Opinnäytetyö (AMK)

Tieto- ja viestintätekniikka

2021

Jarno Wetterstrand

TURVALLINEN WWW-SOVELLUS DJANGOLLA

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK / YAMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka

2021 | 13 sivua

Jarno Wetterstrand

TURVALLINEN WWW-SOVELLUS DJANGOLLA

Samalla kun www-sovellusten teknologiat kehittyvät ja lisääntyvät, myös niitä uhkaavat tietoturvariskit lisääntyvät sekä kehittyvä. Tämän takia on tärkeää pitää huolta www-sovelluksen turvallisuudesta, varsinkin jos web-sovellus sisältää henkilökohtaista dataa.

Opinnäytetyön tavoitteena oli kehittää kurssirekisteri asiakkaalle, joka tarvitsee vanhan kurssirekisterin tilalle ajantasaisempaa rekisteriä. Rekisteri sisältää henkilökohtaista ja arkaluontoisia tietoja, minkä takia tässä opinnäytetyössä painotettiin tietoturvallisuuden tärkeyttä. Kehittämisen jälkeen sovellus asennettiin asiakkaan Microsoft Azure -järjestelmään, josta API yhteys ulkopuoliselle lomakkeelle voitiin muodostaa käyttäen Django REST Framework kirjastoa. Tämä mahdollisti lomakkeelta saatujen tietojen tuomisen tietokantaan.

Opinnäytetyön web-sovellus toteutettiin käyttäen Django web-sovelluskehystä, Django REST Framework -kirjastoa, Django REST Framework API Key -kirjastoa sekä Allauth-kirjastoa. Django web-sovelluskehyksen sisäänrakennetut toiminnot olivat pääsyinä, kun tehtiin päätös mitä web-sovelluskehystä käytettiin kehittämiseen. Toiminnot, kuten cross-site scripting -suojaus, cross-site request forgery -suojaus ja SQL-injektiosuojaus helpottivat suojaamaan käyttäjän tiedot. Asiakkaan Azure-järjestelmä myös lisäsi kerroksia tietojen suojaamiseen, kuten kaksivaiheiseen tunnistautumiseen. Tämä toiminto pakottaa käyttäjän kirjautumaan Azureen ensin ja siitä sovellukseen. Tiedot ulkopuoliselta lomakkeelta on sekä autentikoitu API-avaimella että todennettu, jotta se ei voi injektoida haitallista tietoa sovellukseen.

Lopputuloksena saatiin turvallinen Django web-sovellus, joka oli asennettu asiakkaan Microsoft Azure -järjestelmään, jossa API-avain autentikoi turvallisen yhteyden ulkopuoliselle lomakkeelle.

ASIASANAT:

Django,Python, tietoturva, API

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2021 | 13 pages

Jarno Wetterstrand

DEVELOPING A SECURE WEB-APPLICATION WITH DJANGO

As technologies of web applications develop, so do the security threats they face. As such, when developing applications, it is vital to prioritize strengthening their security, especially if they store sensitive personal information.

The goal of this thesis was to develop a new registry for a customer seeking an up-to-date automated version of their old, dated registry. Due to the registry's content, comprising sensitive personal data, emphasis was given to the application's information security. Once developed, the application was installed in the customer's Microsoft Azure system, where a connection to an external form is established with an API using the Django REST Framework library. This allowed the received data to be imported into the database.

This web application was created using the Django web framework, the Django REST Framework library, the Django REST Framework API Key library, and the Allauth library. The built-in features of the Django web framework were a main factor in the decision to adopt this web framework. Features like cross-site scripting protection, cross site request forgery protection and SQL injection protection aided in reinforcing the security of user's data. The customer's Azure system also added several layers of security, such as the implementation of a two-factor authentication. This feature makes the user log in to Azure first and then log in to the application. Data from the external form is both authorized by the API key and authenticated to prevent harmful data from being injected into the application.

The goals of this thesis were successfully achieved as the outcome was a secure Django web application installed in the customer's Microsoft Azure system, in which the API key authorizes a secure connection between the API and an external form.

KEYWORDS:

Django, Python, information security, API

SISÄLTÖ

KUVAT

KÄYTETYT LYHENTEET TAI SANASTO

API	Ohjelmointirajapinta (Application programming interface)
CSRF	Sivujen risti pyyntöjen väärennös
HTML	Hypertekstin merkintäkieli
HTTP	Hypertekstin siirtoprotokolla
HTTPS	HTTP-protokollan ja TLS-protokollan yhdistelmä
JSON	Avoimen standardin tiedostomumoto
REST	Ohjelmointirajapintojen arkkitehtuurimalli
SQL	Standardoitu tietokanta kyselykieli
TLS	Liikenne kerroksen suojaus (Transport Layer Security)
XSS	Sivustojenvälinen komentosarja

1 JOHDANTO

Samalla kun www-sovellusten teknologiat kehittyvät ja lisääntyvät, myös niitä uhkaavat tietoturvariskit lisääntyvät sekä kehittyvät. On siis tärkeää pitää hyvää huolta tarvittavasta tietosuojasta kehittäessä www-sovelluksia. Lisäksi www-sovelluksien ollessa internetissä, ovat ne mahdollisia hyökkäyksen kohteita. Www-sovelluksien sekä verkkosivujen tietoturvallisuudesta on tehty aikaisempia opinnäytetöitä, kuten esimerkiksi Valentin Gusarovin vuonna 2020 tekemä *TheFIRMAN www-kehityksen tietoturvaohjeitten päivittäminen* ja Teemu Niemisen vuonna 2015 tekemä *Tietoturvallinen verkkosovellus Pythonilla ja Djangolla.*

Opinnäytetyön tarkoituksena oli ottaa selvää mahdollisista www-sovelluksen tietoturva riskeistä. Tietoturvaa tutkitaan myös Django www-sovelluskehyksen kannalta, ja kuinka sen sisäänrakennetut tietoturva-asetukset suojaavat nykyaikaisia tietoturvauhkia vastaan. Tietoturvaa piti myös miettiä siltä kannalta, että www-sovelluksella on API-yhteys ulkopuoliselle lomakkeelle.

Opinnäytetyön tavoitteena toteutetaan asiakkaalle kurssirekisteri käyttäen Django www-sovelluskehystä sekä SQLite-tietokantaa. Koska kurssirekisteri sisältää henkilökohtaista ja arkaluontoista dataa, tarvitsee sovelluksen kehittämisessä pitää erityistä huolta sovelluksen tietoturvallisuudesta. Asiakas julkaisee valmiin sovelluksen yrityksen omassa Microsoft Azure -järjestelmässä.

Luvussa 2 käydään läpi yleisesti www-sovelluksien turvallisuudesta, keskittyen toimintoihin, joita käytettäisiin vastaavanlaisessa projektissa, jota kehitettiin. Luvussa 3 käydään läpi minkälaisia turvallisuustoimintoja Django www-sovelluskehys sisältää. Luvussa 4 käydään läpi tarkemmin, kuinka käytännön työ on toteutettu tietoturvan puolesta. Lopuksi luku 5 on yhteenveto opinnäytetyöstä.

2 WWW-SOVELLUS

Ohjelmiston turvallisuus ja hakkereiden historia ulottuu pitkälle menneisyyteen. Hakkerit ovat yrittäneet ohittaa ohjelmistojen turvallisuutta niin kauan kuin ohjelmistoja on ollut, sekä nykypäivän teknologiat on rakennettu hakkereidein menetelmistä opitun pohjalta. (Hoffman 2020.) Käyttäjät ovat iso osatekijä www-sovellusta, tämän takia käyttäjien henkilökohtaisten tietojen turvaaminen on syytä olla iso keskittymiskohta www-sovelluksen turvallisuuden suunnittelua. On tarpeen määrittää, kuinka paljon on tarvetta keskittyä ohjelmiston tietoturvallisuuteen. Se, kuinka ohjelmisto rakennetaan, vaikuttaa hyvin paljon kuinka sovelluksen käyttäjät osallistuvat turvallisuuteen. (Burnett 2004.)

2.1 Käyttäjien todentaminen

Sisäänkirjautuminen pakottaa käyttäjän todentamaan henkilöllisyyden, yleensä käyttäen nimimerkkiä ja salasanaa. Turvallisuuden takia on tärkeää tehdä kirjautuminen sovelluksessa huolella, jotta voidaan olla varmoja kirjautujan henkilöllisyydestä. Koska sisäänkirjautumislomake on hyvin tärkeässä roolissa kun käyttäjiä tunnistetaan, on tärkeää huolehtia ja turvata sitä omilta haavoittuvuuksilta. Huonosti ohjelmoitu kirjautuminen on haavoittuvainen tietojen vuotamiselle, verkkourkinnalle sekä salasanan tietomurroille. Lisäksi lomake itsessään voi olla haavoittuvainen SQL-injektiota ja cross-site komentosarjoja (XSS) vastaan. (Burnett 2004.) Jos sovellus käyttää salaamatonta HTTP-yhteyttä käyttäjätunnusten lähettämiseen, on ulkopuolisen henkilön mahdollista siepata kirjautumistunnukset. Esimerkiksi samassa sisäisessä verkossa olevan henkilön on mahdollista siepata kirjautumistunnukset. (Stuttard, Pinto ja Pauli 2012.) Usea sovellus käyttää HTTP-yhteyttä todentamattomien sovelluksen alueissa ja vaihtavat turvallisempaan salattuun HTTPS-yhteyteen esimerkiksi sisäänkirjautumisessa. Tässä tapauksessa HTTPS-yhteyttä tulisi käyttää kun sisäänkirjautumissivu on ladattu, jotta käyttäjä varmistuu että kirjautumistiedot kulkevat turvassa. On mahdollista tulla vastaan www-sovellukseen, jossa sisäänkirjautuminen tapahtuu HTTP-yhteydellä ja vaihto HTTPS-yhteyteen tapahtuu, kun tiedot on lähetetty. Tämä ei ole turvallista, sillä mahdollinen hyökkääjä voi muuttaa kohde URL:ää jonka jälkeen käyttäjän tunnukset on murrettu. (Stuttard, Pinto ja Pauli 2012)

2.2 REST rajapinta

Suurin osa nykypäiväisistä www-sovelluksista käyttää RESTful-rajapintaa tai REST-maista rajapintaa, joka tuottaa tiedonvälitykseen tarkoitettua JSONia. REST on arkkitehtuuri spesifikaatio, joka määrittelee kuinka HTTP kartoittaa resurssit serverillä. Suurin osa nykypäivän REST-rajapinnoista käyttää JSON-tiedostomuotoa tiedon siirtämiseen. (Hoffman 2020.) Turvattoman sekä autentikoimattoman REST-rajapinnan dataa on mahdollista päästä muokkaamaan. Jotta voidaan estää autentikoimattoman henkilön mahdollisuutta päästä muokkaamaan JSON-tiedoston tietoja, tarvitsee REST-rajapinta turvata.(Marshall 2020.) REST rajapinnassa on tuki TLS:lle (Transport Layer Security), joten autentikoimaton hyökkääjä ei onnistu pääsemään tai muokkaamaan dataa (Opidi 2020). Nykyaikaisten www-sovellusten asiakaspuoleisen sekä palvelimien yhteyksiä on paljon, joten on datan siirtäminen standardi muodolla on hyvin suositeltavaa. (Hoffman 2020) JSON on yksi mahdollisista ratkaisuista tähän. JSON on avoimen standardin tiedostomuoto, joka muun muassa on todella kevyt. Tämä vähentää verkon kaistantarvetta. JSON myös vaati todella vähän jäsentelyä, joka vähentää asiakasohjelman sekä palvelimen rasitusta. (Hoffman 2020.)

2.3 Tietokanta

Tietokanta koostuu sekä datasta, että metadatasta. Metadatan tarkoituksena on kuvailla kuinka data on tallennettu tietokantaan. Jos tietää datan järjestyksen, voi sen metadatan avulla noutaa. Metadata on tallennettuna data dictionary osioon, joka kuvailee tietokannan taulut, sarakkeet, indeksit sekä muut mahdolliset asiat, jotka muodostavat tietokannan. (Taylor 2010.) www-sovelluksia yhdistävää interaktiivisuus, sekä ne ovat hyvin usein tietokantapohjaisia. Hyvin usein www-sovellukset sisältävät koodeja tai skriptejä, jotka hakevat tietoja tietokannasta. (Clarke 2012.)

Tietokannan turvaaminen suojaa useita asioita, kuten esimerkiksi tietokannan tietoja, tietokantojen hallintajärjestelmiä, tai siihen liittyviä ohjelmistoja. Tietokannan turvallisuus voi olla hyvin monimutkaista, ja siihen myös sisältyvät muut tietoturvan teknologiat ja käytännöt. Myös käytettävyys kulkee rinnakkain tietoturvan kanssa. Liian turvalliseksi kehitetty tietokanta, voi olla liian hankala käyttää, mutta turvatonta tietokantaa on helpompi käyttää (IBM Cloud Education 2019).

Tietokannan uhkia voivat esimerkiksi olla työpaikan sisäiset uhat, inhimilliset erehdykset, tietokannan SQL-injektiot tai puskurin ylivuotovirheen hyväksi käyttäminen. Työpaikan sisäisiä uhkia voi olla esimerkiksi työntekijä, joka kaappaa pääsyn tietokantaan. Voi myös olla, että työntekijä voi vahingossa jakaa pääsyn tietokantaan muille. Sisäpiirin uhkia voi torjua päivittämällä tietokannan käyttöoikeus käytäntöjä. Tietokannan SQL-injektiot tapahtuvat, kun hyökkääjä pääsee lataamaan haitallista SQL-koodia tietokantaan, ja järjestelmä luulee sitä oikeaksi koodiksi. Näitä vastaan voi turvautua valvomalla tietokannan merkkijonoja, sekä tarkastaa tietokanta säännöllisesti (Ilyukha n.d.). Ylivuotovirhettä vastaan voidaan suojautua pitämällä huolta, että tietokantaan syötetty data on vahvistettu, jotta se sisältää vain oikeita tietoja ja sisältää vain sopivan mittaisia merkkijonoja (Vyas 2020).

3 DJANGO WWW-SOVELLUSKEHYS

Django on Python-pohjainen www-sovelluskehys, jonka vahvuuksina on nopea, puhdas ja käytännöllinen malli. Yksi Djangon ainutlaatuisemmista piirteistä on Djangon ylläpitäjän sivu, jonka Django luo automaattisesti tietokantaa luodessa. (Ravindran 2018.) Django on suunniteltu model-template-view (MTV) -mallin ympärille, joka on muuteltu versio model-view-controllerista (MVC), ja on käytössä useassa www-sovelluskehyksessä. Www-sovelluksien mallit helpottavat vähentämään koodin toistoa ja parantamaan koodin laatua. (Romano 2018.)

3.1 XSS-hyökkäys

Cross site scripting (XSS) hyökkääjä pystyy injektoimaan asiakasohjelman puoleista koodia muiden käyttäjien selaimiin. Tämä tapahtuu yleensä tietokannasta hakemalla ja näyttämällä käyttäjille haitallisia komentosarjoja. On myös mahdollista saada käyttäjä klikkaamaan linkkiä, joka ajaa hyökkääjän JavaScript koodia käyttäjän selaimessa. (Django Software Foundation 2021.) Haitallisella XSS-komentosarjalla voidaan saada aikaan esimerkiksi: käyttäjän salasanan vaihtaminen, tietojen kerääminen käyttäjältä tai suorittaa satunnaisia toimintoja (Shaw 2020).

Djangon keino näitä hyökkäyksiä vastaan on, että kaikki mallin data käsitellään epäturvallisena, ellei sitä ole erikseen määritelty turvalliseksi. Tämän myötä suurin osa XSS-hyökkäyksistä eivät toimi Djangoa vastaan. (Shaw 2020.) Djangon mallit suorittavat automaattisen koodinvaihtomerkin merkeille, jotka voivat ovat vaarallisia HTML-koodille (Django Software Foundation 2021).

3.2 CSRF-hyökkäys

Cross site request forgery (CSRF) -hyökkäyksellä loppukäyttäjä pakotetaan suorittamaan jokin haitallinen toiminto. Tämä tapahtuu kun loppukäyttäjä on kirjautuneena sovellukseen, sekä napsauttaa linkkiä. Hyökkääjä on voinut jollain tavalla saanut annettua linkin loppukäyttäjälle, esimerkiksi sähköpostilla. Onnistuneella CSRF-hyökkäyksellä on mahdollista pakottaa käyttäjä suorittamaan haitallisia toimintoja sovelluksessa johon hän on kirjautuneena. Kuten esimerkiksi tilisiirtoja, sähköpostin muuntaminen, ja niin

edelleen. Jos hyökkäyksen uhri on kirjautuneena ylläpitäjän tunnuksilla, CSRF-hyökkäys voi vaarantaa koko www-sovelluksen. (Cross Site Request Forgery (CSRF) | OWASP Foundation n.d.)

Django tarjoaa sisäänrakennetun suojan suurimpia osia CSRF-hyökkäyksiä vastaan. Djangon CSRF-suojaus toimii tarkastelemalla jokaisen POST pyynnön salaisuutta, joka on CSRF-eväste, mikä perustuu salaiseen satunnaiseen arvoon. Tämä estää hyökkääjää uudelleen lataamaan POST lomakkeen ja mahdollisuutta saada toista sisäänkirjautunutta käyttäjää lähettämaan lomaketta. Hyökkääjän tulisi silloin tietää kyseinen salaisuus, joka on käyttäjäkohtainen eväste. Turvallisuus syistä tämä salaisuus luodaan aina uudelleen, kun käyttäjä kirjautuu sovellukseen sisään. (Django Software Foundation 2021.)

3.3 SQL-injektio

SQL-injektio tapahtuu kun hyökkääjä pääsee upottamaan SQL-kyselyitä käyttämällä sovelluksen mahdollisia syöte kohtia. Onnistuneella SQL-injektio hyökkäyksellä on mahdollista esimerkiksi lukea arkaluontoista tietoa tietokannasta, muokata tietokannan tietoja tai suorittaa ylläpitäjän toimintoja tietokannassa. SQL-injektio hyökkäykset ovat yleistyneet tietokanta-pohjaisissa verkkosivuissa. Ongelma on havaittavissa sekä helposti hyväksikäytettävä. Tämän takia verkkosivut jotka sisältävät käyttäjätietoja ovat isolla todennäköisyydellä mahdollisia kohteita SQL-injektio hyökkäykselle. (SQL Injection | OWASP n.d.)

Djangon tietokanta kyselyt ovat suojattu SQL-injektio hyökkäyksiltä, sillä Djangon tietokanta kyselyt ovat rakennettu käyttäen kysely parametrisointia. Kyselyn SQL-koodi on määritelty erillään kyselyn parametreistä. Parametrit voivat olla käyttäjän antamia, joten ne voivat olla turvattomia. Käyttäjän antamat parametrit ovat koodinvaihtomerkitty tietokannan ajurin puolesta. Django antaa myös mahdollisuuden kirjoittaa raakoja SQL-kyselyitä, sekä suorittaa mukautettuja SQL-kyselyitä. Näiden toimintojen käytössä tarvitsee olla varovainen, sekä käyttää niitä vain niukasti, varsinkin jos käyttäjä voi näitä kyselyitä tehdä. On pidettävä myös erityistä huolta koodinvaihtomerkeistä jos käyttää mukautettuja tai raakoja SQL kyselyitä sovelluksessa. (Django Software Foundation 2021.)

4 KURSSIREKISTERI

Asiakkaille piti tehdä kurssirekisteri, joka on turvallinen ja helppokäytettävä. Toteutukseen piti myös tehdä lomake, jolla pystyy hakemaan järjestettäville kursseille, mutta kyseinen lomake tuli erilliselle sivulle www-sovelluksen ulkopuolelle. Kyseinen yhteys lomakkeelle toteutettiin REST API:n avulla. Toteutukseen valittiin Django www-sovelluskehys.

4.1 Kirjautuminen ja Microsoft Azure

Www-sovelluksen kirjautumisessa käytettiin Djangon Allauth-kirjastoa, josta poistettiin rekisteröitymismahdollisuus. Sovellus on vain sisäisessä käytössä, joten riittää että vain ylläpitäjä voi lisätä uusia käyttäjiä ja poistaa vanhoja käyttäjiä.

Sovellus asennettiin asiakkaan Microsoft Azure järjestelmään, josta asetettiin Azureen omat kirjautumistunnukset. Käyttäjät kirjautuvat sisään ensin heidän Azure tunnuksilla, jonka jälkeen he kirjautuvat vielä järjestelmän ylläpitäjän luomilla tunnuksilla. Koska sovellus on tarkoitettu vain asiakkaan omaan käyttöön, ovat Azuren asetukset asetettu sallimaan vain asiakkaan IT henkilöstön sallimien henkilöiden sisään kirjautumisen.

Jotta saadaan tiedot lomakkeelta turvallisesti kurssirekisteriin, piti Azuressa luoda avain sallimaan liikenteen lomakkeen ja kurssirekisterin välille. Tämän voidaan varmistaa, että vain tältä tietyltä lomakkeelta voidaan saada tietoja.

4.2 REST API -kirjasto

Koska sovellus piti yhdistää ulkopuoliseen lomakkeeseen, sovellukseen tehtiin REST API käyttäen Djangon REST Framework -kirjastoa, joka autentikointiin käytettiin Django REST Framework API Key -kirjastoa.

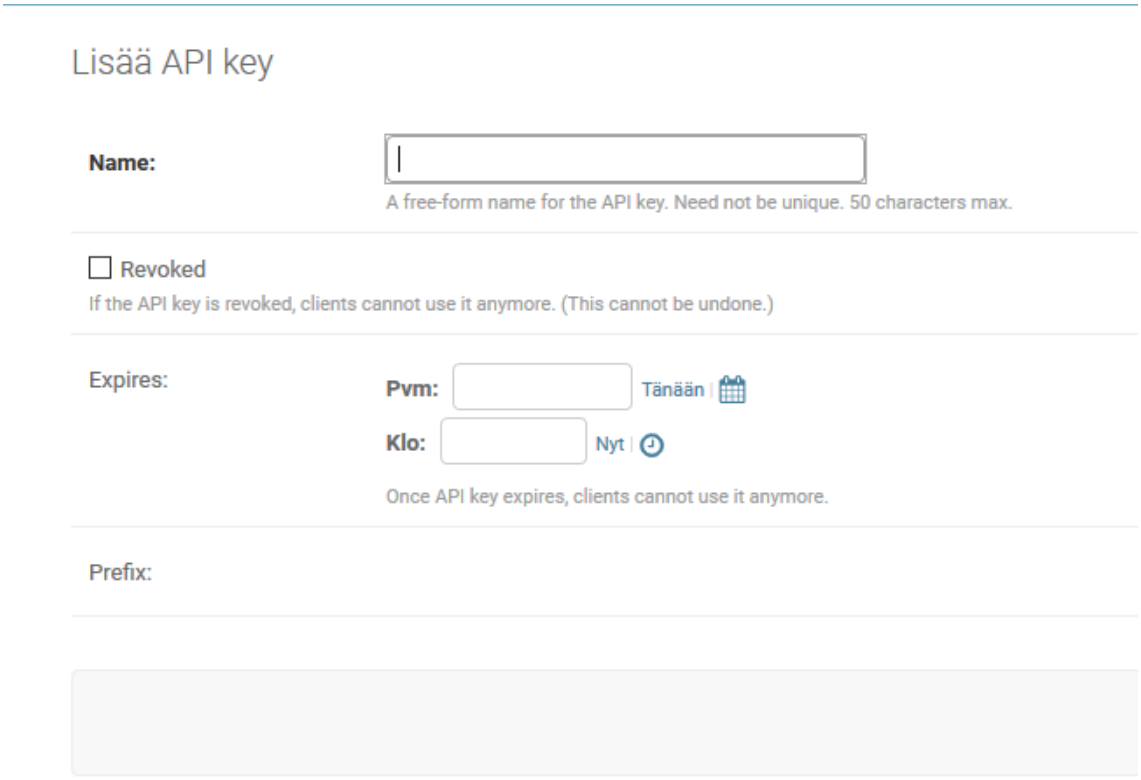
Jotta saadaan tiedot ulkoisesta lähteestä tietokantaan, tarvitaan sopiva tiedonsiirto tapa. Tiedonsiirtämiseen käytettiin JSON-tietomuotoa. Ulkoisesta lomakkeesta lähetettiin JSONia järjestelmään, joka muunnettiin tietokantaan sopivaksi käyttäen Django REST Framework -kirjastoa. Kuvassa 1 olevasta koodiesimerkistä nähdään kuinka lomakkeelta tulevia tietoja muunnetaan tietokannalle sopivaksi. Koodiesimerkissä näkyy myös

mihin malliin tieto tallennetaan. Koodiesimerkistä nähdään jos tietokannassa on jo kyseinen tieto, päivitetään se uusimmalla versiolla.

```
67  class ApplicantTechSerializer(serializers.ModelSerializer):
68      application_id = Application
69
70      class Meta:
71          model = ApplicantTechnology
72          fields = '__all__'
73
74      def create(self, applicant_tech_data):
75          # Create new obj if it doesnt exist, update if it does
76          try:
77              applicant_tech, created = ApplicantTechnology.objects.update_or_create(
78                  application_id=self.application_id,
79                  defaults=applicant_tech_data
80              )
81          # If more than 1 obj was found update the most recent one
82          except MultipleObjectsReturned:
83              print("Found two or more objects.")
84              applicant_tech = ApplicantTechnology.objects.filter(
85                  application_id=self.application_id,
86                  defaults=applicant_tech_data).last()
87              for key, value in applicant_tech_data.items():
88                  setattr(applicant_tech, key, value)
89              applicant_tech.save()
90
91          return applicant_tech
92
```

Kuva 1. Koodiesimerkki JSON serialisoinnista

JSONin autentikointi toteutettiin käyttäen Django REST Framework API Key -kirjastoa, jolla voitiin generoida API-avain. API-avain toimitettiin lomakkeen kehittäjille, jotta he voivat lähettää dataa turvallisesti kurssirekisteriin. API-avain voidaan luoda ohjelman ylläpitäjän sivulla, jonka jälkeen generoitu API-avain näkyy vain hetkellisesti sivun bannerissa, josta se pitää muistaa ottaa talteen. Pitää huomioida että jos ei ota API-avainta talteen, se ei ole enään jälkeenpäin nähtävissä. Kuvassa 2 olevassa API-avaimen luomis esimerkissä nähdään että API-avaimen luomiseen ei tarvita muuta kuin nimi, sekä API-avaimelle on myös mahdollista luoda määräaika jolloin API-avain on voimassa.



Kuva 2. Esimerkki API-avaimen luomisesta admin sivulla

4.3 Tulokset ja pohdintaa

Opinnäytetyöprojektin tuloksena oli Django www-sovelluskehyksellä tehty tietoturvallinen kurssirekisteri asiakkaalle. Valmis www-sovellus asennettiin asiakkaan Microsoft Azure -järjestelmään, joka lisäsi kerroksia tietoturvallisuuteen. API-avain turvaa ulkopuoliselta lomakkeelta tulevan datan. Asiakas on ottanut www-sovelluksen käyttöön ja on ollut siihen tyytyväinen.

Projektin REST API tehtiin käyttäen Django REST Framework -kirjastoa. Tällä kirjastolla on hyvä dokumentaatio, joka helpotti REST API:n kehittämisessä. Mahdollisiin ongelmiin oli hyvin usein vastauksia ja esimerkkejä saatavilla. REST API suojattiin käyttämällä API-avainta. API-avaimella oli myös hyvä dokumentaatio ja helpotti sen lisäämistä REST API:in.

Kurssirekisterin kehitysvaiheessa tuli myös hieman ongelmia. Koska kurssirekisteri asennettiin asiakkaan Azureen hieman myöhäisemmässä vaiheessa, pääsivät asiakas kokeilemaan sovellusta liian myöhään. Tämä johti muokkaus ehdotuksien

viivästymiseen ja aiheutti hieman kiireitä kehityksen loppuvaiheessa. Tämän takia kaikkia asiakkaan toiveista ei pystytty toteuttamaan

Tätä opinnäytetyötä tehdessä opittiin kuinka tärkeätä on saada tuote asiakkaalle kokeiltavaksi hyvissä ajoin. Muuten mahdolliset korjaus- ja muokkaus ehdotukset voivat viivästyä ja aiheuttaa kiireitä.

Django osoittautui hyväksi valinnaksi projektiin, sillä se tarjosi hyvät sisäänrakennetut tietoturva asetukset. Djangon dokumentaation ja yhteisön ansiosta, toimintojen tekeminen onnistui hyvin. Django on hyvin modulaarinen www-sovelluskehys, joten uusien toiminnallisuuksien kehittäminen helpottuu. Tämän takia projektin jatkokehittäminen on hyvinkin mahdollista, sekä asiakkaan toivomat toiminnallisuudet olisivat tulevaisuudessa mahdollista toteuttaa.

5 YHTEENVETO

Opinnäytetyön tehtävänä oli kehittää asiakkaalle kurssirekisteri käyttäen Django www-sovelluskehystä, johon voidaan tallentaa ulkopuoliselta lomakkeelta tulevia kurssihakemustietoja. Ohjelmisto asennettiin asiakkaan Microsoft Azure -järjestelmään, josta on API-yhteys ulkopuoliselle lomakkeelle. Kurssirekisteriä tehdessä oli tärkeää pitää huolta kunnollisesta tietoturvallisuudesta, sillä rekisterissä pidetään henkilökohtaista, sekä arkaluontoista dataa.

Kurssirekisterin saaminen asiakkaan Microsoft Azure -järjestelmään tapahtui myöhemmin kuin suunniteltu. Tämä johti siihen että asiakas ei päässyt kokeilemaan kurssirekisteriä aikaisemmin ja tarvittavat muutokset jouduttiin tekemään myöhemmin kuin toivottu. Osia asiakkaan toivomista toiminnoista ei voitu toteuttaa aikataulusta johtuen.

Koska Django on hyvin modulaarinen www-sovelluskehys, on mahdollista käyttää kurssirekisterin osia. Näitä osia voitaisiin hyvin käyttää vaikka toisenlaisen rekisterin tekemisessä.

Djangon modulaarisuuden ansiosta kurssirekisterin jatkokehittäminen on myös hyvinkin mahdollista. Sekä asiakkaiden toivomien toiminnallisuuksien toteuttaminen olisi hyvinkin mahdollista.

LÄHTEET

Buildmedia.readthedocs.org. 2021. *Django Documentation Release 3.2.1.dev*. [online] Saatavilla: <https://buildmedia.readthedocs.org/media/pdf/django/3.2.x/django.pdf> [Viitattu 14.4.2021].

Burnett, M. 2004. *Hacking the Code*. 1st ed. Burlington: Syngress Publishing, Inc., ss.2 - 56.

Clarke, J. 2012. *SQL injection attacks and defense, second edition*. 2nd ed. Waltham, Mass.: Syngress, ss.2.

Hoffman, A. 2020. *Web application security*. 1st ed. O'Reilly Media, Inc, ss.18 - 32.

IBM Cloud Education, 2019. *Database Security: An Essential Guide*. [online] IBM. Saatavilla: <https://www.ibm.com/cloud/learn/database-security> [Viitattu 2.5.2021].

Ilyukha, V. n.d. *What Is Database Security: Standards, Threats, Protection*. [online] Jelvix. Saatavilla: <https://jelvix.com/blog/database-security> [Viitattu 2.5.2021].

Marshall, L. 2020. *How to Secure REST APIs: API Keys Vs. OAuth*. [online] DreamFactory. Saatavilla: <https://blog.dreamfactory.com/how-to-secure-rest-apis-api-keys-vs-oauth-2/> [Viitattu 21.3.2021].

Opidi, A. 2020. *API Security — a Detailed Guide* –. [online] Rakuten Rapid API. Saatavilla: <https://blog.api.rakuten.net/api-security/> [Viitattu 21.3.2021].

Owasp.org. n.d. *Cross Site Request Forgery (CSRF)* | *OWASP Foundation*. [online] Saatavilla: <https://owasp.org/www-community/attacks/csrf> [Viitattu 23.5.2021].

Owasp.org. n.d. *SQL Injection* | *OWASP*. [online] Saatavilla: <https://owasp.org/www-community/attacks/SQL_Injection> [Viitattu 23.5.2021].

Ravindran, A. 2018. *Django Design Patterns and Best Practices*. 2nd ed. Birmingham: Packt Publishing, ss.9.

Romano, F. 2018. *Learn Python programming*. 2nd ed. Birmingham (UK): Packt Publishing, ss.434.

Shaw, A. 2020. *XSS Exploitation in Django Applications*. [online] Tonybaloney.github.io. Saatavilla: <https://tonybaloney.github.io/posts/xss-exploitation-in-django.html> [Viitattu 16.4.2021

Stuttard, D., Pinto, M. and Pauli, J. 2012. *The web application hacker's handbook*. 2nd ed. Indianapolis, Ind.: John Wiley & Sons, Inc., ss.169-170.

Taylor, A. 2010. *SQL for dummies*. 7th ed. Hoboken, N.J.: Wiley Publishing, Inc., ss.9.

Vyas, M. 2020. *How to Protect Against Buffer Overflow Attack*. [online] SecureCoding. Saatavilla: <https://www.securecoding.com/blog/how-to-protect-against-buffer-overflow-attack/> [Viitattu 3.5.2021].