Usluga za automatizirano dodavanje rasporeda nastave u kalendar

Tonkovac, Hrvoje

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:425904

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-26**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

USLUGA ZA AUTOMATIZIRANO DODAVANJE RASPOREDA NASTAVE U KALENDAR

Završni rad

Hrvoje Tonkovac

Osijek, 2020.

SADRŽAJ

1. UVOD *1*

1.1 Zadatak završnog rada 1

2. PREGLED KORIŠTENIH TEHNOLOGIJA I RAZVOJNIH PRINCIPA *2*

2.1 Git distribuirani sustav za upravljanje izvornim kodom 2

2.2 Gitlab CI/CD sustav za kontinuiranu integraciju 2

2.3 Git Flow okvir rada za Git 3

2.4 Infrastruktura kao kod 3

2.5 Docker platforma za virtualizaciju na razini operacijskog sustava 3

2.6 Google Cloud i Compute Engine 4

2.7 Node.js izvedbeno okruženje i NPM upravitelj paketima 4

2.8 MongoDB baza podataka 5

2.9 OAuth2 protokol 5

2.10 Shell skripte 6

3. RAZVOJ INFRASTRUKTURE I DEFINIRANJE RAZVOJNOG PROCESA *7*

3.1 Željeni razvojni proces 7

3.2 Zakup infrastrukture na Google Cloudu 7

3.3 Docker tehnologija virtualizacije na razini operacijskog sustava 9

3.4 Docker Compose alat za konfiguraciju Dockera 10

3.5 Pokretanje više razvojnih okruženja pomoću Gitlab CI/CDa 12

4. RAZVOJ I KORIŠTENJE APLIKACIJE *16*

4.1 Dohvaćanje rasporeda pomoću FERIT XML aplikacijskog programskog sučelja 16

4.2 OAuth2 autentifikacija sa Google aplikacijskim programskim sučeljem 16

4.3 Dodavanje jednostavnih događaja u Google Calendar 17

4.4 Algoritam eksponencijalne odgode 18

4.5 Spremanje podataka o studentima u MongoDB bazu 19

4.6 Dohvaćanje svih studenata iz baze 20

4.7 Korištenje aplikacije **21**

5. ZAKLJUČAK ***24***

LITERATURA ***25***

SAŽETAK ***26***

ABSTRACT ***27***

ŽIVOTOPIS ***28***

1. UVOD

Internet stranica Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek svojim studentima i djelatnicima nudi internet sučelje na kojemu se nalazi digitalna verzija rasporeda nastave i ispita. Digitalni raspored nastave i ispita integralan je dio organizacije rada fakulteta jer omogućuje da se fakultetski prostor i vrijeme iskorištavaju na optimalan način. Aplikacije za organizaciju osobnog rasporeda, poput Google Calendara imaju veliki broj korisnika. One omogućuju praćenje događaja kako u poslovnom tako i u privatnom životu, olakšavaju planiranje i orijentaciju u vremenu. Cilj ovog završnog rada je izraditi poslužitelj koji će integrirati digitalni raspored fakulteta zajedno s tim aplikacijama, a konkretno je izabran Google Calendar jer je to jedna od najpopularnijih opcija među studentima i djelatnicima fakulteta. Ova aplikacija omogućit će korisnicima da podatke o svom rasporedu imaju u Google Calendaru zajedno sa svojim ostalim obvezama, a namijenjena je primarno studentima, ali i profesorima i ostalim suradnicima kojima je potreban raspored fakulteta. Digitalni raspored je dinamičan te se mijenja s vremenom i njegova finalna verzija za idući tjedan dolazi četvrtkom. Kada dođe finalna verzija rasporeda poslužitelj će pomoću XML (engl. *Extensible Markup Language*) aplikacijskog programskog sučelja FERIT-a dohvatiti podatke o rasporedu za sve profesore, smjerove i godine studija te unijeti u odgovarajuće kalendare korisnika iz baze podataka. Aplikacija će na taj način eliminirati potrebu za ručnim kopiranjem događaja iz FERIT rasporeda u Google Calendar, odnosno svi će se događaji u Google Calendar unositi automatski. Aplikacija će se pokretati u tri izvršna okruženja: razvojnom, testnom i produkcijskom. Razvojna okruženja bit će posluživana na Google Cloud platformi za računarstvo u oblaku. Nove verzije aplikacije će biti automatski ažurirane pomoću Gitlab CI/CD (engl. *Continuous Integration and Continuous Deployment*) alata nakon svake promjene u kodu, prateći Git Flow okvir rada za Git sustav kontrole verzija. Sva infrastruktura aplikacije bit će zapisana kao kod te će se pokretati u Docker kontejnerima.

1.1 Zadatak završnog rada

U teorijskom dijelu rada potrebno je proučiti i opisati tehnologije za izradu pozadinskih usluga (poput Javascript, NodeJS, Git, Docker, Gitlab CI/CD, MongoDB, Cloud). U praktičnom dijelu rada potrebno je izraditi pozadinsku uslugu (engl. BackEnd) te sustav kontinuirane integracije za tu uslugu. Pozadinska usluga treba se pomoću autentifikacije spajati na API za kalendar.

2. PREGLED KORIŠTENIH TEHNOLOGIJA I RAZVOJNIH PRINCIPA

2.1 Git distribuirani sustav za upravljanje izvornim kodom

Git je sustav kontrole verzija za praćenje promjena i za koordinaciju rada na bilo kojem skupu datoteka, a primarno se koristi za razvoj softvera. Git omogućuje verzioniranje koda te održavanje različitih inačica nekoga softvera, spremajući ih u takozvane Git repozitorije. Git sprema programski kod pohranjivanjem snimaka različitih verzija programskog koda u tzv. *commitove*. Jedan *commit* predstavlja stanje koda u nekom trenutku. *Svi commitovi* u sebi sadrže informacije o drugim *commitovima* što nam omogućuje praćenje promjena kroz vrijeme. Grane (engl. *branch*) predstavljaju različite verzije programskog koda koji postoje istovremeno. Više grana može se sjediniti zajedno (engl. *merge*), i tako postižemo primjenjivanje promjena programskog koda nastalih u jednoj grani na drugu granu [1]. Git također sadrži podmodule (engl. *submodule*). *Git* podmoduli predstavljaju mogućnost da se određeni Git repozitoriji nalaze jedni unutar drugih. Podmoduli se koriste kako se kod ne bi duplicirao između repozitorija te kako bi se odvojile odvojene jedinice koda.

Git je izdan 2005. godine, a razvio ga je Linus Torvalds za potrebe razvoja Linux jezgre. Git je u potpunosti otvorenog koda te je izdan pod GNU GPLv2 licencom. Git se danas koristi za razvoj većine svjetskih projekata otvorenog koda, a najveći svjetski poslužitelj Git repozitorija otvorenog koda, GitHub, sredinom 2017. godine brojio je 20 milijuna korisnika i preko 57 milijuna repozitorija [2].

2.2 Gitlab CI/CD sustav za kontinuiranu integraciju

Gitlab je internetski servis koji besplatno poslužuje Git repozitorije, zajedno sa skupom alata koji omogućuju i automatiziranje pokretanja aplikacije u različitim okruženjima. Takozvani Gitlab CI/CD je alat za istoimene CI i CD pristupe razvoju softvera [3]. CI je kratica za kontinuiranu integraciju (engl. *continuous integration*), pristup razvoju softvera koji zahtjeva da se sve promjene unutar programskog kôda uvijek i redovito integriraju u glavnu verziju softvera kako bi se osiguralo da su programski kôdovi uvijek kompatibilni te kako bi se izbjegla potreba za naglom integracijom velike količine koda. CD je kratica za kontinuiranu isporuku (engl. *continuous delivery*), pristup razvoju softvera koji zahtijeva da se softver razvija u kratkim razvojnim ciklusima te da je uvijek spreman za isporuku. Kontinuirana isporuka omogućuje brže iterativne promjene, a smanjuje rizike koji se pojavljuju uz isporuku velikih promjena. Kontinuirana isporuka zahtjeva striktan i ponovljiv proces isporuke.

2.3 Git Flow okvir rada za Git

Git Flow je okvir rada za Git [4]. Git Flow zahtjeva da svako izvršno okruženje aplikacije ima i svoju Git granu. Prilikom razvoja aplikacija agilnim pristupom postoje tri osnovna izvršna okruženja, a to su razvojno (engl. *development*), testno (engl. *testing*) i produkcijsko (engl. *production*). Svako izvršno okruženje ima i svoju istoimenu Git granu, a tako je uvijek poznato koja verzija koda se izvršava u kojem okruženju. Osim osnovnih Git grana postoje i dodatne grane svojstava (engl. *feature*) i grane brzog popravka (engl. *hotfix*). Grane svojstava proizlaze iz razvojne grane te se u njima razvijaju sva nova i veća svojstva aplikacije, a kada svojstva dostignu zadovoljavajuću razinu kvalitete ona se sjedinjuju s razvojnom granom. Grane brzog popravka proizlaze iz produkcijske grane i služe za popravak grešaka u kodu koje zahtijevaju hitnu intervenciju. Te grane se sjedinjuju s razvojnom granom te su navedene promjene odmah vidljive u produkcijskom okruženju.

2.4 Infrastruktura kao kod

Infrastruktura kao kod je princip upravljanja infrastrukturom pomoću opisnog modela, koji će uvijek stvoriti isto okruženje [5]. Prilikom razvoja aplikacije agilnim pristupom ona će se često pokretati u različitim okruženjima. Najčešća okruženja su razvojna, testna i produkcijska okruženjima. Ako broj korisnika aplikacije poraste, potrebno je i povećati količinu računalnih resursa dostupnih poslužitelju u produkcijskom okruženju. Također razni ekonomski, tehnički i legalni razlozi mogu ukazati na potrebu za promjenom opskrbljivača infrastrukture. Opisivanje infrastrukture kao kod ostvaruje paritet između konfiguracije različitih okruženja, te ubrzava proces prebacivanja aplikacije između različitih okruženja. Infrastruktura zapisana kao kod može se čuvati i u nekom od sustava za verzioniranje softvera što omogućuje njezin iterativni razvoj te pohranu povijesti promjena.

2.5 Docker platforma za virtualizaciju na razini operacijskog sustava

Docker je platforma koja omogućuje virtualizaciju na razini operacijskog sustava. Otvorenog je koda izdanog pod Apache 2.0 licencom [6]. Virtualizacija na razini operacijskog sustava još se naziva i kontejnerizacija. Kontejnerizacija se razlikuje od pune virtualizacije operacijskog sustava u tome što sve virtualizirane jedinice, zvane kontejneri, dijele iste resurse s domaćinskim operacijskim sustavom, što nije slučaj kod pune virtualizacije operacijskog sustava. Bitno je naglasiti da su svi kontejneri u potpunosti izolirane jedinice, iako međusobno i s domaćinom dijele zajedničke resurse. Docker je primarno razvijan za GNU Linux operacijske sustave te svaki

kontejner koristi Linux jezgra domaćinskog operacijskog sustava. Na Windows i MacOS operacijskim sustavima Docker se pokreće unutar virtualiziranog Linux operacijskog sustava, no taj je proces apstrahiran od krajnjeg korisnika.

Docker Compose je alat za definiranje i pokretanje aplikacija koje se sastoje od više kontejnera. Prije nego li je postao standard, aplikacije koje se sastoje od više kontejnera pokretale su se pomoću *bash* skripti [7]. Konfiguriranje aplikacije vrši se u takozvanoj *docker-compose.yml* datoteci koja se piše u YAML (engl. *YAML Ain't Markup Language*) formatu. Pokretanje aplikacije vrši se pomoću istoimenog alata za naredbeni prozor.

2.6 Google Cloud i Compute Engine

Google Cloud jedna je od najvećih platforma za računarstvo u oblaku te nudi široki raspon usluga. Osnovna usluga je Compute Engine koja omogućuje zakup virtualnih privatnih servera na Googleovoj infrastrukturi, a zakupljeni serveri mogu biti proizvoljnih karakteristika u vidu procesorske snage, radne memorije, veličine slobodnog prostora za pohranu podataka i pokretanog operacijskog sustava [8]. Googleovo internetsko sučelje omogućuje brz i jednostavan zakup servera uz nekoliko klikova na internetskom sjedištu ili pomoću internetskih zahtjeva koji se mogu poslati pomoću odgovarajućih biblioteka i konzolnih aplikacija.

2.7 Node.js izvedbeno okruženje i NPM upravitelj paketima

Node.js je izvedbeno okruženje otvorenog koda izdano pod MIT licencom namijenjeno za serversko skriptiranje u programskom jeziku JavaScript. Node.js nudi vrlo visoke performanse izvođenja, lagan je i efikasan te osim brzine izvođenja ima i veliku brzinu razvoja zbog činjenice da je JavaScript jezik s dinamičkim tipovima. Node.js je baziran na Googleovom V8 JavaScript izvršitelju visokih performansi. Node.js je također otvorenog koda te napisanom u programskom jeziku C++ [9].

Za svoj brz porast popularnosti kao platforma za razvoj pozadinskih (engl. *backend*) internetskih servisa, osim činjenice da se piše u najčešće korištenom jeziku za pisanje frontend servisa JavaScriptu, zaslužan je i njegov vrlo bogat eko sustav otvorenog koda baziran na NPM (engl. *node package manager*) upravitelju paketa. NPM omogućuje brzo i jednostavno dodavanje paketa, odnosno biblioteka. Svi podaci o paketima zapisuju se kao kod u datoteku *package.json.* Ova datoteka omogućuje lak i brz pregled, njihovo instalaciju, ažuriranje i mijenjanje verzija [10].

2.8 MongoDB baza podataka

MongoDB je NoSQL baza podataka, otvorenog je koda i u potpunosti besplatna te je licencirana pod GNU AGPLv3 licencom. MongoDB baza podataka ima dobre performanse, a izabrana je zbog svoje fleksibilnosti, jer za razliku od SQL (engl. *Structured Query Language*) baza podataka baziran na dokumentima (konkretno se radi o JSON (engl. *JavaScript Object Notation*) strukturi podataka) čija se struktura ne mora unaprijed definirati, što nudi veću brzinu razvoja i fleksibilnost za promjene [11].

2.9 OAuth2 protokol

OAuth2 je protokol koji omogućuje internetskim poslužiteljima ograničen pristup korisničkim računima nekog drugog internetskog poslužitelja. Najčešće se koristi za interakciju s internetskim poslužiteljima poput Facebooka, Githuba i Googlea. OAuth2 je dizajniran tako da autentifikaciju korisnika povjerava posebnom servisu koji čuva informacije o korisničkim računima te se korisnički resursi povjeravaju posebnom servisu koji samo čuva resurse [12]. Podaci mogu biti osobni podaci poput emaila, korisničkog imena i drugih osobnih podataka, a resursi mogu biti slike, video zapisi i ostali dokumenti koji pripadaju korisniku.

OAuth2 omogućuje autorizaciju servisima trećih strana da dobiju određeni djelokrug (engl. *scope*), odnosno pristup unaprijed definiranom skupu informacija i radnji nad korisničkim računima i resursima. Na razini protokola definiramo četiri uloge:

1. Vlasnik resursa (korisnik) - autorizira aplikaciju treće strane da dobije određeni djelokrug nad njegovim podacima i resursima.
2. Klijent - korisnička aplikacija koja će zatražiti pristup korisničkim podacima i resursima. Ovisno o djelokrugu ona ih može čitati ili i čitati i pisati.
3. Poslužitelj resursa i autorizacijski poslužitelj – poslužitelji imaju zadatak čuvati podatke i resurse koji pripadaju korisnicima, aplikacije koje žele pristupiti korisničkim podacima i resursima to mogu učiniti pomoću jednoga aplikacijskog programskog sučelja koje to omogućuje.

OAuth2 protokol omogućuje pristup zaštićenim resursima jedino autentificiranim klijentima. Klijenti svoj identitet potvrđuju pomoću pristupnog žetona (engl. *access token*). Pristupni žeton dobiva se sljedećim nizom koraka:

1. Klijent zatražuje pravo od korisnika, klikom unutar klijentske aplikacije otvorit će se poseban prozor unutar internetskog preglednika koji poslužuje autorizacijski poslužitelj, korisnik će se autentificirati te odobriti zahtjev.
2. Ako vlasnik resursa odobri zahtjev, klijentskoj aplikaciji se šalje odobrenje (engl. *grant*), te se korisnika preusmjerava nazad na klijentsku aplikaciju.
3. Klijentska aplikacija će u pozadini pomoću aplikacijskog programskog sučelja poslati odobrenje te nazad dobiti pristupni žeton.

2.10 Shell skripte

Shell skripte su tekstualne datoteke koje sadrže niz naredbi namijenjenih za izvršavanje u Linux okruženju, a primarno se koriste za automatiziranje zadataka poput pomicanja, izmjene, brisanja i kreiranja datoteka te pokretanje naredbi na udaljenim poslužiteljima [13]. Shell skripte omogućuju pokretanje različitih naredbi poput Docker naredbi.

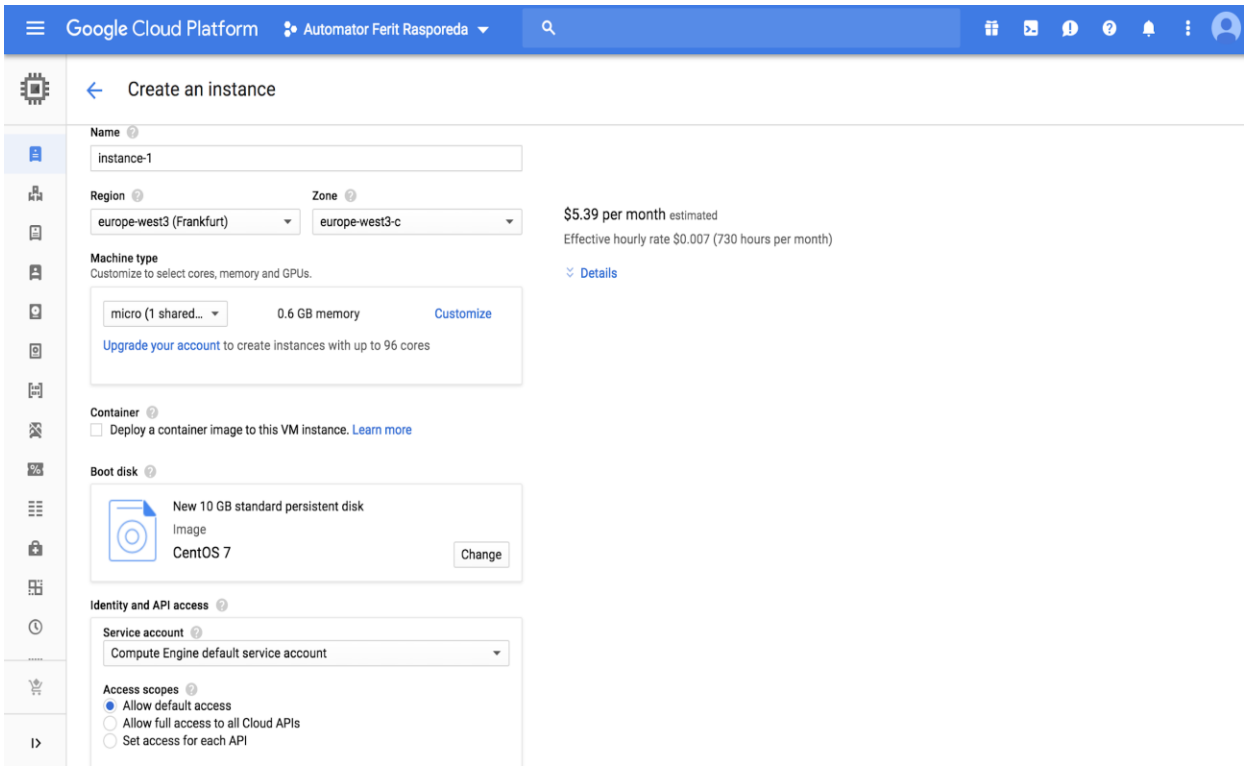
3. RAZVOJ INFRASTRUKTURE I DEFINIRANJE RAZVOJNOG PROCESA

3.1 Željeni razvojni proces

Prije početka razvoja potrebno je definirati razvojni proces i postaviti potrebnu infrastrukturu. Infrastruktura će se sastojati od tri poslužitelja zakupljena na Google Cloud platformi za računarstvo u oblaku koji će pokretati osnovnu instalaciju CentOS operacijskog sustava s instaliranim Dockerom. Na svakom od poslužitelja pokretat će se jedno od tri izvedbena okruženja: razvojno, testno ili produkcijsko. Svako okruženje imat će odgovarajuću Git granu: *develop* grana za razvojno okruženje, *release* grana za testno okruženje i *master* grana za produkcijsko okruženje. Te će grane pratiti Git Flow okvir rada dok će se pomoću Gitlab CI/CD alata svako okruženja automatski ažurirati prema najnovijim promjenama u ogovarajućoj grani. Samo izvedbeno okruženje aplikacije bit će pokrenuto unutar Docker kontejnera te će svi ostali potrebni servisi biti unutar vlastitih kontejnera. Konkretno se radi o kontejneru koji pokreće MongoDB bazu podataka i kontejneru koji pokreće Mongo-express. Mongo-express je sučelje za izravan pristup bazi. U slučaju da aplikacija dobije nove zahtjeve i ona bi također mogla biti razbijena u manje servise od kojih bi se svaki pokretao u zasebnom Docker kontejneru.

3.2 Zakup infrastrukture na Google Cloudu

Zakupljivanje virtualnih privatnih servera na Google Cloudu jednostavan je proces. Potrebno je posjetiti Google Compute Engine odjeljak internetskog sjedišta Google Clouda te popuniti jednostavan formular koji je prikazan na Slici 3.1. Nakon podnošenja zahtjeva virtualni server će se u roku od nekoliko minuta podignuti te postati dsotupan na javnoj internetskoj mreži. Za potrebe ove aplikacije izabran je „*micro*“ tip virtualnog servera. Ovaj tip virtualnog servera sadrži 0,6 GB radne memorije i jedan dijeljeni procesor. Dijeljene procesore istovremeno koristi više virtualnih servera, a svakom virtualnom serveru garantiran je određen podskup taktova procesora. Virtualni serveri imaju pravo povremeno koristiti cijeli procesor, ali samo u kratkim naletima kako bi se osiguralo da i ostali serveri koji dijele računalne resurse imaju dovoljan broj raspoloživih taktova. Pošto više servera dijeli isti procesor, cijena servera je manja, no ipak je povremeno moguće koristiti punu snagu procesora. Cijena ovog servera je $5.39 za 730 sati rada (otprilike 1 mjesec). U ovu cijenu nije uključen mrežni promet i prostor na virtualnom tvrdom disku, no njihova cijena je neznatna ili čak i besplatna za male aplikacije. U daljnjem tekstu slijedi slika 3.1 na kojoj je prikazano sučelje Google Clouda za zakup virtualnih privatnih servera.



Slika 3.1 - Google Compute Engine sučelje za zakup privatnih virtualnih servera.

Pri popunjavanju formulara za zakup prvo je potrebno odabrati ime servera, regiju i zonu unutar te regije u kojoj će se nalaziti virtualni server. Regija će odrediti geografsku lokaciju servera, u ovome slučaju Frankurt u Njemačkoj. Zona određuje računalni centar unutar regije u kojemu će se nalaziti server, ali za krajnje korisnike nema razlike izmedju različitih zona osim pri konfiguraciji visoko dostupnih aplikacija koje zahtjevaju korištenje nekoliko računalnih centara. Ime servera sluzi isključivo za identifikaciju servera na internetskom sjedištu Google Clouda. Također, potrebno je odabrati tip servera. Tip servera odrediti će procesorsku snagu i dostupnu radnu memoriju servera te je u ovom slučaju izabran već ranije objašnjen „*micro*“ tip.

Za glavni disk virtualnog servera izabrana je opcija 10 GB standardnog perzistentnog diska s instaliranim CentOS operacijskim sustavom. Standardni perzistentni disk pogonjen je SSD (engl. Solid State Drive) diskovima za pohranu podataka te nudi dobre performanse za obične web aplikacije i baze podataka. Postavke servisnog računa ostavljene su na zadanim vrijednostima pošto virtualni server neće komunicirati s drugim servisima Google Clouda. Zadane vrijednosti ne dopušaju serveru da sam pravi nove resurse te čita podatke drugih resursa. Nakon zakupa jednog servera na njega je instaliran Docker te je server kloniran u još dva primjerka kako bi postojala sveukupno tri identična izvedbena okruženja za aplikaciju.

3.3 Docker tehnologija virtualizacije na razini operacijskog sustava

Docker tehnologija nudi vrlo jednostavan način da se unutar takozvanog Dockerfilea imperativno definira niz naredbi koje će oblikovati izvršno okruženje aplikacije. Konkretno to uključuje aplikaciju i njezine ovisnosti. Iz Dockerfilea se može napraviti slika, a instance te slike nazivamo kontejnerima [14]. Docker slike nisu ništa drugo nego komprimirane mape koje sadrže sve binarne datoteke potrebne za pokretanje aplikacije, a kontejneri su procesi koji koriste te binarne datoteke definirane u slici za svoje izvođenje. Kontejner je u potpunosti izolirano izvršno okruženje, odvojeno od ostalih izvršnih okruženja koja se nalaze na istom poslužitelju. U Programskom kodu 3.1 vidljiva je Dockerfile datoteka ove aplikacije.

```
FROM node:alpine

ENV NODE_ENV production

WORKDIR /usr/src/app

COPY ["package.json", "package-lock.json*", "npm-shrinkwrap.json*", "./"]
RUN npm install --production --silent && mv node_modules ../

COPY . .
EXPOSE 80

CMD npm start
```

Programski kod 3.1 – Datoteka Dockerfile za izradu slike ove aplikacije.

Dockerfile (Programski kod 3.1) počinje s naredbom *FROM* koja definira od koje početne slike će nastati slika koju definiramo. *FROM* naredba omogućuje nasljeđivanje među slikama te se velika baza takvih slika može pronaći u repozitoriju zvanom DockerHub. Osnovna slika ove aplikacije kao svoju početnu sliku uzet će "*node:9.11*". To je slika bazirana na Debian 8 GNU LINUX operacijskom sustavu i na sebi ima instaliran Node.js verzije 9.11 te NPM upravitelj paketa. Ako bi postojala potreba za promjenom verzije Node.js-a, na Dockerhubu, u vrijeme pisanja ovoga rada, postoje i sve kombinacije slika baziranih na: Alpine Linuxu, Debian 8 i 9 Linuxu te Node.js verzijama 6,8,9 i *unstable* 10 [15].

U kodu 3.1 u liniji 3 korištena je naredba *ENV* koja će unutar slike podesiti varijablu izvedbenog okruženja imena *NODE_ENV* i predati joj vrijednost "*production*". U liniji 4 korištena je naredba

WORKDIR koja će promijeniti trenutnu radnu mapu unutar slike za potrebe ostalih naredbi te će to biti i defaultni direktorij kada se u Docker kontejner spaja sa SSH (engl. *Secure Shell*). U liniji 6 korištena je naredba *COPY* koja će datoteke "*package.json*", "*package-lock.json**", "*npm-shrinkwrap.json**" prebaciti u trenutni radni direktorij. U liniji 7 korištena je naredba *RUN* koja unutar slike pokreće naredbu koja joj je predana, konkretno je predana naredba "*npm install*" s dodatnim opcijama "*production*" i "silent". Ova naredba će na temelju ranije prebačenih json datoteka instalirati sve potrebne ovisnosti pomoću *npm* package managera, a dodatne opcije korištene su zbog toga što ubrzavaju proces instalacije. Također je uz pomoć naredbe "*mv*" promijenjena lokacija "*node_modules*" u prethodnu mapu.

3.4 Docker Compose alat za konfiguraciju Dockera

Docker Compose je konzolna aplikacija koja omogućuje jednostavnu konfiguraciju serverskih aplikacija koje se sastoje od više kontejnera. Docker Compose omogućuje da definiramo unaprijed poznatu konfiguraciju za umrežavanje, putanje dijeljenih diskovnih datoteka, varijable izvedbenog okruženja itd. To se sve zapisuje u takozvanoj *docker-compose.yml* datoteci u YAML formatu[16].

Unutar YAML datoteke definiramo listu s četiri elementa:

1. *version* u kojoj definiramo verziju sintakse *docker-compose.yml* datoteke koja je korištena, u ovoj datoteci radi se o verziji 2.
2. *services* u kojoj se definiraju svi kontejneri od kojih se aplikacija sastoji.
3. *networks* u kojoj definiramo sve virtualne mreže u koje će preko tcp/ip protokola biti povezani kontejneri, u ovoj datoteci definirana je samo jedna mreža pod nazivom "*backend*".
4. *volumes* u kojoj definiramo sve volumene, odnosno točke u kojima će datotečni sustavi kontejnera imati doticaj s datotečnim sustavom domaćinskog operacijskog sustava, u ovoj datoteci definiran je samo jedan volumen pod nazivom "*database*".

Unutar elementa *services* definirana je lista kontejnera od kojih se sastoji aplikacija. Za ovu aplikaciju potrebna su tri kontejnera. To su automator_rasporeda, koji sadrži Node.js aplikaciju opisanu u ovom radu, zajedno sa njenim ovisnostima, koji smo definirali kroz *Dockerfile*. Mongo kontejner koji sadrži bazu podataka i njenu konfiguraciju, te Mongo-express kontejner koji sadrži istoimeno grafičko sučelje za upravljanje bazom podataka. U nastavku slijedi programski kod 3.2 koji prikazuje datoteku *docker-compose.yml* ove aplikacije.

```yaml
version: '2'
services:
  automator_rasporeda:
    image: automator_rasporeda
    build:
      context: .
      dockerfile: Dockerfile.dev
    volumes:
      - .:/usr/src/app:ro
    environment:
      NODE_ENV: development
    ports: ${PORT}:${PORT}
    depends_on:
        - mongo
    networks:
      backend:
  mongo:
    image: mongo:3.7-jessie
    restart: always
    environment:
      MONGO_LOG_DIR: "/data/log"
    volumes:
      - database:/data/db
    expose: "27017"
    networks:
      backend:
        aliases:
          - ${MONGO_NETWORK_ALIAS}
  mongo-express:
    image: mongo-express
    restart: always
    ports: "8081:8081"
    networks:
      - backend
    environment:
      ME_CONFIG_MONGODB_SERVER: ${MONGO_NETWORK_ALIAS}
    depends_on:
      - mongo
      - automator_rasporeda
    command: tini -- bash -c 'sleep 5 && node app'
networks:
  backend:
volumes:
  node_modules:
  database:
```

Programski kod 3.2 – Datoteka docker-compose.yml za opisivanje izvedbenog okruženja.

Svaki kontejner definiran je listom čiji elementi mogu biti sljedeći:

1. *Image* je ime slike kontejnera, u slučaju da je definiran i element *build* onda je to ime koje će biti dopridjeljeno slici nakon što se ona sagradi.
2. *Build* predstavlja putanju do *Dockerfilea* od kojega će se praviti slika.
3. *Restart* je politika ponovnog pokretanja kontejnera u slučaju greške.
4. *Ports* i *expose* elementi definiraju mapiranje portova unutar kontejnera na portove operacijskog sustava.
5. *Environment* je element u kojemu se definira lista svih varijabli izvedbenog okruženja koje trebaju biti definirane unutar kontejnera.
6. *Depends_on* je element u kojem se definirana lista svih kontejnera koji moraju biti pokrenuti kako bi taj kontejner pravilno radio.
7. *Networks* element je lista svih mreža na kojima kontejner mora biti vidljiv, svakom elementu (odnosno mreži) možemo pridružiti listu *aliases* u kojoj ćemo definirati *poslužiteljsko ime* pod kojim će taj kontejner biti vidljiv na mreži.

3.5 Pokretanje više razvojnih okruženja pomoću Gitlab CI/CDa

Gitlab omogućuje automatsko pokretanje aplikacije u više programskih okruženja prateći Git Flow. U Programskom kodu 3.3 slijedi *gitlab-ci.yml* datoteka u kojoj je definirano kako će Gitlab alat izgraditi Docker slike te kako će ih pokretati u raznim okruženjima. U priloženom kodu je vidljivo jedino razvojno (engl. *development*) okruženje, no ostala okruženja su identična, jedino je riječ "*development*" zamijenjena riječju "*testing*" odnosno "*production*". U prvoj liniji *gitlab-ci.yml* datoteke (Programski kod 3.3) definirana je Docker slika u čijem kontejneru će se izvršavati daljnje naredbe, Dockerfile te slike prikazan je u Programskom kodu 3.4. U trećoj liniji *gitlab-ci.yml datoteke* (Programski kod 3.3) definirane su Docker slike koje su dostupne preko HTTP (engl. *Hypertext Transfer Protokol*) protokola osnovnom kontejneru u kojem se izvršavaju sve naredbe. U ovoj datoteci radi se jedino o slici *docker:dind*, odnosno "*Docker in Docker*". Kontejneri te slike unutar sebe imaju vlastiti Docker pozadinski servis te ga poslužuju preko http protokola. Takozvana „*dind*" slika koristi se kako bi svaki gitlab build bio pokrenut u vlastitom izoliranom okruženju (kontejneru), ali i s pristupom osnovnim Docker naredbama kako bi se mogle graditi slike. Na šestoj liniji ove datoteke definirane su etape automatiziranog CI/CD procesa.

Definirane su dvije etape, *build* i *deploy*. Etapa *build* je etapa u kojoj se izgrađuju Docker slike, dok je *etapa* deploy ona u kojoj se pokreću kontejneri datih slika u nekom od izvršnih okruženja.

Na desetoj liniji definirane su interne gitlab varijable kojima se može mijenjati rad CI/CD alata. Varijabla izvedenog okruženja GIT_SUBMODULE_STRATEGY postavljena je na vrijednost "normal". Vrijednost ove varijable govori Gitlab CI/CD alatu da uz Git repozitorij preuzme i sve njegove submodule.

```
image: htonkovac/docker-runner-image

services:
  - docker:dind

stages:
  - build
  - deploy

variables:
  GIT_SUBMODULE_STRATEGY: normal

build_docker_images:
  stage: build
  before_script:
    - docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD" $CI_REGISTRY
  script:
    - ls -d */ |  xargs -I@ /bin/sh -c "./docker-build-and-push.sh @"

development:
  stage: deploy
  before_script:
    - chmod +x ./setup-ssh.sh
    - ./setup-ssh.sh
  script:
    - chmod +x docker-deploy.sh
    - ./docker-deploy.sh $DEPLOYMENT_IP_DEVELOPMENT
  environment:
    name: development
    url: http://$DEPLOYMENT_IP_DEVELOPMENT
  only:
    - development
```

Programski kod 3.3 – Datoteka gitlab-ci.yml za konfiguriranje GitlabCI/CD-a.

U liniji 14 *gitlab-ci.yml* datoteke (Programski kod 3.3) definiran je jedan od zadataka za CI/CD alat. Zadatak je imena *build_docker_images* te pripada etapi *build*. Prije njegovog izvršavanja pokrenut će se *docker login* naredba kako bi se obavila prijava u Docker registar u kojemu će se spremati Docker slike. Glavna skripta ovog zadatka izlistat će imena svih datoteka te ih predati naredbi *xargs* koja će jedno po jedno ime datoteke predati skripti *docker-build-and-push.sh*.

Skripta *docker-build-and-push.sh* bit će prikazana u Programskom kodu 3.6. U liniji 20 definiran je zadatak pod imenom *development* , a u daljnjim linijama definirani su i zadaci *testing* i *production*, koji imaju istu implementaciju. Zadatak *development* pripada etapi deploy, a prije njegovog izvršavanja pokrenut će se naredba *chmod +x setup-ssh.sh* koja će nam dati prava na pokretanje skripte *setup-ssh.sh* te će se pokrenuti skripta setup-ssh.sh, skripta *setup-ssh.sh* će biti prikazana u Programskom kodu 3.6. U glavnom dijelu zadatka u prvom koraku izvršit će se naredba *chmod +x docker-deploy.sh.* Ona će nam omogućiti izvršavanje skripte *docker-deploy.sh*. U drugom koraku glavnog dijela zadatka pokreće se skripta *docker-deploy.sh* koja će biti prikazana u Programskom kodu 3.7. U daljnjem tekstu slijedi Programski kod 3.4 koji prikazuje Dockerfile datoteku od koje je načinjena slika *docker-runner-image*.

```
FROM docker:latest
RUN apk add openssh-client gettext
```

Programski kod 3.4 – Datoteka Dockerfile za izradu slike u kojoj ce se pokretati naredbe GitlabCI/CD-a.

Radi se o slici baziranoj na slici *docker:latest* koja sadrži osnovno Docker konzolno sučelje te su instalirani dodatni paketi *openssh-client* koji omogućuje povezivanje putem SSH protokola i *gettext* koji omogućuje korištenje naredbe *envsubst* koja omogućuje zamjenu varijabli unutar neke datoteke. U daljnjem tekstu slijedi Programski kod 3.5 na kojoj je prikaz skripte docker-build-and-push.sh.

```
mkdir -p ~/.ssh
echo "$GC_SSH_KEY" | tr -d '\r' > ~/.ssh/id_rsa
chmod 600 ~/.ssh/id_rsa
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_rsa
ssh-keyscan -H $DEPLOYMENT_IP_DEVELOPMENT >> ~/.ssh/known_hosts
ssh-keyscan -H $DEPLOYMENT_IP_TESTING >> ~/.ssh/known_hosts
ssh-keyscan -H $DEPLOYMENT_IP_PRODUCTION >> ~/.ssh/known_hosts
```

Programski kod 3.5 – Skripta setup-ssh.sh za postavljanje konfiguracije za rad sa SSH protokolom.

Skripti *docker-build-and-push.sh* bit će predano ime direktorija u kojemu se nalazi *Dockerfile*, skripta će pomoću naredbe *cd* ući u predani joj direktorij, definirat će se novo ime slike bazirano na osnovnom imenu slike s dodanim imenom direktorija. Slijedi naredba echo koja će ispisati sadržaj direktorija radi kontrole u outputu gitlab internetskog sučelja. Naredba „*docker build*“ izgradit će Docker sliku, predane su joj sljedeće opcije: *--cache-from* koja govori koja slika bi mogla sadržavati iste slojeve kao slika koja se trenutno gradi kako bi se ubrzao proces izgradnje,

--*pull* opcija koja govori Docker procesu da s interneta povuče najnovije verzije raspoložive osnovne slike, te *-t* opcije pomoću kojih postavljamo oznaku (engl. *tag*) slike. Posljednje dvije naredbe *"docker push"* će sliku označenu nekom oznakom poslati u registar, kako bi je se moglo povući u daljnjim etapama te nadolazećim izgradnjama za svrhu ponovnog korištenja istih slojeva. U daljnjem tekstu slijedi Programski kod 3.6 koji prikazuje skriptu setup-ssh.sh.

Ova skripta će kreirati skriveni direktorij *.ssh* unutar *home* direktorija *root* korisnika, unutar datoteke *id_rsa* zapisat će SSH privatni ključ za spajanje na poslužitelj na kojemu će se pokretati izvršno okruženje aplikacije. Naredba *chmod* promijenit će prava te datoteke kako bi samo vlasnik mogao čitati i pisati po njoj, pošto se radi o privatnom ključu, u suprotnom *ssh* klijent odbit će koristiti datu datoteku za svrhu spajanja na server. Naredba *eval "$(ssh-agent -s)"* pokrenut će novi proces koji pamti *ssh* tajne ključeve kako ih ne bi morali predavati kao opciju svakoj naredbi koja se povezuje na poslužitelj, slijedi naredba *ssh-add* kojom ćemo *ssh-agent* procesu predati tajni ključ koji treba zapamtiti. Na kraju slijede tri *ssh-keyscan* naredbe koje će kontaktirati servere na datim IP adresama te od njih pokupiti informacije o njihovim javnim SSH ključevima. Zatim će se ti podaci o javnim ključevima nadodati na kraj datoteke *known_hosts* koja sadrži parove adresa i javnih ključeva poslužitelja čije adrese se smatraju sigurnim za spajanje. U nastavku slijedi programski kod 3.7 koja prikazuje skriptu *docker-deploy.sh*.

```
envsubst < environment.env > .env
scp docker-compose.yml .env gitlab@$1:~
ssh gitlab@$1 "docker-compose pull; docker-compose down; docker-compose up -d; rm
.env"
```

Programski kod 3.6 – Skripta docker-deploy.sh za pokretanje docker kontejnera na udaljenom serveru.

Prvo se izvršava naredba *envsubst* koja će na standardni ulaz primiti sadržaj datoteke *environment.env* te u sadržaju zamijeniti imena varijabli izvedbenog okruženja s njihovim trenutnim vrijednostima u ljusci operacijskog sustava. Standardni izlaz ove naredbe je nova datoteka imena *.env*. Druga naredba će *.env* i *docker-compose.yml* datoteke prebaciti na server s IP adresom predanoj kao prvi parametar *bash* skripte. Naredba *scp* koristit će privatni ključ kojeg je zapamtio proces *ssh-agent* u skripti *setup-ssh.sh*. Posljednja naredba, *ssh*, povezat će se na isti server te izvršiti sljedeće naredbe: "*docker-compose down*" koja će zaustaviti sve Docker kontejnere pokrenute od prijašnjih pokrenutih zadataka, "*docker-compose up*" koja će pokrenuti Docker kontejnere definirane u novoj *docker-compose* datoteci, te naredbu "*rm .env*" koja će obrisati *.env* datoteku nakon pokretanja Docker kontejnera kako bi varijable koje se u njoj nalaze ostale tajne.

4. RAZVOJ I KORIŠTENJE APLIKACIJE

4.1 Dohvaćanje rasporeda pomoću FERIT XML aplikacijskog programskog sučelja

Kako bi se podaci o rasporedu unjeli u Google Calendar potrebno ih je dohvatiti preko FERIT XML aplikacijskog programskog sučelja. U nastavku slijedi Programski kod 4.1 koja prikazuje funkciju *fetchCalendarData*. Funkcija *fetchCalendarData* prima parametar *date* koji predstavlja datum u formatu GGGG-MM-DD. Ova funkcija će dohvatiti kompletni FERIT raspored za jedan dan u XML formatu tako što će pomoću modula *http* poslati HTTP GET zahtjev na odgovarajuću pristupnu točku FERIT XML aplikacijskog programskog sučelja. Dobiveni rezultat biti će u XML formatu te će ih funkcija parsirati u JSON format pomoću modula *xmlParser* radi lakše daljnje obrade.

```
function fetchCalendarData(date) {
  var calendarDataXML = http.get('https://www.ferit.unios.hr/raspored/' + date +
'.xml'
  var calendarDataJSON = xmlParser.toJson(calendarData);

  return calendarDataJSON;
}
```

Programski kod 4.1 - Funkcija fetchCalendarData koja dohvaća podatke o kalendaru koristeći FERIT XML API.

4.2 OAuth2 autentifikacija sa Google aplikacijskim programskim sučeljem

Ova aplikacija koristi OAuth2 autentifikaciju jer je on a primarni način za pristup Google aplikacijskim programskim sučeljima. Google je objavio *npm* modul otvorenog koda koji pruža sučelje koje apstrahira obavljanje svih mrežnih poziva koji su potrebni kako bi se izveo proces autentifikacije. Možemo ga instalirati koristeći naredbu: "*npm install -s google-auth-library*". Najlakši način za autentifikaciju je da napišemo funkciju omotač oko danog modula. Ta funkcija primat će dva parametra, *credentials* i *callback*. *Credentials* je JSON objekt koji nam pruža Google prilikom registracije aplikacije u Googleovom internetskom sjedištu, a sadrži tajne ključeve. Predat ćemo ga aplikaciji pomoću varijable izvedbenog okruženja kako bismo omogućili njegovu laku zamjenu. *Callback* je funkcija povratnog poziva, ona će se pozvati s predanim parametrom *oauth2client*, objektom koji omogućuje komunikaciju s Google aplikacijskim programskim sučeljem. Ako neka funkcija mora komunicirati s Google aplikacijskim programskim sučeljem, nju ćemo pozvati tako da pozovemo funkciju *authorize* i njoj predamo funkciju koji želimo pozvati

kao parametar *callback*, odnosno funkciju povratnog poziva. U nastavku slijedi Programski kod 4.2 u kojemu je prikazana implementacija funkcije *authorize*.

```
function authorize(credentials, callback) {
  var clientSecret = credentials.web.client_secret;
  var clientId = credentials.web.client_id;

  var redirectUrl = credentials.web.redirect_uris[0];
  var auth = new googleAuth();
  var oauth2Client = new auth.OAuth2(clientId, clientSecret, redirectUrl);

  callback(oauth2Client)
}
```

Programski kod 4.2 - Funkcija authorize koja koja autorizira aplikaciju pri Google aplikacijskom programskom sučelju.

4.3 Dodavanje jednostavnih događaja u Google Calendar

U Programskom kodu 4.3 slijedi implementacija funkcije *insertWelcomeEvent* za dodavanje jednostavnog događaja u kalendar.

```
function insertWelcomeEvent(student) {
  time = new Date();
  time2 = new Date()
  time2.setHours(time.getHours() + 1);
  authorize(clientSecret, (oauth2Client) => {
    oauth2Client.credentials = student.tokens;
    calendarUpdater.addEventsToCalendar(oauth2Client, [{
      'summary': 'Automator Ferit Rasporeda je aktiviran!',
      'description': 'Automator Ferit Rasporeda je aktiviran!',
      'start': {
        'dateTime': time.toISOString(),
        'timeZone': 'America/Los_Angeles',
      },
      'end': {
        'dateTime': time2.toISOString(),
        'timeZone': 'America/Los_Angeles',
      },
      'reminders': {
        'useDefault': true
      },
      'colorId': 9
    }])
  });
}
```

Programski kod 4.3 - Funkcija insertWelcomeEvent koja unosi događaj dobrodošlice u kalendar.

Dodavanje događaja u kalendar vrši se pomoću metoda definiranih preko Google aplikacijskog sučelja. Kako bi se do događaj unio u kalendar potrebno je poslati ispravan JSON objekt koji sadrži informacije o događaju na odgovarajuću adresu Google aplikacijskog programskog sučelja. Potrebne pozive može obaviti funkcija *addEventsToCalendar* iz *calendarUpdater* modula za rad s Google Calendarom.

4.4 Algoritam eksponencijalne odgode

Google aplikacijsko programsko sučelje je besplatno uz određena ograničenja. Jedno od tih ograničenja je i ograničenje prosječnog broja zahtjeva u sekundi na 100 zahtjeva u sekundi. Zbog svoje asinkrone prirode, Node.js može postići i puno veći broj zahtjeva u sekundi. Kako bi se postigla optimalna brzina unosa što bliža 100 zahtjeva u sekundi potrebno je koristiti algoritam eksponencijalne odgode [17]. Odgoda mora biti eksponencijalno rastuće prirode kako bi se izbjeglo bespotrebno slanje zahtjeva koji će zasigurno biti odbijeni te stvarati nepotrebne zastoje na mreži i trošenje računalnih resursa servera. Odstup treba rasti eksponencijalno, ali mu je potrebno i dodati nasumičnu komponentu. Nasumična komponenta dodaje se kako bi s izbjegla situacija u kojoj veliki broj zahtjeva dolazi u valovima.

```
function exponentialBackoff(err, event, calendar, auth, delay = 1) {
  if (delay < 100) {
    delay = delay * 2

    setTimeout(() => {
      calendar.events
      .insert({
        auth: auth,
        calendarId: 'primary',
        resource: event,
       }, (err, evnt) => {
        if (err == null) {
          console.log('%s: Event created with exponentialBackoff:'+delay+': %s',
           (new Date()).toISOString(), evnt.htmlLink);
        } else {
          exponentialBackoff(err, event, calendar, auth, delay)
        }
      })

    }, 500 * delay + getRandomTimeInMiliseconds(1000);
    return;
  }
}
```

Programski kod 4.4 – Programski kod algoritma eksponencijalne odgode.

U slučaju da Google aplikacijsko programsko sučelje odbije zahtjev, ono će vratit HTTP statusni kod „*403 Forbidden*“. Također može doći do greške na mreži, što će rezultirati greškom *socket timed out*. U oba slučaja potrebno je ponoviti zahtjev. Algoritam prikazan u Programskom kodu 4.4 implementiran je pomoću rekurzivne funkcije. Ako unošenje događaja u kalendar ne uspije, funkcija na slici bit će pozvana kako bi ponovno pokušala unijeti događaj u kalendar. Ako zahtjev bude neuspješan funkcija će rekurzivno pozivati samu sebe s eksponencijalno rastućim parametrom *delay* koji će odgađati ponovni pokušaj unosa događaja u kalendar. Ovo će se događati sve dok događaj ne bude unesen u kalendar ili odgoda postane predugačka.

U ovoj implementaciji algoritma, zahtjev će se ponoviti najviše sedam puta. Pokušaji će redom imati odgodu: 1000 do 2000 milisekundi, 2000 do 3000 milisekundi, 4000 do 5000 milisekundi, 8000 do 9000 milisekundi, 16000 do 17000 milisekundi i 32000 do 33000 milisekundi. Ukoliko i zahtjev s najvećom mogućom odgodom ne uspije pretpostavlja se da je došlo do trajne greške te se zahtjev više neće ponavljati.

4.5 Spremanje podataka o studentima u MongoDB bazu

Kako bismo studentima unosili podatke u kalendar potrebno je spremiti neke osnovne informacije o njima u bazu podataka. MongoDB je baza podataka koja omogućuje jednostavno spremanje podataka u JSON formatu, te ne zahtijeva prethodno definiranje sheme. JSON objekti u bazu se mogu spremiti jednostavnim pozivanjem funkcije *insertOne* nad nekom kolekcijom. Kolekcija predstavlja polje JSON objekata i nije ju potrebno prethodno inicijalizirati. U ovom slučaju odabrana je kolekcija *students*.

```
function storeStudentInDB(student) {

  MongoClient.connect(url, function (err, db) {
    if (err) console.error(err);

    db.collection("students").insertOne(student, function (err, res) {

      if (err) console.error(err);
      console.log("1 new  student");

      insertWelcomeEvent(student);
      db.close();
    });
  });
}
```

Programski kod 4.5 – Funckija storeStudentInDB koja vrši spremanje studenata u bazu.

U Programskom kodu 4.5 prikazana je funkcija *storeStudentInDB* koja sprema JSON objekt jednog studenta u bazu. Na početku funkcije, funkcija se spaja na MongoDB bazu podataka pozivajući *connect* funkciju modula *MongoClient*. Funkciji *connect* kao parametar predana je anonimna funkcija koja će pozvati *insertOne* funkciju i unijeti podatke o studentima. Na kraju pozivaju se funkcije *InsertWelcomeEvent (Programski kod 4.3)* i funkcija *db.close* koja će zatvoriti konekciju s bazom.

4.6 Dohvaćanje svih studenata iz baze

U Programskom kodu 4.6 slijedi funkcija koja dohvaća sve studente iz baze.

```
function getStudentsFromDBAsync(query) {

  return new Promise(
  (resolve, reject) => {

    MongoClient.connect(url,
      (err, db) => {

        if (err) reject(err);
        db.collection("students")
           .find(query)
          .toArray((err, students) => {

            if (err) reject(err);
             return resolve(students);
            db.close();
          });
      });
  });
}
```

Programski kod 4.6 – Funckija getStudentsFromDBAsync koja dohvaća studentea iz baze podataka.

Dohvaćanje podataka iz MongoDB baze je jednostavno, potrebno je pozvati funkciju *find()* te joj predati parametar *query*, koji će u pravilu biti prazan objekt kako bi se dohvatili svi studenti. Funkcija u kodu omotala je funkciju za dohvaćanje studenata u *Promise* objekt kako bi se olakšalo baratanje rekurzivnim pozivima nad rezultatom iz baze. *Promise* objekti su jednostavni objekti koji imaju zadaću izvršiti neki asinkroni poziv. Kao parametar primaju funkciju koja vrši neki asinkroni poziv. Ta funkcija, koja se predaje kao parametar, prihvaća točno dva parametra. Prvi parametar, *resolve*, predstavlja funkciju koja će biti izvršena u slučaju da asinkorni poziv uspije.

Drugi parametre, *reject*, predstavlja funkciju koja će biti izvršena ako asinkroni poziv iz nekog razloga rezultira dizanjem iznimke.

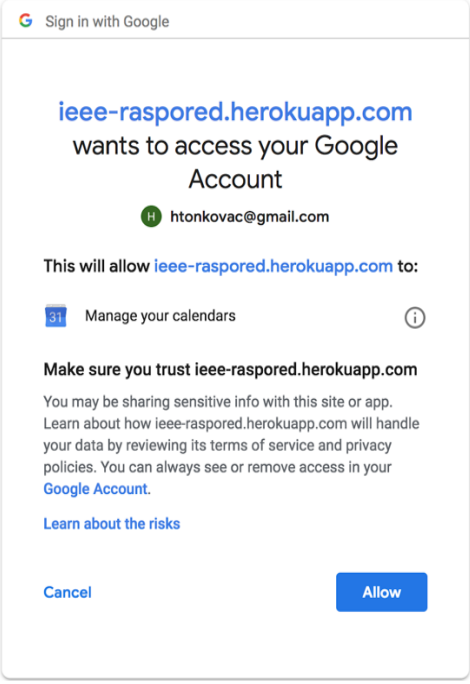
4.7 Korištenje aplikacije

Produkcijsko okruženje aplikacije nalazi se na sljedećem linku: http://35.198.75.154/. Prezentacijski sloj ove aplikacije relativno je jednostavan i služi samo za aktivaciju aplikacije. Prezentacijski sloj se sastoji od jednog jednostavnog obrasca za unos osobnih podataka o korisniku. Ovi podaci spremiti će se u MongoDB bazu podataka. Podaci su potrebni kako bi se mogli unijeti točni podaci o rasporedu, a ti su podaci: ime, prezime, smjer i godinu. Obrazac je vidljiv na Slici 4.1. Unošenje podataka o imenu i prezimenu ostvaruje se slobodnim tekstualnim unosom dok su podaci o smjeru i godini dostupni u padajućim izbornicima.

Slika 4.1 - Obrazac za prijavu.

Pritiskom na tamno plavi gumb „I ja želim automatizirati raspored!“ korisnik će završiti registraciju te će njegovi podaci biti uneseni u bazu podataka, no kako bi se zahtjevi na Googleovo programsko sučelje mogli vršiti u ime registriranog korisnika on mora prvo aplikaciji dati odobrenje. Odmah nakon registracije korisnik će biti preusmjeren na Googleovu stranicu za davanje odobrenja prikazanu na Slici 4.2. Korisnik će biti upoznat s pravima koja ova aplikacija ima nad njegovim resursima, a radi se jedino o pravu upravljanja osobnim kalendarom. Ovo pravo potrebno je kako bi se događaji mogli unositi u kalendar. Korisnik mora potvrditi ovaj obrazac

ako želi koristiti aplikaciju, a ukoliko odbije ovo odobrenje, neće biti moguće unositi događaje u njegov kalendar.

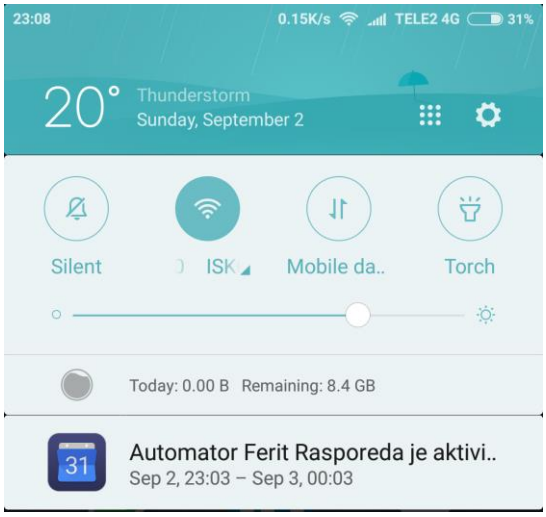


Slika 4.2 - Googleov obrazac za davanje odobrenja.

Nakon što korisnik potvrdi obrazac odobrenja bit će preusmjeren na stranicu zahvale te će mu u kalendar biti unesen event dobrodošlice. Na Slici 4.3 vidimo stranicu zahvale. Stranica zahvale jednostavna je statična stranica koja služi kako bi korisniku dala povratnu informaciju o uspješnoj registraciji i aktivaciji aplikacije.

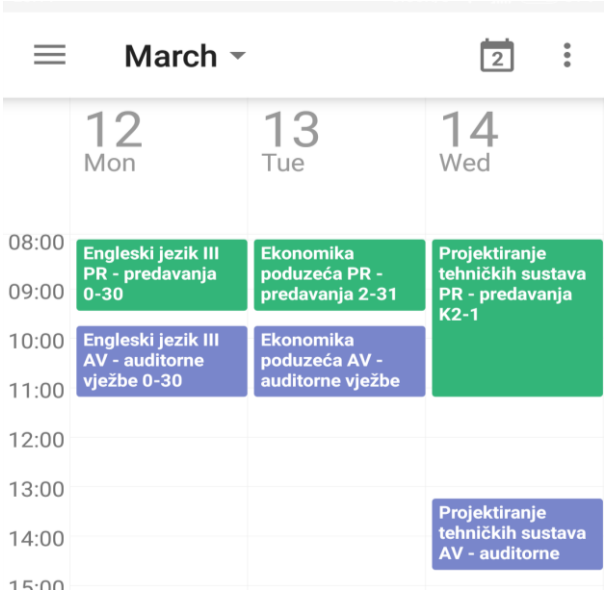
Slika 4.3 - Stranica zahvale.

Odmah nakon registracije u kalendar će biti dodan događaj dobrodošlice te će on biti vidljiv i u obliku *push* notifikacije ako su omogućene na samom uređaju, na Slici 4.4 vidljiv je događaj dobrodošlice u *push* notifikaciji.



Slika 4.4 - Push notifikacija s događajem dobrodošlice.

Aplikacija će svakoga tjedna samostalno unositi podatke o rasporedu u Google Calendar. Na Slici 4.5 vidljiv je primjer rasporeda nastave unesenog u kalendar.



Slika 4.5 - Primjer rasporeda.

5. ZAKLJUČAK

Aplikacija za automatizirano dodavanje rasporeda u kalendar integrira digitalni raspored Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek s Google Calendar internetskom uslugom. Aplikacija je namijenjena studentima i djelatnicima fakulteta kojima su potrebne informacije o digitalnom rasporedu fakulteta te im olakšava planiranje te usklađivanje fakultetskih i ostalih obaveza. Veliki naglasak prilikom razvoja aplikacije stavljen je na razvoj popratne infrastrukture te automatizaciju razvojnog procesa. Korištena je Google Cloud platforma za računarstvo u oblaku na kojoj su pokrenuta tri izvršna okruženja. Sva infrastruktura zapisana je kao kod koristeći Docker tehnologiju za virtualizaciju na razini operacijskog sustava. Proces razvoja automatiziran je pomoću Gitlab CI/CD alata kako bi se osigurala kontinuirana integracija i isporuka te je praćen razvojni okvir rada Git Flow. Sama aplikacija napisana je u jeziku JavaScript te se izvršava u Node.js izvršnom okruženju za pozadinske servise. Implementiran je algoritam eksponencijalne odgode kako bi broj zahtjeva u sekundi bio unutar poslužiteljskih zahtjeva te su osnovne informacije o korisnicima spremljene u MongoDB NoSQL bazu podataka.

LITERATURA

[1] UHP Digital, Git Flow, H. Tonkovac, E. Dragun, http://www.uhp-digital.com/hr/blog-article/12/git-flow, kolovoz 2018

[2] Github, Celebrating nine years of GitHub, https://blog.github.com/2017-04-10-celebrating-nine-years-of-github-with-an-anniversary-sale/, kolovoz 2018

[3] Gitlab, GitLab Continuous Integration & Deployment, https://about.gitlab.com/features/gitlab-ci-cd/, kolovoz 2018

[4] A successful Git branching model, V. Driessen, https://nvie.com/posts/a-successful-git-branching-model/, kolovoz 2018

[5] What is Infrastructure as Code?, S. Guckenheimer, https://docs.microsoft.com/en-us/azure/devops/learn/what-is-infrastructure-as-code, kolovoz 2018

[6] What is Docker?, Red Hat, https://www.redhat.com/en/topics/containers/what-is-docker, lipanj 2020

[7] Docker Compose, https://docs.docker.com/compose/, kolovoz 2018

[8] Compute Engine, https://cloud.google.com/compute/, kolovoz 2018

[9] B. A. Syed, Beginning Node.js, Chapter 2: Understanding Node.js, Apres, New York, 2014

[10] B. A. Syed, Beginning Node.js, Chapter 4: Node.js packages, Apres, New York, 2014

[11] What is MongoDB, https://www.mongodb.com/what-is-mongodb, kolovoz 2018

[12] Internet Engineering Task Force, Request for Comments 6749, The OAuth 2.0 Authorization Framework, D. Hardt, https://tools.ietf.org/html/rfc6749, lipanj 2020

[13] C. Negus, Linux Bible Ninth Edition, Chapter 3: Using The Shell, Wiley, Indianapolis, travanj 2015

[14] Docker Cookbook, S. Goasguen, prosinac 2015

[15] DockerHub, Node, https://hub.docker.com/_/node/, kolovoz 2018

[16] Docker Compose, https://docs.docker.com/compose/, kolovoz 2018

[17] Google Cloud Documentation, Truncated Exponential Backoff, https://cloud.google.com/storage/docs/exponential-backoff, rujan 2019

SAŽETAK

U ovom završnom radu opisan je razvoj automatiziranog internetskog poslužitelja za dodavanje rasporeda u kalendar s naglaskom na infrastrukturu i razvojni proces. U teorijskom dijelu rada opisane su korištene tehnologije: Git sustav kontrole verzija te njegov okvir rada Git Flow, Gitlab CI/CD alat za kontinuiranu integraciju i dostavu, Google Cloud platforma za računarstvo u oblaku, Docker tehnologija virtualizacije na razini operacijskog sustava, Node.js izvedbeno okruženje za programski jezik JavaScript, MongoDB NoSQL baza podataka te OAuth2 protokol za autentifikaciju s Google Calendar aplikacijskim programskim sučeljem. U praktičnom dijelu rada korištene su prethodno opisane tehnologije kako bi se razvila infrastruktura potrebna za pokretanje aplikacije u tri izvršna okruženja te je automatiziran proces njihovih ažuriranja. Implementirane su funkcije za komunikaciju s Googleovim aplikacijskim programskim sučeljima putem OAuth2 protokola te je implementiran algoritam eksponencijalne odgode za ograničenje broja poslanih zahtjeva. Sustav razvijen u ovom završnom radu olakšava upravljanje vlastitim kalendarom, automatizirajući proces kopiranja stavki iz FERIT kalendara u Google Calendar.

Ključne riječi:

Algoritam eksponencijalne odgode, Docker, Google Calendar API, Gitlab CI/CD, OAuth2

ABSTRACT

Title: Service for Automated Addition of Course Schedule to a Calendar

This bachelor’s thesis describes the development of an Internet server for automated calendar scheduling with emphasis on infrastructure and development process. Theoretical part of the thesis describes the technologies used: Git version control system and it's framework Git Flow, Gitlab CI/CD tool for continuous integration and delivery, Google Cloud cloud computing platform, Docker operating system level virtualization technology, Node.js runtime environment for JavaScript programming language, MongoDB NoSQL Database, OAuth2 Authentication Protocol for authentication with Google Calendar Application Programming Interface. In the practical part of the thesis, previously described technologies have been used to develop the infrastructure needed to run the application in three distinct runtime environments and automate the process of their updates. Functions for communicating with Google application programming interfaces through the OAuth2 protocol have been implemented and an exponential delay algorithm has been implemented to limit the number of requests sent.

Keywords:
Exponential backoff algorithm, Docker, Google Calendar API, Gitlab CI/CD, OAuth2

ŽIVOTOPIS

Hrvoje Tonkovac rođen je 17.03.1997. u Osijeku. Pohađao je Osnovnu školu Retfala u Osijeku. 2011. godine upisuje III. Prirodoslovno-matematičku gimnaziju u Osijeku. 2015. godine upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija na Sveučilištu Josipa Jurja Strossmayera u Osijeku. Na drugoj godini studija zapošljava se kao PHP programer pozadinskih servisa u osječkoj tvrtki *UHP Digital*. Erasmus praksu odradio je u Kölnu u Njemačkoj, u firmi Detecon International GmbH, članici Deutsche Telekom grupe. Certificirani je administrator Red Hat Enterprise Linux operacijskih sustava.

Hrvoje Tonkovac

_____