Full-Stack aplikacija z uporabo AWS storitev

**Uvod**

V nalogi bom predstavil proces konfiguracije in delovanje spletne aplikacije, ki omogoča pridobitev podatkov s spletne strani igre <https://www.buzzerbeater.com/default.aspx?lang=sl-SI> in prikaz teh podatkov. Gre za seznam igralcev, ki so v tem trenutku na prodajni listi, program v ozadju pobere vse lastnosti teh igralcev in jih shrani v podatkovno bazo.   
Spletna aplikacija je napisana v frameworku React, ozadje »backend« s pomočjo frameworka Django, zgrajena pa sta s tehnologijo Docker. Za podatkovno bazo pa sem izbral Amazonovo storitev RDS.

**Arhitektura z AWS**

Za postavitev aplikacije sem uporabil AWS-ove storitve VPC,EC2 in DRS. Najprej sem napravil virtualno omrežje v oblaku, nanj dodal 2 subneta, enega javnega in enega privatnega. Javnega sem povezal na Route Table in na njo dodal Intenet Gateway, tako da ima javno omrežje povezavo z internetom.

A screenshot of a computer

AI-generated content may be incorrect.  
Slika 1: Resource map

**EC2 instanca**

V javno podomrežje sem dodal EC2 instanco. Ta server gosti frontend in backend del aplikacije s pomočjo Docker kontejnerjev.

A screenshot of a computer

AI-generated content may be incorrect.  
Slika 2: EC2 instanca

**RDS**

Podatki se shranjujejo v podatkovni bazi RDS, ki je dostopna preko endpointa. Dostopna je le znotraj VPC omrežja in se z njo lahko upravlja preko EC2 instance.

A screenshot of a computer

AI-generated content may be incorrect.  
Slika 3: RDS podatkovna baza

**Delovanje aplikacije**

Glavni del aplikacije je python skripta, ki s pomočjo Seleniuma samodejno odpre Chrome brskalnik navigira na spletno stran, vpiše podatke za login in potem navigira na željeno spletno podstran. Tam prebere vse igralce in to naredi za več strani (konfiguracija v kodi). Ko skripta opravi delovanje shrani podatke v podatkovno bazo in sicer v tabelo »igralci«, hkrati pa v tabelo »time\_of\_call« shrani še timestamp opravila. Tako se zapolnita dve tabeli.

Tabeli sta avtomatsko zgenerirani s pomočjo migracije modelov v Django aplikaciji, tako da so ob zagonu python aplikacije v bazi že zgenerirane tabele.

A screenshot of a computer

AI-generated content may be incorrect.  
Slika 4: Seznam tabel v podatkovni bazi

Ob zagonu spletne aplikacije se naredi API klic, ki pridobi podatke o seznamu shranjenih igralcev ter podatek o zadnjem opravljenem klicu skripte. Še en API klic se zgodi ob pritisku na gumb Scrape players v aplikaciji. Ta klic zažene python skripto.

**Docker**

Docker je naložen na EC2 instanci. Projekt je pridobljen z githuba in je razdeljen v 2 podmapi. Ena je frontend in druga backend. Vsaka mapa vsebuje svoj Dockerfile – datoteko ki opisuje, kako zgraditi sliko vsake aplikacije. Docker s pomočjo teh dveh datotek zgradi vsako okolje posebej in jih zažene kot svoj kontejner s svojim portom. Tako sta lahko dva okolja gostovana na istem strežniku.

**Komunikacija**

Frontend aplikacija komunicira z backendom preko API-klicev. Za to so odgovorni 3 endpointi:

* /scrape
* /lastCall
* /players

Frontend in backend sta zagnana na istem serverju, tako da imata isti IP, le port imata druačen. Za to poskrbi Docker in nastavitev v docker-compose.yml, ki pove na katerih portih sta kontejnerja zagnana.

A screen shot of a computer program

AI-generated content may be incorrect.  
Slika 5: Docker compose

Vsak projekt ima svojo .env datoteko, ki hrani pomembne spremenljivke. V Rect aplikaciji se določi endpoint backenda, v Fjango aplikaciji pa vse nastavitve podatkoven baze.  
  
URL backenda, ki je potreben za API klice:

VITE\_API\_URL=http://18.185.136.8:8000/

Nastavitve podatkovne baze:  
  
DATABASE\_NAME=postgres

DATABASE\_USER=postgres

DATABASE\_PASSWORD=postgres

DATABASE\_HOST=bb-scraper-db.c3ayey04ul7w.eu-central-1.rds.amazonaws.com

Pod HOST je zapisan endpoint podatkovne baze iz RDS-a.

A screen shot of a computer code

AI-generated content may be incorrect.  
Slika 6: Uporaba spremenljivk v settings.py