Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Huy Nguyen

Python Flask CRUD Web App

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja Viestintätekniikka

Insinöörityö

11.11.2020

Tekijä Otsikko Sivumäärä Aika	Huy Nguyen Python Flask CRUD Web App 22 sivua 11.11.2020
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja Viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Lehtori Jorma Räty

Insinöörityön tarkoituksena oli kehittää Airbus Defence and Space Oy:n tutkimus- ja tuotekehityksen osaston laboratoriotiimin käyttöön verkkosovellus, jolla voitaisiin ylläpitää ja hallinnoida osaston laitteiston tietoja. Sovelluksen tuli olla yksinkertainen ja helppo käyttää, jotta se voisi korvata nykyisen järjestelmän.

Verkkosovellus toteutettiin käyttämällä Python 3.8 -ohjelmointikieltä ja Flask-mikrosovelluskehystä, jotta saataisiin luotua yksinkertainen ja monipuolinen CRUD-verkkosovellus. Sovelluksen kanssa käytettiin MariaDB SQL -relaatiotietokantaa. Sovelluksen käyttöliittymä luotiin Flaskia ja Bootstrap-kirjastoa hyödyntäen.

Ohjelmistontuotantoon kuului alussa suunnitteluvaihe, jossa käytiin tiimin jäsenien ja esimiehen kanssa keskustelua siitä, mitä vaatimuksia verkkosovellukselle on. Kehitysvaihe oli erittäin itsenäistä työskentelyä, ja luotu verkkosovellus tuli vaiheittain laboratoriotiimin käyttöön. Sovelluksen kehitys perustuu käyttäjäpalautteen ja yhteisen ideoinnin perusteella.

Lopputulokseksi kehitetty verkkosovellus on otettu käyttöön laboratoriotiimissä. Verkkosovellus on osoittautunut toimivaksi ja tehokkaaksi työkaluksi, joka helpottaa tietojen hallinnoimista.

Avainsanat	Python, Flask, CRUD, Verkkosovellus, SQL

Author	Huy Nguyen
Title	Python Flask CRUD Web App
Number of Pages	22 pages
Date	11 November 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Jorma Räty, Senior Lecturer

The aim of the thesis was to develop a web application for a laboratory team of Airbus Defence and Space Oy's Research & Development departments, which could be used to maintain and manage the department's hardware data. The application had to be simple and easy to use to replace the current system.

The web application was implemented using the Python 3.8 programming language and the Flask micro-framework to create a simple and versatile CRUD web application. The MariaDB SQL relational database was used with the application. The application interface was created using Flask and the Bootstrap library.

Software production initially included a design phase where a discussion was held with the team members and a line manager about what the requirements are for the web application. The development phase was mostly independent working, and the created web application was gradually introduced to the laboratory team. The development of the application is based on user feedback and ideas provided by the team members.

The web application developed has been implemented in the laboratory team. It has proven to be a functional and efficient tool for data management.

Keywords	Python, Flask, CRUD, Web Application, SQL

Sisällys

Lyhenteet

CRUD	Create, read, update, delete. Tietokannan neljä perustoiminnallisuutta.
HTML	Hypertext Markup Language, avoimen standardin kuvauskieli, jolla voidaan merkata hypertekstiä.
CSS	Cascade Style Sheets, HTML-merkintäkielen tyylitysohje.
WSGI	Web Server Management System, Pythonin ja verkkopalvelimen välinen standardiväylä.
SSH	Secure Shell, salatun tietoliikenteen protokolla.
REST	Representational State Transfer, ohjelmistoarkkitehtuuri, jota käytetään www-sovelluspalvelun käyttöönotossa.
JSON	JavaScript Object Notation, tiedostomuoto, joka perustuu avoimeen standardiin.
IP	Internet Protocol address, verkkosovittimien yksilöimiseen tarkoitettu protokollaosoite.
URL	Uniform Resource Locator, tietyn tiedon sijainnin määrittelevä merkkijono.
SQL	Structured Query Language, kyselykieli relaatiotietokannassa.
MML	Man Machine Language, määrittelykieli konsolin kautta hallinnoimaan telelekommunikaatio- tai verkkolaitteita.
DNS	Domain Name System, internetin nimipalvelujärjestelmä, joka muuttaa domainin eli verkkotunnuksen IP-osoitteeksi.

1 Johdanto

Opinnäytetyö tehtiin Airbus Defence and Space Oy:n tutkimus- ja tuotekehitysosaston laboratoriotiimille. Laboratoriotiimi vastaa osaston laitteiston ohjelmistotasoista ja mahdollisesta tuotekehityksestä. Tarkoituksena oli kehittää verkkosovellus, jossa olisi SQL-relaatiotietokannan perustoiminnallisuudet luoda, lukea, muuttaa ja poistaa tietoja tietokannasta.

Valitsimme projektin kehitykseen Python-ohjelmointikielen osaamiseni perusteella. Tutkiessani vaihtoehtoja totesin, että Python-ohjelmointikieli tarjoaa erittäin laajan valikoiman erilaisia sovelluskehyksiä, joita voitaisiin hyödyntää projektissamme. Päätimme myös, että minun olisi parempi vahvistaa Python-osaamistani ja katsoa muita sovelluskehyksiä myöhemmin.

Verkkosovelluksen backend-toiminto luotiin Pythonin ja Flask-mikrosovelluskehyksen avulla. Flask on kevyt ja yksinkertainen sovelluskehys, joka on saanut GitHubissa toiseksi eniten tähtiä Pythonin verkkosovellukseen käytetyistä sovelluskehyksistä. [1.] Flask on myös äänestetty vuonna 2018 verkkosovelluskehittäjien suosituimmaksi sovelluskehykseksi Djangon lisäksi. [2.]

Opinnäytetyössä keskityn eri teknologioiden ominaisuuksiin ja niiden käyttöön projektissa. Raportissa kerrotaan projektin työn kulusta ja työn saavutuksista Python-ohjelmointikieltä ja Flask-sovelluskehystä käyttäen.

2 Lähtökohta

2.1 Nykytilanne ja ongelma

Yrityksemme työskentelee eri valtioiden ja yksityisten järjestöjen kanssa, esimerkiksi hätäkeskusten ja valtion laitosten kanssa. Airbus Defence and Space on toimittanut koko maan kattavia viranomaisverkkoja maailmanlaajuisesti yli 70 maahan. Suurimpiin asiakkaisiin kuuluvat muun muassa Suomi, Ruotsi, Saksa, Belgia, Viro ja Unkari. [3.] Työskentelen yrityksemme tutkimus- ja tuotekehitysosaston laboratoriotiimissä, joka on vastuussa osastomme laitteistosta. Tiimimme työskentelee laitteiden ja niiden päivityksien parissa, joten on tärkeää pitää lukua laitteistomme ja ohjelmistomme versioista ja muista oleellisista tiedoista kuten IP-osoitteista, laitteiden nimistä ja versioista. Meille tulee usein muilta tiimeiltä laitteiden ohjelman päivityspyyntöjä, jolloin tarvitsemme yllä mainitut tiedot, jotta pääsemme manuaalisesti päivittämään laitteita.

Tällä hetkellä hallisemme tietoja monilla eri tavoilla, kuten Excel-taulukoilla ja PowerPoint-dioilla. Excel-taulukoilla pidämme lukua verkkojen nimistä ja IP-osoitteista, PowerPoint-dioilla taas suunnittelemme verkkokuviamme, joissa näkyvät laitteet ja niiden IP-osoitteet. Kaikista taulukoista ja dioista on monta eri versiota versiohallinnan takia. Tämän seurauksena on ollut välillä vaikeaa etsiä ajan tasalla olevaa tietoa. Esimerkiksi itselläni on ollut paljon vaikeuksia etsiä oikeaa tietoa nopeasti, sillä joudun käymään välillä todella monta tiedostoa läpi ennen kuin löydän oikeat tiedot.

2.2 Työn tavoitteet

Tiimissämme on aiemmin kokeiltu luoda Ansible-skriptausta käyttäen taulukkomainen sivusto, joka on staattinen, eli taulukon tiedot eivät ole muokattavissa. Tämä sivusto ei ollut ollenkaan käytännöllinen, sillä sitä oli vaikea ylläpitää, eikä se ollut skaalautuva.

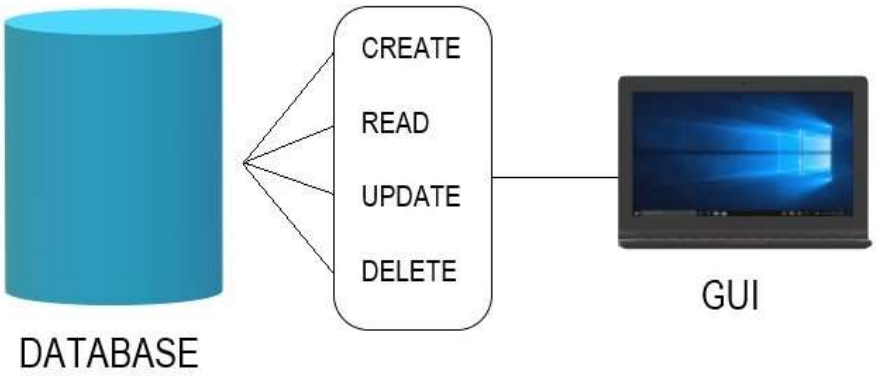
Totesimme, että aiemmat kokeillut versiot ja versiohallinnat olivat epäkäytännöllisiä, joten aloitimme projektini suunnittelun. Tavoitteena oli luoda verkkosovellus, josta löytyy

CRUD-toiminnot, eli että käyttäjä pystyy luomaan, lukemaan, päivittämään ja poistamaan tietoja tietokannasta käyttöliittymän kautta ja mahdollisesti ajamaan skriptejä taustalla.

3 Käytetyt teknologiat

3.1 CRUD-toiminnot

CRUD-termi tulee englanninkielisistä sanoista Create, Read, Update ja Delete, jotka ovat relaatiotietokannan neljä perustoimintoa.



Kuva 1. Tietokannan CRUD-toiminto.

Toiminnot	SQL	RESTful WS
Create	INSERT	POST
Read	SELECT	GET
Update	UPDATE	PUT
Delete	DELETE	DELETE

Kuva 2. CRUD-toiminnot SQL-kielessä ja REST-protokollassa.

Create-toiminto on yksinkertaisin ja se mahdollistaa uuden merkinnän luomisen tai lisäämisen tietokantaan. Read-toiminto on hieman laajempi, sen avulla käyttäjä voi hakea tai lukea tietoa tietokannasta. Read-toiminnolla voidaan esimerkiksi luoda hakukenttä, joka hakee tietokannasta juuri sillä syötteellä, jonka käyttäjä syöttää hakukoneeseen. Update-

toiminnolla käyttäjä pystyy päivittämään tai editoimaan haluamaansa tietokenttää tietokannasta ja Delete-toiminnolla voidaan deaktivoida tai poistaa haluttu olemassa oleva merkintä tietokannasta. [4.]

3.2 MariaDB SQL -tietokanta

Projektimme tietokannaksi valittiin MariaDB-relaatiotietokanta, joka on haarautunut aiemmasta MySQL-tietokannasta. Vuonna 2009 Oracle Corporation osti Sun Microsystemin, joka omisti myös MySQL:n. Michael Widenius, joka on MariaDB:n pääkehittäjä, oli MySQL:llä töissä kehittämässä MySQL-relaatiotietokantaa ja huolestui Oracle Corporationin yrityskaupasta, joka aiheuttaisi ongelmia avoimen lähdekoodiohjelmiston kanssa, eikä mahdollistaisi kaikille ilmaista vastaavaa tietokantaa. [4.]

Valitsimme MariaDB:n tähän projektiin, sillä se on avoimeen lähdekoodiin perustuva SQL-tietokanta, jossa on täysin sama SQL-syntaksi kuin MySQL:ssä. Tämä mahdollistaa MySQL-tietokannan ohjelmointirajapintojen sekä kirjastojen käytön.

Tiimimme aiemmassa projektissa oli käytetty juuri sopivasti MariaDB-tietokantaa, ja olen itse aiemmin käyttänyt kyseistä tietokantaa kouluprojekteissani, joten sen käyttö tuntui luonnolliselta.

3.3 Python 3.8 -ohjelmointikieli

Käytin sovelluksessa Python 3.8.0 -versiota, joka oli uusin vakaa julkaisu Pythonilta, kun aloitin applikaation suunnittelun. Yrityksessämme on käytössä monia eri versioita pythonista, mutta suurin osa käyttää Python 2.7.5 -versiota, sillä se tulee monien käyttöjärjestelmäasennuspakettien yhteydessä asennettuna laitteisiimme. Python 2.7.x:n virallinen ylläpitäminen loppui 1. tammikuuta 2020 [5], jonka takia totesimme, ettei ole enää kannattavaa luoda uutta ohjelmistoa tällä Python-versiolla.

Python 3.8.0 on julkaistu 14. lokakuuta 2019 [6], ja yrityksemme on valinnut tämän seuraavaksi versioksi, jota käytetään muillakin osastoilla, kuten testiautomaatiotiimissä.

Testiautomaatiotiimi oli tehnyt valmiit asennuspaketit kyseiselle Python-versiolle, mikä helpotti asennusvaihetta, sillä yrityksessämme ei ole pääsyä internetverkkoon työkoneilta.

3.4 Flask-mikrosovelluskehys

Flask on Pythonin mikro-ohjelmistokehys, mutta se ei tarkoita, että se olisi mitenkään puutteellinen, vaan siinä on pyritty pitämään ohjelmisto yksinkertaisena, mutta skaalautuvana. Flask ei vaadi mitään erillisiä työkaluja eikä kirjastoja toimiakseen. Se ei määrittele tietokantaa, validointia tai muuta komponentteja, jotka mahdollisesti tarvitsisivat kolmannen osapuolen kirjastojen toiminnallisuutta. Flask on suosituimpia verkkosovelluskehyksiä Pythonilla, ja sillä on erittäin suuri yhteisö, joka jatkuvasti luo uusia sekä ylläpitää vanhoja lisäosia. [7.]

Flask käyttää samojen luojien muita Python-projekteja, kuten Werkzeugia ja Jinjaa. Werkzeug on WSGI-työväline, jota hyödynnetään ohjelmistoissa kaikissa pyynnöissä ja vastauksissa. Jinja on koodikehys, joka mahdollistaa pythonmaisen koodin käyttämisen HTML-merkkikieltä käytettäessä.

```
<!doctype html>
<title>Hello from Flask</title>
{% if name %}
    <h1>Hello {{ name }}!</h1>
{% else %}
    <h1>Hello, World!</h1>
{% endif %}
```

Kuva 3. Jinjan avulla tarkistetaan, onko käyttäjää olemassa ja tervehtii, muutoin palauttaa ”Hello, World”.

3.5 Gunicorn

Gunicorn on pythonin WSGI-rajapinnan HTTP-palvelin Unix-pohjalle. Gunicorn on erittäin skaalautuva ja toimii kaikkien verkkosovelluksien kanssa niin kauan, kun se kykenee kommunikoimaan WSGI-rajapinnan kanssa. Se mahdollistaa sovelluksen kommunikoimisen monien verkkopalvelimien kanssa ja pystyy pyörittämään monta eri prosessia samalla, kun applikaatio on käynnissä. Gunicorn on palvelinklusteri, joka osaa jakaa kuormituksen useamman palvelimen kesken, kun palvelimelle tulee useita samanaikaisia HTTP-pyyntöjä.

Gunicorn perustuu pre-forked worker -malliin, joka tarkoittaa, että master-säie pyörittää työntekijät käyntiin hoitamaan palvelimien HTTP-pyyntöjä, mutta ei kontrolloi miten työntekijät toimivat pyyntöjen kanssa. Tämä tarkoittaa, että jokainen työntekijä, jonka säie pyörittää käyntiin, on riippumaton kontrolleri. [8.]

3.6 Nginx

Nginx on verkkopalvelin, jota voidaan samalla käyttää eri tarkoituksiin, esimerkiksi reverse proxyna, kuormantasaajana ja http-välimuistina [9]. Nginx on kehitetty vapaana lähdekoodina, minkä takia se on saavuttanut suuren suosionsa vuosien saatossa. Nginx on maailman käytetyin verkkopalvelinohjelmisto, joka isännöi jopa 34 % kaikista verkkosivuista. [10.]

```
server {
    listen 80;
    server_name hellab-version;

    location / {
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_pass http://path/to/sock/;
    }
}
```

Kuva 4. Nginxin välipalvelinpyyntöjen konfigurointi.

Projektissamme käytämme Nginxiä välipalvelimien määrittelyssä. Kuvassa 4 näkyy konfigurointiesimerkki, jossa annetaan server_name:lle ”hellab-version”. Server_name:ssa määritellään verkkosivun DNS-nimi tai annetaan IP-osoite. Alemmassa kohdassa määrittelimme polun, josta välipalvelin löytää Gunicornille konfiguroidun Systemd.unit socketin.

3.7 CentOS 7

CentOS 7 on vapaaseen lähdekoodiin perustuva GNU/Linux-käyttöjärjestelmä. CentOS pohjautuu vastaavaan maksulliseen Red Hat Enterprise Linux (RHEL) GNU/Linux-käyttöjärjestelmään ja RHEL:n kehittäjät ovat aktiivisesti mukana CentOS:n kehityksessä. Molemmat käyttöjärjestelmät ovat hyvin samanlaisia ja eroavat vain vähän toisistaan. [11.]

Käytimme CentOS 7:ää projektimme isännöimisessä. Kyseinen käyttöjärjestelmä on erittäin laajasti yrityksemme käytössä jo valmiiksi, sillä sitä pidetään hyvin vakaana ja turvallisena.

4 Työn toteutus

4.1 Verkkosivujen historia

Verkkosivujen ensimmäinen sukupolvi tunnettiin nimellä Web 1.0, jonka loi Tim Buners vuonna 1989 [12; s. 410]. Web 1.0 oli erittäin alkukantainen, ja se tunnettiin myös tuolloin nimellä ”Read-Only Web”, joka tarkoittaa, että se ei ollut interaktiivinen. Tunnetuimpiin ominaisuuksiin kuuluivat staattiset verkkosivut, jotka käyttivät vain perus HTML-merkintäkieltä, ja kaikki, mitä käyttäjä näki verkkosivuilla, oli kovakoodattua sisältöä.

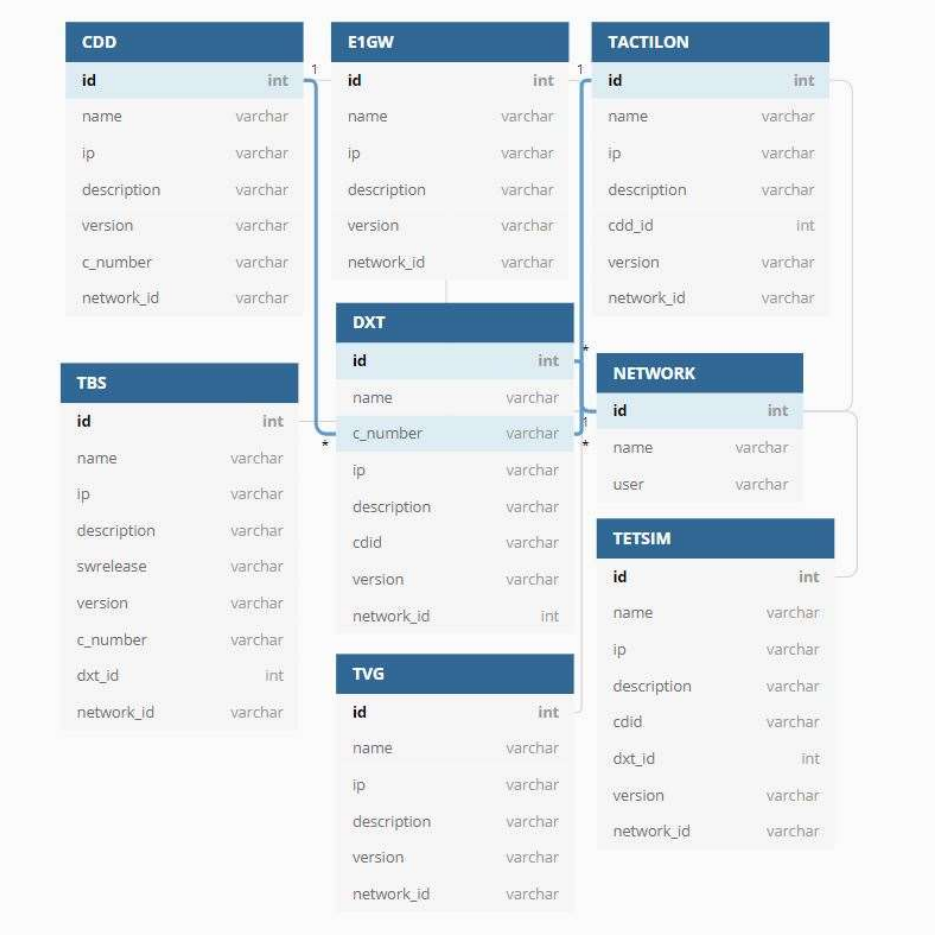
Toinen sukupolvi, Web 2.0, sisälsi suuria muutoksia aikaisempaan sukupolveen. Se mahdollisti interaktiivisen käyttäjäkokemuksen verkkosivun ja käyttäjän välillä, joustavan verkkosivujen suunnittelun ja monia muita isoja uudistuksia. Niistä tärkein uudistus Web 1.0:aan verrattuna oli palvelujen käyttäjien mahdollisuus kerätä tietoja yhteistöiden avulla, joka vahvisti kollektiivisen älykkyyden kehityksessä. Tämä mahdollisti myöhemmin viestintävälineiden informaation digitalisoitumisen [12 s. 411]. Web 2.0:n aikana blogit nousivat suureen suosioon, vaikka suurin osa blogeista koostui tekstistä, syntyi myös muunlaisia blogeja, kuten valokuvablogeja ja valokuvalogeja.

Kolmas ja nykyinen sukupolvi on Web 3.0, joka tunnetaan myös nimellä Semantic Web, on yhdistetyistä datoista koostuva verkko. Verkkoa kuvaillaan yleensä kerrosrakenteisena arkkitehtuurina, joka luo sen laajasta kokoelmasta teknologioita käyttäjille ympäristön, jossa he voivat sovelluksen kautta hakea tietoa kaikkialta verkosta. [12 s. 414].

4.2 Suunnitteluvaihe

Projektin suunnittelu alkoi keskustelemalla tiimin kesken sovelluksen tarpeista ja mitä käyttöominaisuuksia siihen halutaan. Tärkeintä suunnittelussa oli sovelluksen helppokäyttöisyys ja sen helppo ylläpitäminen. Projektin suunnitteluvaiheessa katsoimme aiempaa tiimin jäsenten luomaa sivustoa, jossa oli vain yksi näkymä ja joka ei ollut ollenkaan muokattavissa. Aiempi luotu sivusto toi tiedot suoraan tietokannasta ja näytti ne staattisesti nettisivulla. Sivustoa ei voinut muokata mitenkään, ellei mennyt suoraan tietokantaan ja sieltä manuaalisesti muokannut tietoa. Tämä järjestelmä ei ollut kovin skaalautuva ja hyödyllinen, sillä tiedot muuttuivat todella usein, mutta tietoja ei voitu ylläpitää.

Aloitimme projektin suunnittelemalla tietokannan rakenteen ja mitä tietoja mahdollisesti haluaisimme tallentaa tietokantaamme. Suunnitelman alussa kävimme läpi eri laitteiden tietoja sitä, millaisia tietoja tarvitaan ja mitä tietoja haluamme myöhemmin käyttää sovelluksessamme.



Kuva 5. Relaatiotietokannan diagrammi suunnitteluvaiheessa.

Suunniteltuamme tietokannan piti seuraavaksi päättää, millä teknologialla kyseinen sovellus tulisi luoda, ja päädyimme lopuksi kahteen vaihtoehtoon: Pythonin Django-sovelluskehyseen ja Flask-mikrosovelluskehykseen. Django on maailman suosituimpia backend-teknologioita, mutta totesimme, että kyseinen teknologia on liian monimutkainen ja raskas meidän tarpeisiimme. Päädyimme lopuksi Flask-mikrosovelluskehyksen käyttämiseen, sillä se on kevyt ja yksinkertaisempi ja siten helpompi kehittää ja ylläpitää.

4.3 Kehitysvaihe

Sovelluksen kehitys alkoi testaamalla sovelluksen toimivuutta ja teoreettista toimintaa. Ennen varsinaista sovelluksen kehitystä luotiin minimaalinen sovellus, jolla pystyttiin kokeilemaan Flaskin toimivuutta ja tutustumaan siihen liittyvään logiikkaan.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

Kuva 6. Minimaalinen Flask-sovelluksen luokka.

Kuvassa 6 luodaan instanssi luokalle ja määritellään URL-polku. Luomamme polku määrittelee kotisivun eli sivun, joka tulee näkyviin kun sovelluksen avaa, palauttamaan tekstin ”Hello World”, kun ohjelma ajetaan lokaalisesti. Luotu sovellus oli minimaalinen, joten se ei sisällä mahdollisia toimintoja eikä palauta HTML-sivua, vaan pelkän tekstin.

Minimaalisen sovelluksen toiminnan varmistettuamme aloitimme varsinaisen ohjelmistokehityksen. Ensimmäisenä aloimme suunnittelemaan Flaskin route()-dekoraattoria käyttäen, mitä funktioita haluamme käynnistää määritetyssä URL:ssa. Kyseistä dekoraattoria käyttäen luodaan polkuja ja määritellään, mitä halutaan palauttaa käyttäjän näkymäksi.

Seuraavassa kuvassa 7 tarkastellaan laajempaa esimerkkiä Flaskin route()-dekoraattoria, jota käytettiin myös todellisessa projektissa.

```
@bp.route('/')
def index_all():
    # Query all networks
    all_networks = models.NETWORK.query.all()
    # Query all tables in database
    all_tables = db.engine.table_names()
    # Remove NETWORK and USERS table from list
    all_tables.remove("NETWORK")
    all_tables.remove("USERS")
    # Return all networks and tables in database without NETWORK-table
    dxt = models.DXT.query.order_by(models.DXT.network_id.asc(), models.DXT.cdid.desc()).all()
    if current_user.is_authenticated:
        return redirect(url_for('bp.index'))

    form = LoginForm()
    # Validate login attempt
    if form.validate_on_submit():
        user = models.USERS.query.filter_by(name=form.name.data).first()
        if user and user.check_password(password=form.password.data):
            login_user(user)
            return redirect(url_for('bp.index'))
        flash('Invalid username/password combination')
    return render_template('templates_for_others/index.html', form=form, networks=all_networks, tables=all_tables, dxt=dxt)
```

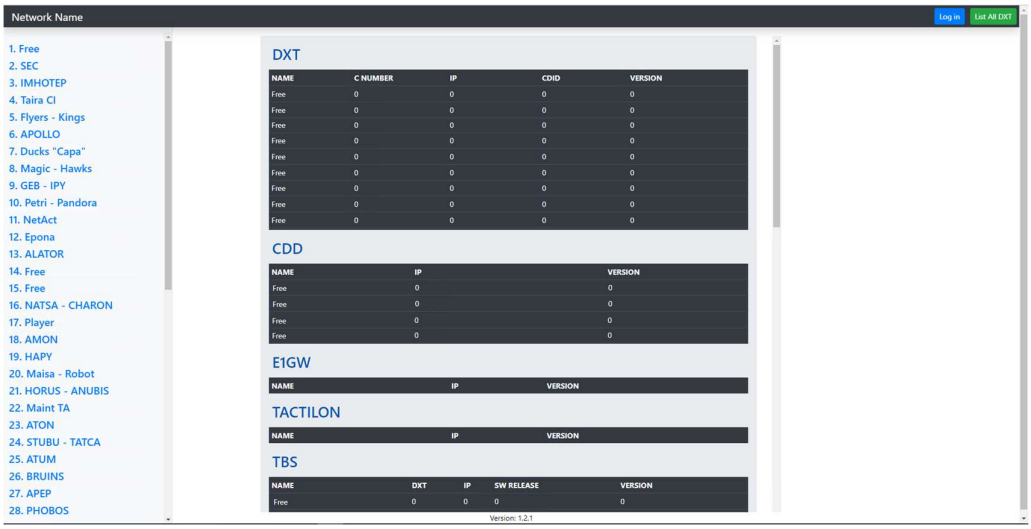
Kuva 7. Index-sivun kyselyt ja kirjautumistoiminto.

Tietokannan ja sovelluksen välisen yhteyden luomiseen valitsimme SQLAlchemy-kirjaston sen yksinkertaisen rakenteen ja nopeiden kyselyiden takia. Kuvassa 7 määritellään index-sivulle URL-polku ja kaikki sivun tietokantakyselyt. Funktiomme hakee aluksi tietokannan Network-taulukosta kaikki taulukoidemme nimet ja listaa ne tulevan sovelluksen vasemmassa reunassa navigointipalkin toimintona. Kyseisen navigointipalkin avulla voidaan siirtyä verkosta toiseen katselemaan, mitkä laitteet sijaitsevat missäkin (katso kuva 8).

Tietokantakyselyssä haetaan kerralla kaikki taulukot tietokannasta, mutta poistetaan matkan varrelta network- ja users-taulukot, sillä kyselyn on tarkoitus hakea suoraan tietokannasta kaikki laitteemme, ja yllä mainitut taulukot eivät ole laitteita.

Toisella tietokantakyselyllä haetaan samalle sivulle kaikkien taulukoiden sisällöt, kun edeltävä funktio kyseli ja haki taulukoiden nimet, jotka listataan suoraan tietokannasta. Kuvassa 8 navigointipalkin kohdalla vasemmassa laidassa näkyvät taulukoiden otsikot on haettu suoraan tietokannasta, joten nimet ovat dynaamisia ja niitä voi vaihtaa.

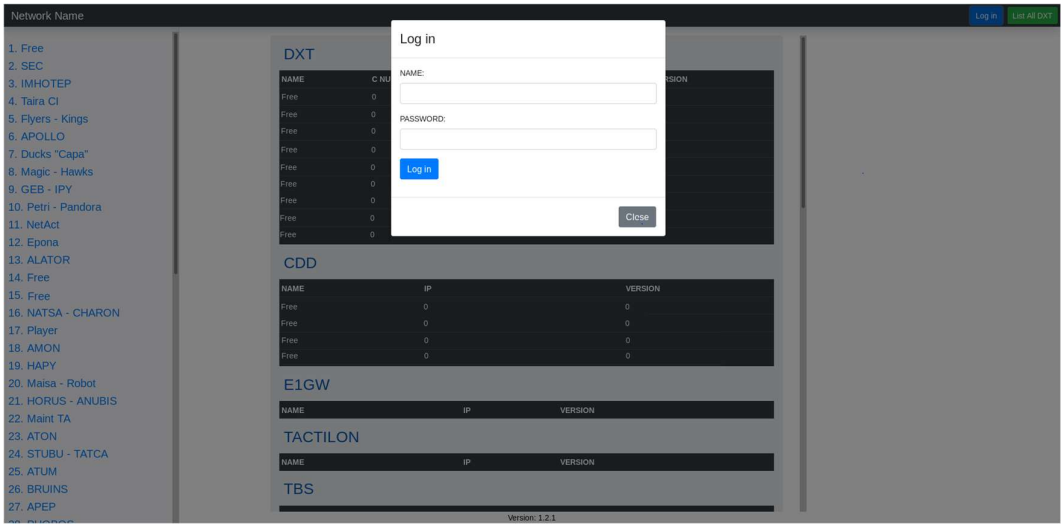
Verkkosovelluksessamme on myös ”List all DXT” -painike, jota painamalla sovellus kyselee tietokannasta kaikki DXT-laitteiden tiedot ja listaa ne taulukossa verkon numeroinnin ja viimeisimpien ohjelmistotasojen mukaan.



Kuva 8. Näkymä verkkosovelluksestamme kirjautumatta sisään.

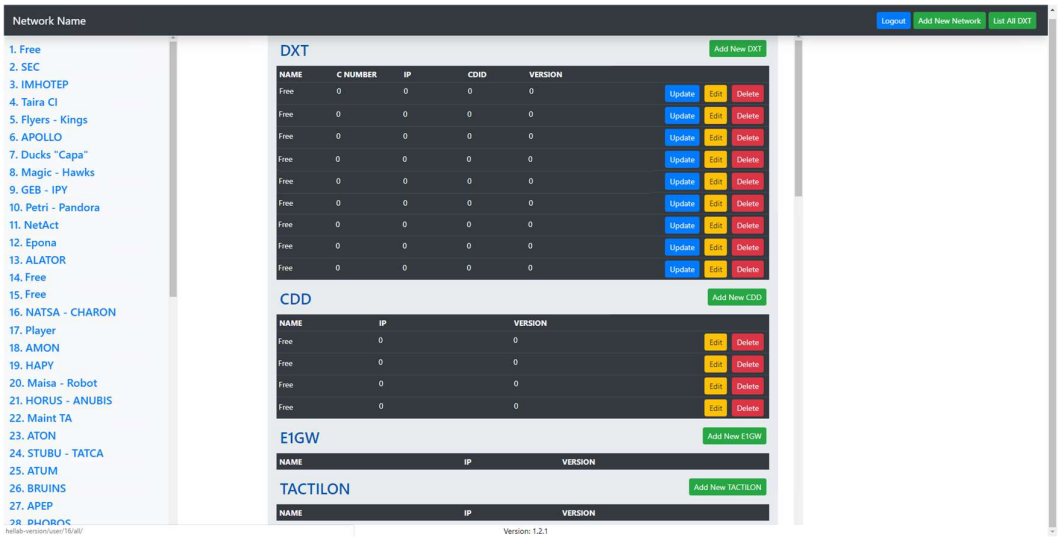
Index-sivulla on navigointipalkissa myös kirjaudu-painike, joka palauttaa modaalisen ikkunan, joka kysyy käyttäjältä käyttäjätunnuksen ja salasanan (katso kuva 9). Verkkosovelluksen kaikki muotoilut ja tyylittelyt on tehty käyttämällä Bootstrap4-CSS-kirjastoa.

Alkuperäisessä versiossa ei ollut hyödynnetty Bootstrap-kirjastoa, vaan kirjoitin kaikki CSS-tyylittelyt itse ja huomasin, että kyseinen sivusto oli erittäin kankea, eikä se ollut kovin responsiivinen.

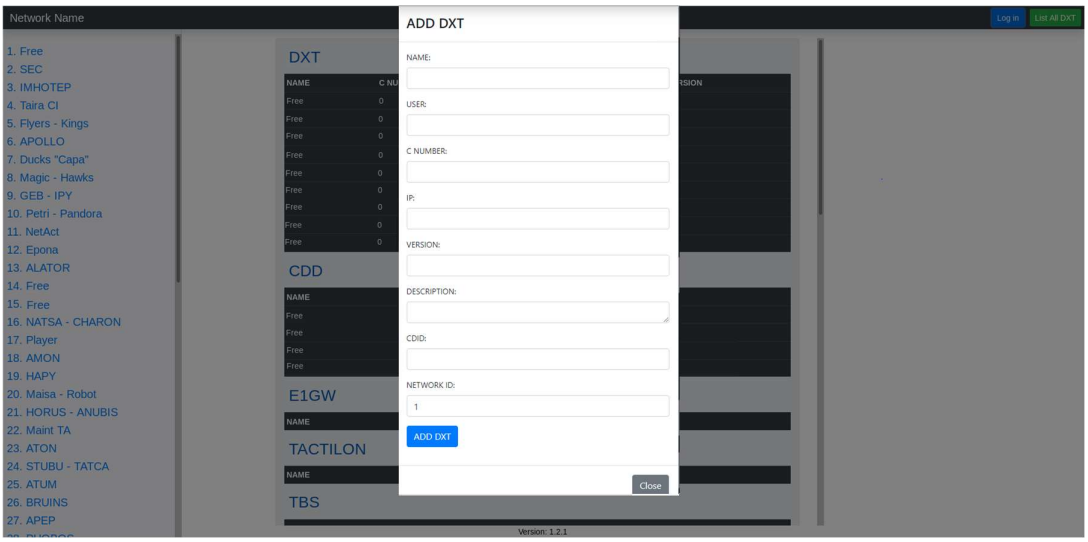


Kuva 9. Log in-painikkeen modaalinen ikkunan näkymä.

Sisään kirjauduttuaan käyttäjällä on enemmän käyttöominaisuuksia, kuten tietojen lisääminen taulukoihin, niiden muokkaaminen ja poistaminen.



Kuva 10. Käyttäjän näkymä sisäänkirjautumisen jälkeen.



Kuva 11. DXT:n lisäyksen modaalinen ikkuna, joka on identtinen DXT:n editoimisen modaalisen ikkunan kanssa.

Seuraavassa esimerkissä (kuva 12) käydään läpi DXT:n lisäysfunktion koodia projektissamme.

```
@bp.route('/add_dxt', methods=['POST'])
def add_dxt():
    if request.method == 'POST':

        name = request.form['name']
        c_number = request.form['c_number']
        ip = request.form['ip']
        version = request.form['version']
        description = request.form['description']
        cdid = request.form['cdid']
        network_id = request.form['network_id']
        user = request.form['USER']

        my_data = models.DXT(name, user, c_number, ip, version, description, cdid, network_id)
        db.session.add(my_data)
        db.session.commit()
        flash("DXT Added Successfully")
        return redirect(request.referrer)
```

Kuva 12. add_dxt()-metodi, joka hakee modaalisen ikkunan syötteistä tiedot ja lisää ne tietokantaan.

Käyttäjä voi kirjauduttuaan painaa DXT-taulukon kohdalla ADD DXT -painiketta, joka avaa modaalisen ikkunan. (Katso kuva 11). Kaikki tiedot, jotka ovat listattuna sovelluksessamme, on Jinjaa hyödyntäen iteroitu ja listattu ID-tunnisteen mukaan. Kaikilla painikkeilla, jotka näkyvät käyttöliittymässämme, on uniikit tunnisteet, joten kun tiedot on lisätty, sovellus osaa lisätä oikean laitteen oikeaan verkkoympäristöön.

Kuvassa 12 näkyy DXT:n lisäämisfunktio. Funktio aktivoituu sovelluksen puolelta painettua ADD DXT -painiketta, joka aktivoi modaalisen ikkunan. Ikkunassa on pakollisten tietojen sarakkeet (kuva 11). Kun kaikki pakolliset tiedot on annettu, add_dxt()-funktio tekee pyynnön ja hakee tiedot funktion sisälle. Kun tiedot on tallennettu funktioon, ne lisätään SQLAlchemyn session.add()-metodin avulla erittäin nopeasti ja yksinkertaisesti. Kun tiedot on lisätty tietokantaan funktio, päivittää sivun uusilla tiedoilla.

```
#Update button activates update script of DXT
@bp.route('/runscript/', methods=['GET', 'POST'])
def runscript():
    #Insert DXT data into database and launch update.py through subprocess.
    if request.method == 'POST':
        name = request.form['name']
        c_number = request.form['c_number']
        ip = request.form['ip']
        my_data = models.DXTLIST(name, c_number, ip)
        db.session.add(my_data)
        db.session.commit()

    def inner():
         with subprocess.Popen(
                [sys.executable or 'python', '/var/www/webapp/project/Script/update.py',
                 textwrap.dedent("")],
                stdin=subprocess.DEVNULL, stdout=subprocess.PIPE, stderr=subprocess.STDOUT,
                bufsize=1, universal_newlines=True) as p:
            for line in p.stdout:
                yield "<code>{}</code>".format(html.escape(line.rstrip("\n")))
                yield "<br>\n"
    return Response(inner(), mimetype='text/html')
```

Kuva 13. DXT:n ohjelmistopäivitys skriptin kutsufunktio.

Kuvassa 13 näkyy Update-painike, joka aktivoi DXT:n päivitysohjelman. Update-painike palauttaa modaalisen ikkunan, joka kysyy laitteen nimeä, IP-osoitetta ja sen C-numeroa ja tallentaa ne erilliseen tietokannan taulukkoon. Update-skriptissä haetaan aiemmin mainitut tiedot taulukosta viimeksi lisätyn rivin mukaan varmistaen, että tiedot täsmäävät juuri lisätyn laitteen mukaan. Seuraavaksi funktio kutsuu Pythonin omaa subprocess-moduulia ja ajaa python-skriptin. Funktio palauttaa standardisoituna ulostulona kutsutun python-skriptin konsoliulostulona, joka iteroidaan for-loopissa, jotta ulostulo saadaan jaksotettua. Lopuksi funktio formatoi ja palauttaa ulostulon verkkosovellukseen HTML-tyylissä, samassa tyylissä kuin Python-skripti palauttaisi, kun sitä ajetaan normaalisti.

Kuvaa ei voida näyttää arkaluonteisen tiedon takia.

4.4 Python-skriptit taustalla

Tietokantaa pidetään ajan tasalla luomalla Python-ohjelmointikielellä skriptejä, jotka tiedustelevat eri kaikkien tuotteiltamme niiden versiot SSH-ohjelman kautta. Python-skripti yhdistää laitteisiin Pythonin Paramiko-kirjastoa hyödyntäen ja tiedustelee tietyillä Bash- eli unix-pohjaisilla komennoilla tai MML-komennoilla laitteiston ohjelmatasoa, jonka jälkeen se päivittää automaattisesti tietokantaa. Tietokannan päivityksen yhteydessä merkataan, onko tieto automatisoidulla skriptillä haettua tietoa vai onko se manuaalisesti laitettu tietokantaan verkkosovelluksemme kautta, sillä meillä on suljettuja verkkoja, jonne ei ole pääsy mahdollisuuksia SSH-ohjelman kautta kuin tietyistä verkoista. Kone, jolla ajetaan skriptejä, kuuluu eri verkkoon kuin kyseiset suljetut verkot.

Skriptien ajot on automatisoitu ajamaan öisin joka arkipäivä crontab-ohjelmaa. Crontab on töiden aikataulutusohjelma, jonka avulla voidaan määritellä, kuinka usein ja mihin aikaan skriptit ajetaan unix-pohjaisella käyttöjärjestelmällämme.

5 Tulokset

Projektimme lopputuloksesi luotiin tiimimme omaan käyttöön CRUD-verkkosovellus, joka on monipuolinen ja yksinkertainen käyttää. Tarkoituksena oli luoda sovellus, joka säästäisi aikaa tulevaisuudessa tutkimus- ja tuotekehitysosaston laitteiden ylläpidossa ja hallinnoimisessa.

Kehitetty verkkosovellus on ollut käytössä tutkimus- ja tuotekehitysosaston laboratoriotiimissä viimeiset kaksi kuukautta, ja se on osoittautunut erittäin käytännölliseksi ja aikaa säästäväksi sovellukseksi. Tiimin palautteen mukaan kehitetyllä verkkosovelluksella on luonnollisesti kehitysmahdollisuuksia tulevaisuudessa, mutta tämänhetkinen verkkosovellus on ollut käytännöllinen, ja sillä voitaisiin korvata mahdollisesti aikaisempi tietojen ylläpito- ja hallinnointijärjestelmä, joka oli tiimissämme aikaisemmin käytössä.

Lopullinen tulos on selaimella toimiva verkkosovellus, jossa on relaatiotietokannan fundamentaaliset toiminnot, joilla voidaan ylläpitää ja hallinnoida osastomme laitteiston perustietoja ja tarkistaa ohjelmistotason.

6 Yhteenveto

Opinnäytetyön tarkoituksena oli kehittää Airbus Defence and Space Oy:n tutkimus ja tuotekehitysosaston laboratoriotiimin käyttöön verkkosovellus, jossa voitaisiin ylläpitää ja hallinnoida osaston laitteiston perustietoja relaatiotietokannassa. Verkkosovelluksen fundamentaaliset CRUD-toiminnot toimivat SQL-relaatiotietokannan kanssa

Kehitetty verkkosovellus on todettu hyödylliseksi työkaluksi laitteiston perustietojen ylläpitämisessä ja hallinnoimisessa. Sovelluksen käyttöönotto on helpottanut ja nopeuttanut laboratoriotiimin työtä laitteiston perustietojen haussa ja DXT:n ohjelmistotason päivityksessä.

Verkkosovelluksessamme ei käytetä tällä hetkellä JavaScript-sovelluskehyksiä kuin Bootstrapin CSS-sovelluskehystä, joka sisältää oman Bootstrapin JavaScript-kirjaston. Tavoitteena on tulevaisuudessa käyttää mahdollisesti React.js- tai Vue.js-JavaScript-sovelluskehystä. JavaScriptin lisäämisen myötä joutuisimme muuttamaan nykyisen backendin REST-rajapinnan mukaisemmaksi ja funktioiden tulisi palauttaa JSON-muodossa, joka helpottaisi tiedon iteroimisen.

Opinnäytetyössä kehitetty verkkosovellus on ollut mainio oppimismahdollisuus itselleni, sillä pääsin tutustumaan full stack -kehityksen ja verkkosovelluksien perusteisiin. Tutustuttuani aiheeseen kiinnostuin enemmän verkkosovellusten kehityksestä, kun pääsin projektissani kehittämään koko sovelluksen alusta loppuun asti. Tämä antoi minulle mahdollisuuden tutustua verkkosovellusten arkkitehtuuriin ja vahvisti osaamistani ohjelmistokehityksessä.

Lähteet

1 GitHub. Python libraries. Verkkoaineisto. https://github.com/search?l=Python&q=stars%3A%3E100&s=stars&type=Repositories. Luettu 10.11.2020.

2 JetBrains. Python Developer Survey. Verkkoaineisto. https://www.jetbrains.com/research/python-developers-survey-2018/ Luettu 11.11.2020.

3 Airbus Finland. References. Verkkoaineisto. https://www.airbusfinland.com/referenssit#TETRA. Luettu 1.11.2020.

4 MariaDB. Management SQL commands . Verkkoaineisto. https://mariadb.com/kb/en/account-management-sql-commands. Luettu 2.11.2020.

5 Python 2.7. Release Schedule. Verkkoaineisto. https://www.python.org/dev/peps/pep-0373/#update. Luettu 1.11.2020.

6 Python 3.8. Release. Verkkoaineisto. https://www.python.org/downloads/release/python-380/. Luettu 1.11.2020.

7 Flask. Foreword. Verkkoaineisto. https://flask.palletsprojects.com/en/1.1.x/foreword/. Luettu 5.4.2020.

8 Gunicorn. Latest Versiom Verkkoaineisto. https://docs.gunicorn.org/en/latest/index.html. Luettu 2.11.2020.

9 Nginx. Document. Verkkoaineisto. http://nginx.org/en/ Luettu 2.11.2020.

10 Netcraft. Web Server Survey. Verkkoaineisto https://news.netcraft.com/archives/2020/09/23/september-2020-web-server-survey.html Luettu 10.11.2020.

11 CentOS. About. Verkkoaineisto https://www.centos.org/about/ Luettu 2.11.2020.

12 ReseachGate. Verkkoaineisto. https://www.researchgate.net/publication/262562142_Incremental_Journey_for_World_Wide_Web_Introduced_with_Web_10_to_Recent_Web_50_-_A_Survey_Paper/ Luettu 11.11.2020.

13 Flask. Guide for Quickstart. Verkkoaineisto. https://flask.palletsprojects.com/en/1.1.x/quickstart/#a-minimal-application. Luettu 5.4.2020.