Sisällys

TIEA306 Ohjelmointityö itsearviointi ja jälkiselvitys

Tehtävän kuvaus, tausta ja tavoitteet.

Työn ideana oli toteuttaa sovellus, joka hakee Twitterin API:sta päivittäin trendaavimmat puheenaiheet, ja tallentaa tiedon tietokantaan. Haetut tiedot näytetään ohjelmiston käyttöliittymässä. Myös jos aikaa oli, tarkoituksena oli toteuttaa mahdollisuus pääkäyttäjälle lisätä lisätietoja kunkin päivän kohdalle, tarkentaakseen sitä miksi jokin aihe on ollut trendaavana puheenaiheena. Tavoitteena oli suorittaa työ annetussa ajassa, eli kuudessa kuukaudessa, jonka ajan aihe on ”voimassa”.

Käytännön toteutus

Sovellus on esisuunnitelman mukaisesti toteutettu Pythonin Flask -frameworkilla, jolla on aiemminkin toteuttanut web-sovelluksia. Ohjelman rakenne noudattaa tyypillisen Flask-sovelluksen rakennetta, ja hyödyntää Flaskille ominaisia ”blueprintteja” erinäisten näkymien selkeään toteuttamiseen. Ohjelma käynnistetään app.py tai app.wsgi tiedostosta, riippuen millä alustalla sovellusta ajetaan; app.py on tiedosto jota lokaalisti komento ”flask run” kutsuu, kun taas app.wsgi on toteutettu Apache-palvelimelle tuotanto deploymenttia varten, ja sisältää siitä syystä hieman tarkempia määrityksiä. Kummalle kuitenkin yhteistä on kutsua __init__.py tiedostossa määritettyä create_app() -metodia, joka on Flaskille tyypillinen ”application factory” metodi, joka alustaa Flask-sovelluksen ”app” olion. Application factoryssä alustetaan myös sovelluksen tietokanta, rekisteröidään blueprintit, lisätään mahdollinen testidata, sekä käynnistetään sovelluksen tärkeimmän toiminallisuuden mahdollistava scheduler, joka toteuttaa sille annetun tehtävän sille määritellyssä aikavälissä; hakee trendaustiedot kerran päivässä ja tallentaa ne tietokantaan. Sovelluksen eri näkymien toiminnallisuus määritellään /views -hakemiston sisältämissä *näkymä*.py tiedostoissa, kun taas Flaskille tyypilliseen tapaan JINJA-templatet löytyvät oletuksena /templates -hakemistosta, josta Flask olettaa niiden löytyvän. Templates-hakemiston tiedostoissa siis määritellään sovelluksen käyttöliittymä, joka on html-tiedostoja, joihin Flaskin käyttämän JINJA template enginen avulla upotetaan python koodia, kuten silmukoita ja muuttujia. Staattiset tiedostot kuten css-tiedostot ja kuvat, Flask sovelluksessa sijoitetaan /static -hakemistoon. Hakemisto /control, sisältää nimensä mukaisesti sovelluksen toiminnallisuuden; eli python-tiedostot, joissa määritellään: API-avainten lataaminen käyttöön (api_keys.py), model.py jossa määritellään tietokannan rakenne, scheduler.py aiemmin mainitun schedulerin määritykseen, sekä trends.py joka sisältää kaiken trendaustietojen hakemisesta, haetun tiedon käsittelystä, tietokantaan tallentamisesta, tietokantatietojen hakemisesta ja poistamisesta lähtien.

Sovelluksen SQL-tietokanta sisältää kaksi eri taulua, jotka ovat trending_data ja additional_info. Tietokannan hallintaan sovelluksessa on käytetty flask-sqlalchemy

kirjastoa, joka tekee sql:n käytöstä python ohjelmistoissa erityisen helppoa. Oheisesta kuvasta voimme havainnoida tietokannan rakennetta:

```
class trending_data(db.Model):

    # id -> primary key
    id = db.Column(db.Integer, primary_key=True)
    # String muotoa dd-mm-yyyy
    date_string = db.Column(db.String(10), nullable=False, unique=True)
    # JSON muotoinen String
    trends_string = db.Column(db.Text, nullable=False)

    additional_info = db.relationship('additional_info', backref='trending_data', lazy=True)

    db.UniqueConstraint(date_string)


class additional_info(db.Model):
    # id -> primary key
    id = db.Column(db.Integer, primary_key=True)
    # vapaateksti
    additional_info_string = db.Column(db.String(500))
    # lähteiden URL
    sources_url_string = db.Column(db.String(500))
    # päivän id
    trending_data_id = db.Column(db.Integer, db.ForeignKey('trending_data.id'),
        nullable=False)
```

Huomattavaa on kahden taulun välinen yhteys; kunkin päivän ”additional_info” on lista objekteja additional_info -taulusta.

Sovelluksen käyttöliittymä on hyvin yksinkertainen, mutta mielestäni erittäin siisti ja toimiva. Se koostuu html-tiedostoista, joihin aiemmin mainitun mukaan istutetaan toiminnallisuutta ja dynaamista dataa jinja-syntaksin avulla. Sovellukselle olisi voinut kehittää hienomman, esim. johonkin javascript-frameworkkiin pohjautuvan käyttöliittymän, mutta yhden kurssin vastuuopettajan mukaan kurssilla ei ole tarkoituksena opetella mitään uutta, vaan käyttää aiemmin opittuja taitoja ja menetelmiä. Minulla ei ole aiempaa kokemusta esim. react-frontendin ja flask-backendin yhdistämisestä, joten jätän sen mielelläni jatkokehittäjälle.

Kehityksen aikana sovellusta voidaan ajaa lokaalisti, jolloin se toimii Flaskin oman web-palvelimen avulla. Tätä palvelinta ei kuitenkaan voida käyttää tuotannossa, josta Flask sovellus käynnistyessään ilmoittaakin seuraavasti: ”WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.” Sovelluksen ”tuotannossa” käytössä onkin siis Apache web-palvelin. Flask-sovelluksen ja Apachen välillä vaikuttaa apache-moduuli mod_wsgi, joka mahdollistaa sovelluksen ja palvelimen välisen kommunikaation. Sovellus käynnistetään antamalla Apachelle aiemmin kuvailtu app.wsgi. Tuotantoon deployaamisesta tarkemmat ohjeet ja muistiinpanot löytyvät versionhallinnasta; /apache_conf/httpd.conf -tiedostosta.

Sovelluksen käyttöohjeet on kuvattu versionhallinnan README tiedostossa, kuten yleisesti on tapana.

Oman työn arviointi.

Mielestäni työn tekeminen onnistui juuri niin kuin ajattelinkin, eikä suurempia haasteita tullut vastaan. Yksi vaikeimmista asioista työn tekemisessä oli pohtia se, missä sovelluksen toimintaa voi demonstroida, eli minne sovellus voidaan laittaa pyörimään niin että se voi kerätä trendaustietoja useiden päivien ajalta. Yhdeksi vaihtoehdoksi tämän osalta tuli ilmainen Heroku. Ongelmaksi sen osalta muodostui se, että Herokun ilmaiset kontit nollautuvat kerran päivässä, pyyhkien kaiken levylle kertyneen datan, kuten tietokannan. Löysin kuitenkin pian Google Cloudin, jossa hyvin edulliseen hintaan voi pystyttää pieniä virtuaalikoneita, joista valitsin käyttööni Centos 7 palvelimen, jonka käyttämisestä minulla on aiempaa kokemusta. Palvelimen hinnaksi googlen arvion mukaan tulee noin 7 € kuukaudessa.

Twitterin API:n dokumentaatioon tutustuttua pian huomasin, että sitä kautta saatavat trendaustiedot ovat kategorisoitu maantieteellisen sijainnin mukaan, joka annetaan parametrina tehtävään GET-pyyntöön. Mahdolliset maantieteelliset sijainnit sovellus noutaa /control/locations.txt tiedostoon. Näissä sijainneissa ei ole Suomea mukana, ja kun ”Worldwide” sijainti antaa trendaustietoina paljon aasialaisia kirjaimia sisältäviä puheenaiheita, koin parhaaksi hakea vain USA:n trendaustiedot, jotta sovelluksen kielenä voidaan käyttää pelkästään englantia. Kaikkien eri sijaintien trendaustietojen hakeminen taas olisi erittäin työlästä ja tietokanta paisuisi päivittäin merkittävästi. Tämä toiminnallisuus voidaan kuitenkin toteuttaa tulevaisuudessa, mutta tässä projektissa en siihen ryhtynyt, sillä käytössäni on pienin mahdollinen palvelin, jolla ei ole runsaasti levytilaa.

Sovellus pyörii itsenäisesti palvelimella, enkä ole havainnut sen kaatumista. Olen pyrkinyt parhaani mukaan löytämään tapoja käsitellä virheitä sovelluksessa niin, että ne eivät johtaisi kaatumiseen. Käyttäjän antamat syötteet tarkastetaan, eli virheellisiä syötteitä sovellukselle ei voi antaa tavallisen käyttäjän puolesta. Ongelmaksi uuden asentajan puolesta voi olla esimerkiksi se, että API-avaimia ei ole käytössä, sillä niitä ei turvallisuus syistä voida sijoittaa versionhallintaan. Tämäkään ei kuitenkaan aiheuta sovelluksen kaatumista, vaan silloin sovellus ei kykene hakemaan uusia trendaustietoja twitterin API:sta.

Tehtävän ajoittaminen puolestani oli melko heikkoa, alussa tein paljon töitä sovelluksen eteen ja sain tehtyä perustoiminnallisuuden, mutta sitten koittivat kandin aiheuttamat kiireet, joista suunnitelmassanikin mainitsin. Projektin aikarajaksi on kuitenkin säädetty 6kk, johon kerkesin, vaikka sovellusta en juurikaan edistänyt kevään aikana. Viimeisinä viikkoina käytin lähes jokaisen päivän sovelluksen kehittämiseen ja ideoimiseen. Olen siis onnistunut projektissani, sillä olen pystynyt toteuttamaan suunnitelmassani kuvaaman sovelluksen, joka sisältää myös lisäominaisuuden; mahdollisuus pääkäyttäjälle lisätä tietoja päivien kohdalle.

Sovelluksessa käytetyt teknologiat eivät ole olleet käytössä käymilläni kursseilla yliopistossa, mutta olen opetellut käyttämään niitä työelämässä ja vapaa-ajalla. Esim. Git-versionhallintaa ei ole käytetty aiemmin käymilläni kursseilla, mikä mielestäni on ollut todella outoa, sillä se on ollut jo (kymmeniä)vuosia alan standardi. Kuitenkin ohjelmoinnin perusteet olen oppinut yliopistossa, ja nämä taidot ovat olleet hyödyllisiä myös tässä projektissa.

Ylläpitäjän tarvitsemat tiedot ja ohjeet.

Ylläpitäjän ei varinaisesti tarvitse tehdä mitään ylläpitääkseen sovellusta. Ylläpitäjän on kuitenkin mahdollista lisätä lisätietoja päivittäisten trendaustietojen tueksi, esim. halutessaan tarkentaa käyttäjälle miksi jokin aihe on ollut trendaavana tiettynä päivänä. Tämä ominaisuus on käytössä sovelluksen /admin -polussa, joka toistaiseksi on avoimena kaikille käyttäjille, jotta he voivat testata sovelluksen toimintaa. Tulevaisuudessa tarkoituksena on kuitenkin lisätä /admin polkuun kirjautuminen, joka voidaan toteuttaa mm. OpenID Connect / OAuth 2.0 protokollalla, käyttäen Apachen mod_auth_openidc -moduulia, jota olen käyttänyt aiemmin työelämässä. Kirjautuminen voi siis olla esim. google tai twitter tunnuksilla.

Sovelluksen mahdollinen jatkokehitys sekä ohjeet jatkokehittäjälle.

Tässä itsearviossa olen jo käsitellyt muutamia erilaisia ideoita sovelluksen jatkokehitykseen. Näitä ideoita ovat mm:

- Hienompi käyttöliittymä (Esim. React.js)
- Eri maantieteellisten sijaintien trendaustietojen tallentaminen ja esittäminen
- OIDC-kirjautuminen Twitter- tai Google-tilin avulla.
- Mainosten lisääminen sivustolle

Sillä olen itse sovelluksen ainoa kehittäjä, tulen luultavasti myös olemaan näiden ominaisuuksien toteuttaja, mikäli koen sen aikani arvoiseksi tai muuten vain opettavaiseksi projektiksi. Myös isomman ja tehokkaamman palvelimen valjastaminen sovelluksen alustaksi lienee hyvä idea, kun enemmän ja enemmän päivämääriä tallennetaan tietokantaan.

Lähteet

Twitter API: https://developer.twitter.com/en/docs/twitter-api/v1

Flask: https://flask.palletsprojects.com/en/2.1.x/

Flask-sqlalchemy: https://flask-sqlalchemy.palletsprojects.com/en/2.x/

Apache: https://httpd.apache.org/

Mod_wsgi: https://modwsgi.readthedocs.io/en/master/

mod_auth_openidc: https://github.com/zmartzone/mod_auth_openidc

Google Cloud: https://cloud.google.com/compute

Heroku: https://www.heroku.com/

Stack Overflow: https://stackoverflow.com/questions