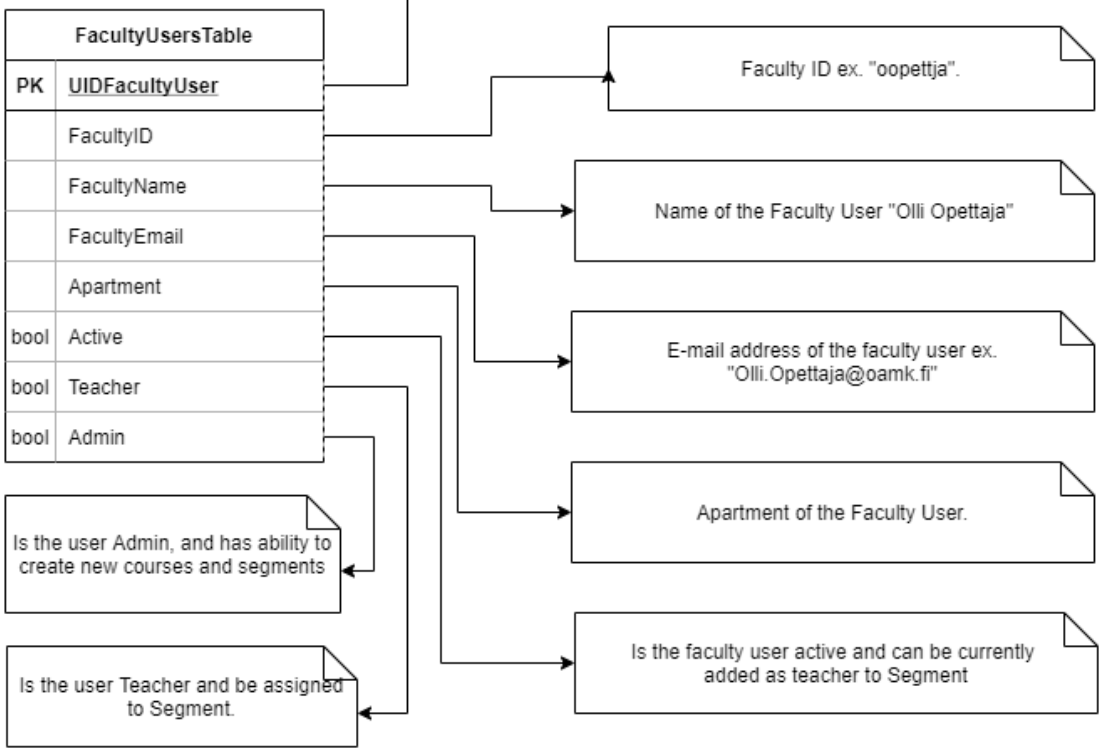


School Segments Table, content and explanations

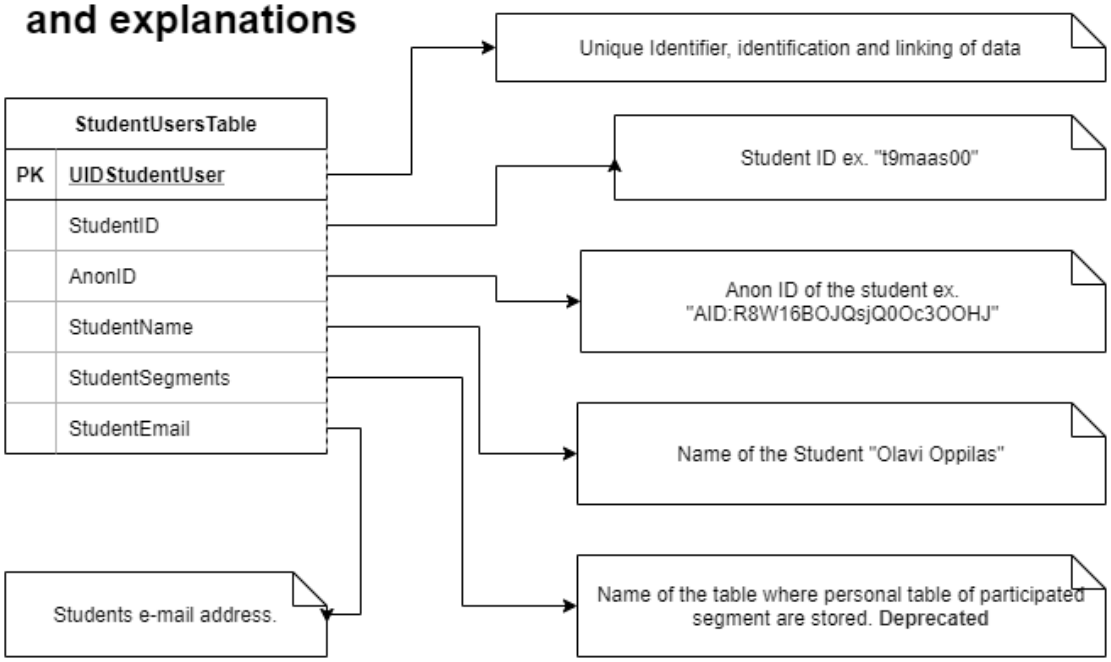
School's table for tracking student participation on segments

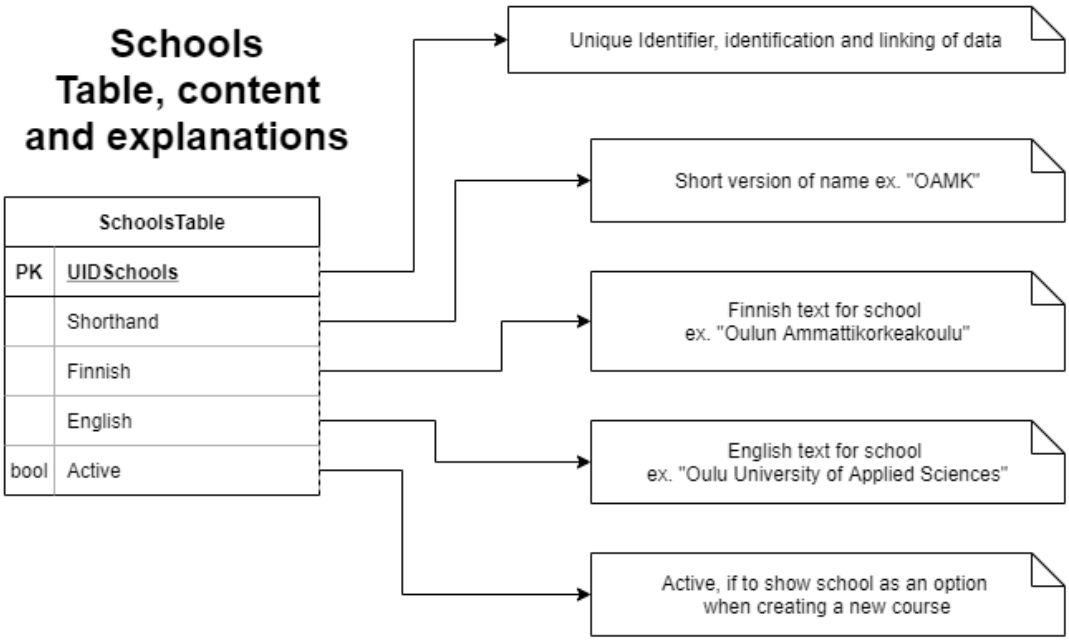


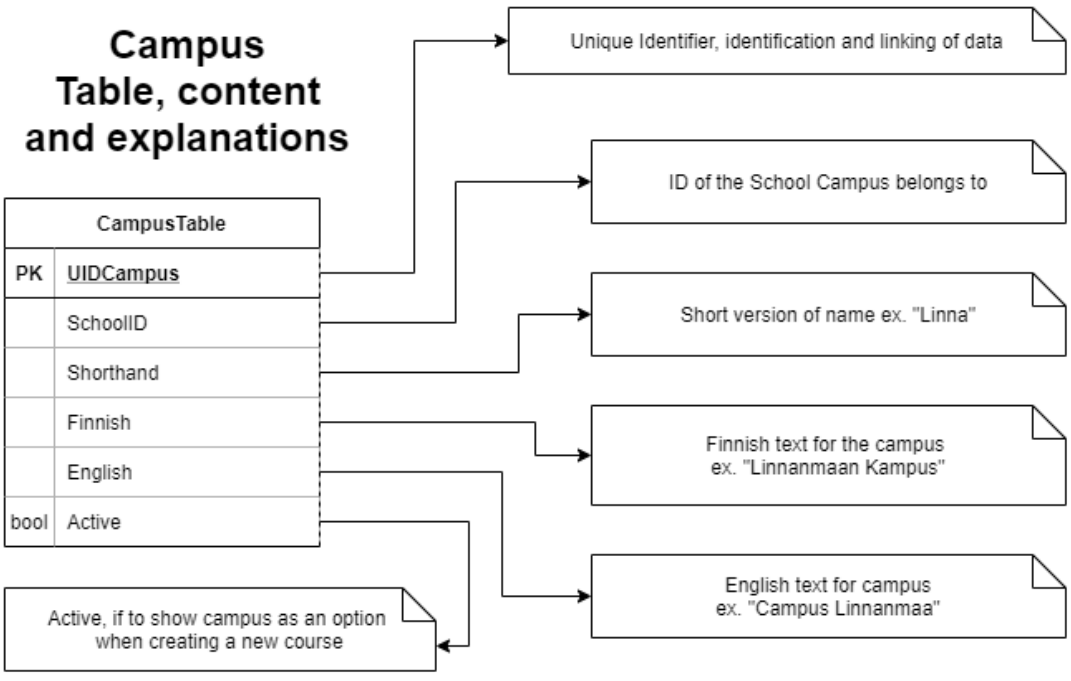
Faculty User Table, content and explanations

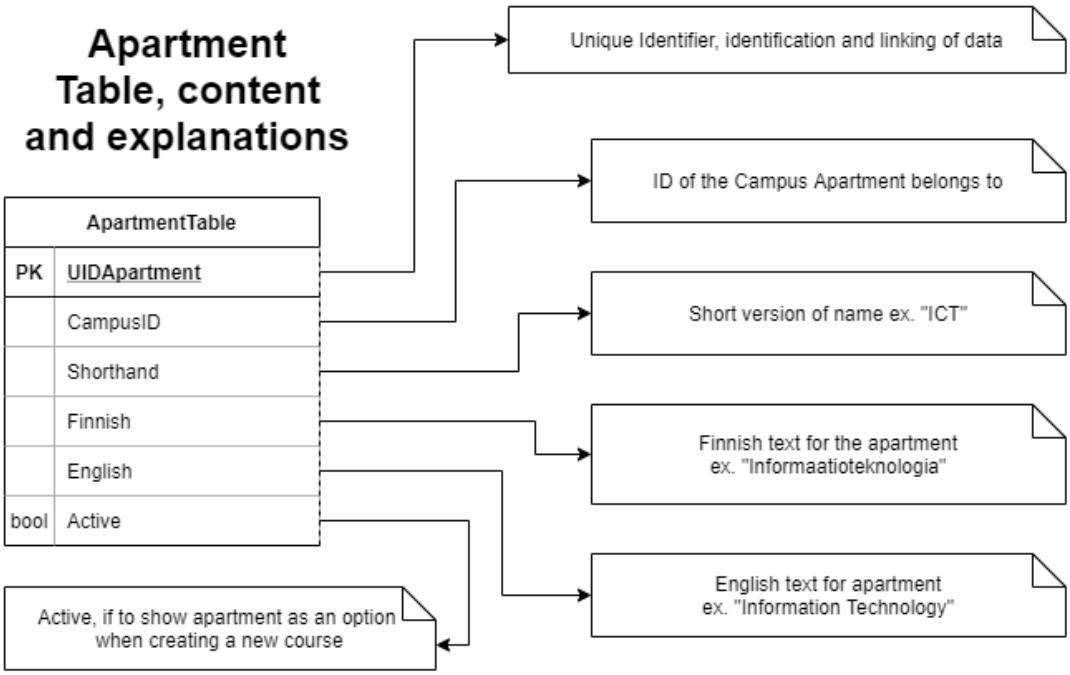


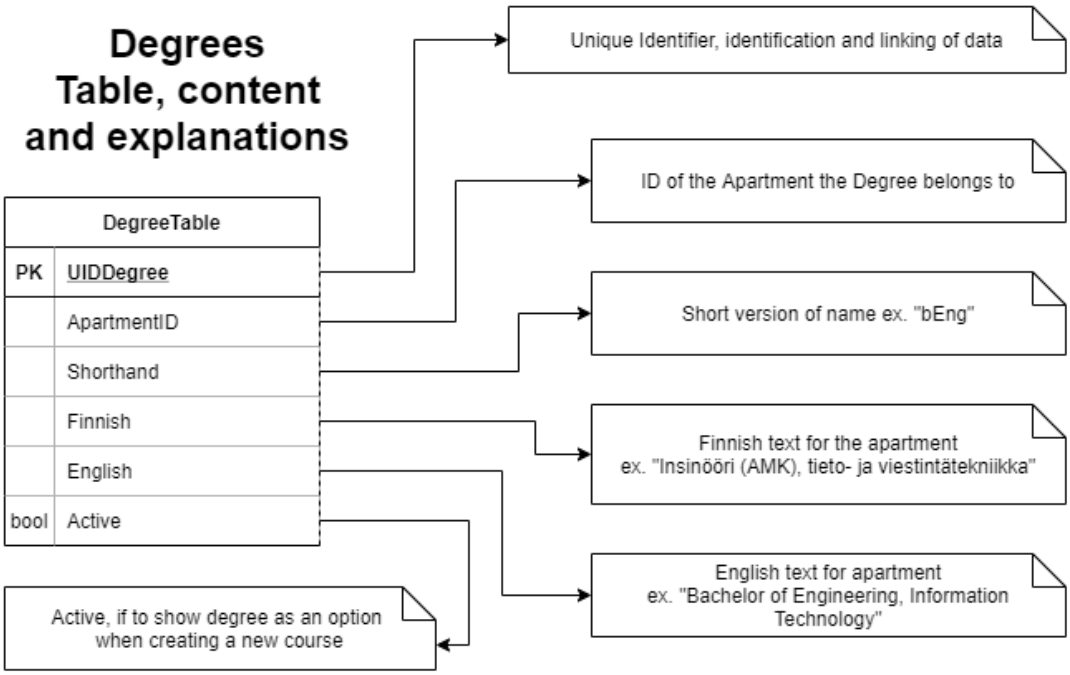
Student User Table, content and explanations




Schools
Table, content
and explanations
SchoolsTable
PK
UIDSchools
Shorthand
Finnish
English
bool
Active
Unique Identifier, identification and linking of data
Short version of name ex. "OAMK"
Finnish text for school
ex. "Oulun Ammattikorkeakoulu"
English text for school
ex. "Oulu University of Applied Sciences"
Active, if to show school as an option
when creating a new course


Campus
Table, content
and explanations
CampusTable
PK
UIDCampus
SchoolID
Shorthand
Finnish
English
bool
Active
Active, if to show campus as an option
when creating a new course
Unique Identifier, identification and linking of data
ID of the School Campus belongs to
Short version of name ex. "Linna"
Finnish text for the campus
ex. "Linnanmaan Kampus"
English text for campus
ex. "Campus Linnanmaa"


Apartment
Table, content
and explanations
ApartmentTable
PK
UIDApartment
CampusID
Shorthand
Finnish
English
bool
Active
Unique Identifier, identification and linking of data
ID of the Campus Apartment belongs to
Short version of name ex. "ICT"
Finnish text for the apartment
ex. "Informaatioteknologia"
English text for apartment
ex. "Information Technology"
Active, if to show apartment as an option
when creating a new course


Degrees
Table, content
and explanations
DegreeTable
PK
UIDDegree
ApartmentID
Shorthand
Finnish
English
bool
Active
Unique Identifier, identification and linking of data
ID of the Apartment the Degree belongs to
Short version of name ex. "bEng"
Finnish text for the apartment
ex. "Insinööri (AMK), tieto- ja viestintätekniikka"
English text for apartment
ex. "Bachelor of Engineering, Information
Technology"
Active, if to show degree as an option
when creating a new course