УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Ивона Милутиновић

Развој апликације за асистенцију вежбачима ослањањем на корисничке податке употребом Android и Django оквира

мастер рад

Београд, 2024.

Ментор:

др Иван Чукић, доцент
Универзитет у Београду, Математички факултет

Чланови комисије:

проф. др Саша Малков, ванредни професор
Универзитет у Београду, Математички факултет

др Богдан Павковић, доцент
Универзитет у Новом Саду, Факултет техничких наука

Датум одбране: дан. месец 2024.

Наслов мастер рада: Развој апликације за асистенцију вежбачима ослањањем на корисничке податке употребом Android и Django оквира

Резиме: TODO

Кључне речи: (In progress) машинско учење, рекурентне неуронске мреже, Android, Django REST Framework, scrapy, план тренинга, трчање

Садржај

Глава 1

Увод

ТОДО: Описати шта је одрађено у којој глави

1.1 TODO

Глава 2

Оперативни систем Android

2.1 Историјат

Глава 3

Django и Django REST Framework

У првим корацима веб развоја, програмери су писали веб странице ручно користећи HTML. Уколико би веб сајт требало ажурирати, то је подразумевало мењање свих HTML страница које та измена обухвата. У случају великих сајтова, ажурирање је представљало напоран посао који је одузимао доста времена.

Први напредак у односу на овај приступ је направила група инжињера из *Националног центра за суперрачунарске апликације (енг. the National Center for Supercomputing Applications, NCSA)* која је креирала *Општи приступни интерфејс (енг. Common Gateway Interface, CGI)*, протокол који омогућава веб серверима да динамички генеришу HTML странице уз помоћ екстерних програма.

Међутим, и CGI је имао своје недостатке - скрипте које је одржавао су морале да имају доста поновљеног шаблонског кода, што је отежавало његово вишеструко искоришћење као и разумевање програмера на почетку рада. У сфери веб програмирања појављује се програмски језик *PHP* који решава многе наведене проблеме и постаје један од најпопуларнијих алата за креирање динамичких веб сајтова. Његове главне погодности јесу једноставност коришћења, будући да се лако уграђује у HTML, и блага крива учења за познаваоце HTML-а. Нажалост, ни ово решење није прошло без недостатака, безбедносна заштита коју PHP нуди није била на високом нивоу, а лакоћа писања PHP кода је лако доводила до несистематичног и поновљивог кода.

Ово је довело до развијања веб оквира нове генерације способних да се

изборе са већином изазова и амбиција веб програмирања. Овој генерацији припадају оквири *Ruby on Rails* и *Django*. Django омогућава развијање комплексних и динамичких веб сајтова за кратко време захваљујући апстракцији високог нивоа уобичајених шаблона веб развоја и јасним упутствима и методама за решавање учесталих програмерских задатака. На тај начин, програмери се могу фокусирати на кључну логику, док ће се Django побринути за све рутинске задатке. [1]

3.1 Основне карактеристике

Django је веб оквир - да ли је ово потребно јер сам већ поменула горе? заснован на Python програмском језику који има приступ *„укључене батерије" (енг. "batteries included")*. Овај приступ представља филозофију дизајна програмских алата која подразумева да се њиховом инсталацијом добија богати скуп уграђених функционалности које омогућавају корисницима да одмах почну са радом. Django настоји да очува овај принцип пружајући корисницима стандардну библиотеку са разним додацима (енг. *add-ons*) за најчешће задатке веб развоја.

-based web framework that han- dles the challenging parts of building a website: authentication, connecting to a database, logic, security, and so on. There are also thousands of third-party packages that add functionality to Django itself, the most prominent of which is Django REST Framework, which allows developers to transform any existing Django project into a powerful web API. Django and Django REST Framework are used by the largest tech companies in the world, in- cluding Instagram, Mozilla, and Pinterest. But they are also well-suited to beginners or weekend side projects because Django's "batteries-included" approach masks much of the underlying complexity, allowing for rapid and secure development.

One of the many strengths of Python is its "batteries included" philosophy: when you install Python, it comes with a large standard library of packages that you can start using immedi- ately, without having to download anything else. Django aims to follow this philosophy, and it includes its own standard library of add-ons useful for common Web-development tasks. This chapter covers that collection of add-ons.

- Filozofija: „Batteries included" pristup, DRY (Don't Repeat Yourself) princip.

- MVT - Bezbednost i Performanse - administrativni panel - URL routing - Models: Kreiranje i korišćenje modela za rad sa bazom podataka. - Views i Templates: Kreiranje pogleda i šablona za prikaz podataka korisnicima. - Autentifikacija i autorizacija: Built-in mehanizmi za rad sa korisnicima. - crud == ovde ili u drf

3.1.1 MBT

+ дијаграм

3.2 Postavljanje i konfiguracija Django projekta

3.3 Django REST Framework (DRF)

uvod sa: Potreba za RESTful servisima

3.3.1 Основни концепти DRF-a

– Serializers Views Routers Permissions Authentication

3.3.2 Kreiranje API-a: Koriscenje DRF-a za kreiranje i upravljanje API endpointima

3.3.3 Set-up projekta: Koraci za integraciju DRF-a u postojeći Django projekat.

section Закључак – kako ovo ubaciti? Сумирање предности - Sumiranje prednosti: Zasto koristiti Django i DRF u modernom web razvoju. Изазови и решења – mozda samo ovo

———————————————————— Ideje sta pisati:

https://chatgpt.com/c/8bfb0d05-563e-44b2-8f5b-c428264c2066

• Arhitektura Djang-a

MVT

Batteries included,, pristup

Potreba za RESTful servisima

Razvoj Web Aplikacija sa Djangom

Postavljanje i konfiguracija Django projekta

Upravljanje statičkim i medijskim fajlovima

Izgradnja REST API-ja sa Django REST Framework-om

• Osnove REST arhitekture

Dizajn baze podataka

• Kreiranje modela i migracije

Razvoj API krajnjih tačaka (endpoints)

• Bezbednosni aspekti Django i DRF aplikacija

Prednosti i mane korišćenja Django i DRF ————————————————————

————————

1. Uvod u Django Istorijat i razvoj: Kratka istorija Django-a, osnivači, i razlozi za njegovo kreiranje. Filozofija: "Batteries included„ pristup, DRY (Don't Repeat Yourself) princip. Osnovne karakteristike: Model-View-Template (MVT) arhitektura, administrativni panel, sigurnost, brz razvoj.

2. Osnovni koncepti Django-a MVT arhitektura: Detaljno objašnjenje Modela, Pogleda i Šablona. URL routing: Kako se definišu i koriste URL šeme. Models: Kreiranje i korišćenje modela za rad sa bazom podataka. Views i Templates: Kreiranje pogleda i šablona za prikaz podataka korisnicima. Forms: Korišćenje obrazaca za unos i validaciju podataka. Autentifikacija i autorizacija: Built-in mehanizmi za rad sa korisnicima.

3. Django REST Framework (DRF) Uvod u DRF: Kratka istorija, razlozi za korišćenje, prednosti u odnosu na Django standardne view-ove. Osnovni koncepti DRF-a: Serializers: Transformacija kompleksnih podataka (queryset) u primitivne tipove podat aka. – ovo mozda ne

6. Zaključak Sumiranje prednosti: Zašto koristiti Django i DRF u modernom web razvoju. Izazovi i rešenja: Potencijalni problemi i kako ih prevazići.

Django is a very popular Python-based web framework that han- dles the challenging parts of building a website: authentication, connecting to a database, logic, security, and so on. There are also thousands of third-party packages that add functionality to Django itself, the most prominent of which is Django REST Framework, which allows developers to transform any existing Django project into a powerful web API. Django and Django REST Framework are used by the largest tech companies in the world, in- cluding Instagram, Mozilla, and Pinterest. But they are also well-suited to beginners or weekend side projects because Django's "batteries-included" approach masks much of the underlying complexity, allowing for rapid and secure development.

An API (Application Programming Interface) is a shorthand way to describe how two computers communicate directly with one another. For web APIs, which exist on the world wide web, the dominant architectural pattern is know as REST (REpresentational State Transfer) and will be covered properly later on in this book. Back in 2005, when Django was first released, most websites consisted of one large monolithic codebase. The back-end of database models, views, and URLs were combined with front-end templates to control the presentational layer of each web page. But these days it is far more common for websites to adopt an API-first approach of formally separating the back-end from the front-end. This allows a website to use a dedicated JavaScript front-end framework, such as React or Vue, which were released in 2013 and 2014 respectively.Introduction

When the current front-end frameworks are eventually replaced by even newer ones in the years to come, the back-end API can remain the same. No major rewrite is required. Another major benefit is that one single API can support multiple front-ends written in different languages and frameworks. Consider that JavaScript is used for web front-ends, while Android apps require the Java programming language, and iOS apps need the Swift programming language. With a traditional monolithic approach, a Django website cannot support these various front-ends. But with an internal API, all three can communicate with the same underlying database back-end! Growing websites can also benefit from creating an external API that allows third-party devel- opers to build their own iOS or Android apps. When I worked at Quizlet back in 2010 we did not have the resources to develop our own iOS or Android apps, but we did have an external API available that more than 30 developers used to create their own flashcard apps powered by the Quizlet database. Several of these apps were downloaded over a million times, enriching the developers and increasing the reach of Quizlet at the same time. The major downside to an API-first approach is that it requires more configuration than a traditional Django application. However as we will see in this book, the fantastic Django REST Framework library removes much of that complexity for us.

Глава 4

Неуронске мреже

Глава 5

Имплементација апликације *Train Wiser*

(In progress) Употреба технологија описаних у претходним поглављима представљена је у практичном делу рада - креираној апликацији *Train Wiser* [2]. Апликација пружа статистку и асистенцију за тренирање и напредак вежбача на основу корисничких података увезених са апликације Strava путем њеног јавног интерфејса, као и на основу резултата трка одржаних у Републици Србији. Апликација *Train Wiser* је намењена тркачима који желе да детаљније прате своје перформансе, као и да унапреде своје резултате.

У наставку су описане додатне технологије коришћене у имплементацији апликације, њен дизајн и архитектура.

5.1 Кључни аспекти при развоју апликације

TODO: (! измени наслов) (In progress) За израду клијентске стране апликације, коришћен је програмски језик Java уз Android развојни оквир, док је за креирање серверског дела апликације употребљен програмски језик Python уз оквире Scrapy, Django и Django REST Framework. У наставку су кроз кључне тачке при развоју апликације укратко описане додатне употребљене технологије.

Да ли да наслов остаје "Преглед коришћених технологија" или да променим у нешто попут ""?

5.1.1 Аутентификација и ауторизација

Ауторизација и аутентификација представљају кључне концепте безбедности система.

Аутентификација је процес доказивања индентитета одређених ентитета, корисника или других система (у даљем тексту - корисника), и утврђивања да ли је тим корисницима дозвољен приступ систему. Доказивање идентитета најчешће подразумева прилагање корисничког имена и лозинке, а додатно се могу тражити и биометријски подаци и/или аутентификацијски токени. Ауторизација, с друге стране, долази након аутентификације, и подразумева додељивање права и привилегија аутентификованом кориснику. На овај начин се одређује којим ресурсима система корисник има дозволу да приступи, као и скуп акција које може извршити.

У савременим веб апликацијама, често се јавља потреба да једна апликација приступа ресурсима друге апликације у име корисника. Овај концепт је важан у контексту интеграције различитих услуга и обезбеђивања доброг корисничког искуства. На пример, апликација може имати опцију за предлагање контакта на основу пратилаца са одређене друштвене мреже корисника, и да за то тражи од корисника да се пријави на ту друштевену мрежу.

Уколико би апликација добила корисничке креденцијале како би се аутентификовала на друштвеној мрежи, она поред приступа листи пратиоца добија потпун приступ корисничком налогу. Такође, апликација би имала могућност да сачува корисничке креденцијале за друштвену мрежу, као и да то уради на небезбедан начин. У случају да се апликација компромитује, креденцијали за друштвену мрежу би такође били компромитовани. Наведене проблеме оваквог приступа настоји да реши протокол OAuth 2.0.

OAuth 2.0 омогућава апликацијама да добију ограничени приступ корисничким ресурсима на спољним сервисима без прослеђивања креденцијала за пријављивање апликацији, истовремено задржавајући висок ниво безбедности. Апликација ће приступати ресурсима у складу са датим дозволама од стране корисника, чиме се повећава контрола над приватним подацима.
Два најчешћа случаја употребе су:

1. Аутентификовање корисника на основу налога са спољног сервиса

- Овај принцип се зове *федеративни идентитет (енг. federated identity).*

- На пример, Instagram дозвољава корисницима да се пријаве преко Facebook налога.

2. Пружање дозвола апликацији да приступа ресурсима спољног сервиса у одсуству корисника

- Овај принцип се зове *делегирање ауторизације (енг. delegated authority).*

- На пример, Instagram приступа сликама са Facebook налога без посредства корисника.

Принцип рада протокола се заснива на следећим корацима:

1. Регистрација апликације

Регистрација апликације представља процес који је потребно извршити једанпут како би се успоставила интеграција, односно веза од поверења, између апликације и спољног сервиса. Исход овог процеса је да су апликацији доступни наредни параметри:

- *Клијентски идентификатор (енг. client ID)* – јединствени идентификатор на нивоу спољног сервиса који представља апликацију која жели да се региструје; добија се од спољног сервиса или се дефинише од стране власника апликације

- *Клијентски тајни кључ (енг. client secret)* – тајни кључ који се увек добија од спољног сервиса. Овај кључ, заједно са клијентским идентификатором, представља креденцијале апликације који се прилажу спољном сервису како би се обавила аутентификација апликације пре процеса делегирања ауторизације

- *Веб адреса за преусмерење одговора од стране спољног сервиса (енг. redirection endpoint)* – веб адреса на коју спољни сервис шаље одговоре апликацији који обично представљају токене или грешке. Ову веб адресу у највећем броју случаја обезбеђује власник апликације.

- *Веб адреса ауторизације (енг. authorization endpoint)* – веб адреса коју ће клијентска апликација користити да иницира процес ауторизације, одређена је од стране спољног сервиса

- *Веб адреса за токене (енг. token endpoint)* – веб адреса обезбеђена од стране спољног сервиса коју ће апликација користити да иницира процес размене токена

2. Добијање приступног токена

Начин за осигуравање приступа заштићеним дигиталним ресурсима јесте обезбеђивање истих *приступним токеном (енг. access token)* и онемогућавање других начина приступа. Приступни токен има свој опсег и време трајања. Опсег представља скуп заштићених ресурса којима се може приступити, док време трајања представља време након чијег истицања ће присуп са датим токеном постати невалидан.

Након успешне регистрације, спољни сервис ће узвратити приступни токен апликацији. Уколико спољни сервис подржава обнављављање приступног токена коришћењем *токена за обнову (енг. refresh token)*, овај токен ће бити додатно достављен уз приступни токен.

3. Преузимање захтеваних ресурса посредством приступног токена

Након што се корисник аутентификује на спољни сервис на који га је преусмерила апликација и одобри захтев за ауторизацију, апликација ће добити приступ оним ресурсима којима је корисник одобрио приступ. Захтев за добијање ресурса треба да садржи приступни токен. Приступ захтеваним ресурсима се може радити све док приступни токен не истекне или буде поништен.

4. Коришћење токена за обнову приступног токена

Након што приступни токен истекне, апликација има опцију да испочетка понови процес аутентификације, што може захтевати да се корисник поново пријави или да користи токен за обнову приступног токена за шта није потребно учешће корисника. У другом случају се нови приступни токен добија кроз захтев за обнову приступног токена уз слање токена за обнову.

У претходном процесу разликујемо апликације од поверења и неповерљиве апликације у зависности да ли апликација има могућност да сигурно чува и преноси информације. Апликације засноване на веб прегледачу код којих се све информације чувају у самом прегледачу (нпр. HTML и JavaScript апликације) као и Flash апликације које не захтевају дуготрајан приступ корисничким подацима јесу неке од примера неповерљивих апликација. Пример

апликације од поверења би била апликација која има клијентски и серверски део уз базу података, где је серверски део задужен за безбедно руковање и чување података.

Клијентски тајни кључ, приступни токен и токен за обнову се добијају једино у случају апликације од поверења. У контексту неповрељиве апликације, процес добијања захтеваних ресурса је поједностављен. У овом случају, сваки пут када апликација има потребу да користи ресурсе спољног сервиса, преусмериће корисника на одобрење ауторизације апликације са спољним сервисом. Након што корисник одобри ауторизацију, апликација добија кључ од спољног сервиса који ће користити у захтеву за приступ жељеним ресурсима. Спољни сервис ће у случају успешне валидације кључа проследити апликацији тражене ресурсе. [3]

Апликација Strava користи OAuth 2.0 [4] за аутенификацију са V3 API-ем, јавним интерфејсом ове апликације који даје могућност програмерима да приступе подацима ове апликације. *Train Wiser* апликација је преко овог протокола увезана са Strava апликацијом како би имала приступ свим активностима корисника који одобре ауторизацију.

5.1.2 Формирање скупа података са активностима корисника

У тренутку када нови корисник одобри ауторизацију са апликацијом Strava преко апликације *Train Wiser*, апликација *Train Wiser* ће екстраховати све његове релевантне претходне активности и сместити их у базу *StravaActivity*. Информације о новим активностима корисника се уписују у базу како их корисних отпреми на апликацији Strava.

Будући да Strava API допушта максимално 200 упита на сваких 15 минута са највише 2000 упита у дану, допуњавање базе претходним активностима корисника је одрађено коришћењем библиотеке Django за планирање задатака, *django-crontab*. Овај механизам аутоматизације заједно са појмом *мрежних кука (енг. webhooks)* које су коришћене за аутоматско прикупљање нових активности корисника су детаљније описани у наставку.

5.1.2.1 Django библиотека django-crontab

У Django апликацијама, планирање периодичних задатака може бити корисно за различите сврхе, као што су ажурирање база података, слање извештаја, чишћење привремених фајлова и друге периодичне активности. Један од популарних начина за управљање Cron задацима у Django-у је коришћење библиотеке *django-crontab* [5] која у позадини користи системски процес *Cron*.

Cron процес, подразумевано доступан на Unix и Linux системима, омогућава аутоматизацију извршавања `shell` команди у одређено време, елиминишући потребу за ручним покретањем. Cron се ослања на конфигурациону датотеку *crontab* (скраћено од "cron table") у којој је у сваком реду уписано време извршавања и, у виду `shell` команде, посао који је потребно извршити. Ова датотека се од стране Cron процеса проверава у сваком минуту, и на основу њених уноса се извршавају доспели послови. [6]. Један ред ове датотеке представља један *Cron-посао (енг. cronjob)*. [7] Синтакса за навођење Cron послова у `crontab` датотеци је приказана на слици 5.1. [6]

На пример, следећи унос ће покренути скрипту `/home/user/script.sh` сваког дана у 2:30 ујутру:

```
30 2 * * * /home/user/script.sh
```

У Django апликацијама [5], Cron задаци се додају у `settings.py` модул у оквиру `CRONJOBS` променљиве:

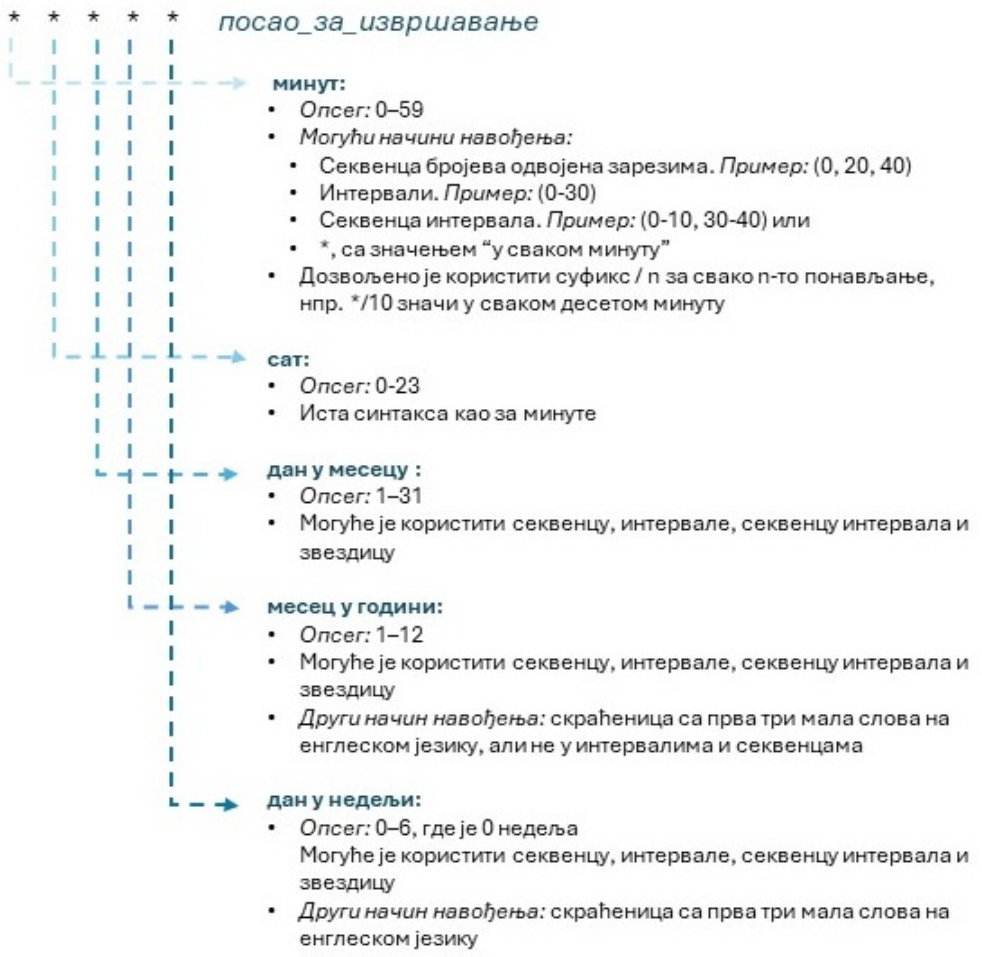
```
CRONJOBS = [
    ('0 0 * * *', 'scheduled_job'),
    ('0 4 * * *', 'django.core.management.call_command',
    ['django_menagement_command']),
]
```

где је `scheduled_job` функција дефинисана у пројекту, а `django_menagement_command` уграђена Django menagement команда (попут `migrate`, `collectstatic`, ...) или прилагођена команда креирана од стране корисника.

Да би се дефинисани Cron задаци извршавали, потребно их је додати са командом:

```
python manage.py crontab add
```

Додате задатке је могуће уклонити наредном командом:



Слика 5.1: Синтакса конфигурационе датотеке `crontab`

```
python manage.py crontab remove
```

У апликацији *Train Wiser* се користе два Cron посла, један који на сваких сат времена уписује претходне активности корисника у базу и други за додавање детаља о новим активностима корисника.

5.1.2.2 Мрежне куке

Мрежне куке (енг. webhooks) пружају начин за аутоматизацију комуникације између различитих сервиса путем HTTP захтева. Када се одређени догађај деси у изворном систему, HTTP захтев ће се аутоматски послати одредишном систему, најчешће садржећи корисне податке о догађају које одредишни систем захтева. Периодично слање захтева изворном систему ради провере да

ли постоје нови или ажурирани догађаји (енг. *pooling*) може бити неефикасно и оптеретити изворни систем. Коришћењем мрежних кука апликација ће добити информације о догађајима у реалном времену и искључиво онда када се догађаји десе. На тај начин се омогућава уштеда ресурса и ефикасна комуникација.

Мрежне куке се јако често користе на *SaaS (енг. software as a service)* платформама, будући да оне на основу активности које се дешавају подржавају креирање различитих типова догађаја. Да би апликација могла да добија HTTP захтеве мрежних кука, потребно је да се региструје за догађаје за које их изворна платформа нуди, као и да обезбеди веб адресу на коју ће се захтеви слати. [8, 9]

Коришћење мрежних кука у апликацији *Train Wiser* Апликација Strava подржава мрежне куке за одређене промене у личним подацима и активностима корисника, и охрабрује све API апликације да их користе.

Када се догађај за који је мрежна кука везана догоди, POST захтев се шаље на веб адресу дефинисану од стране апликације. Очекивано је да апликација узврати одговор са статусом 200 у оквиру 2 секунде. Уколико се то не деси, POST захтев се понавља још максимално два пута, стога се саветује да се добијене информације обрађују асинхроно уколико је за обраду потебно више времена.

Да би се апликација регистровала за мрежне куке апликације Strava, потребно је да се креира POST захтев ка адреси `https://www.strava.com/api/v3/push_subscriptions`. Очекивано је да захтев садржи параметре у URL формату који укључују клијетски идентификатор и клијентски тајни кључ апликације, тј. креденцијале апликације за приступ Strava ресурсима, поменуте у подпоглављу 5.1.1, затим, веб адресу повратног позива дефинисану од стране апликације на којој ће се захтеви слати *(URL callback)* и јединствени токен дефинисан од стране власника апликације који ће апликација Strava узвратити слањем верификационог GET захтева ка адреси апликације. Верификациони GET захтев поред токена садржи и два поља чије вредности су ниске карактера, *hub.mode* који је увек фиксне вредности *"subscribe"*, и *hub.challenge*, чија је вредност насумична ниска коју апликација узвраћа као одговор Strava апликацији затварајући процес креирања регистрације. [9]

Да ли додати како је одређена регистрација у *Train Wiser* апликацији?

Од догађаја за које се мрежне куке нуде од стране апликације Strava, за формирање скупа података са активностима вежбача за *Train Wiser* апликацију је релевантно креирање нових активности.

Након што корисник постави нову активност и апликација *Train Wiser* добије POST захтев мрежне куке, у базу у којој се чувају активности (*StravaActivity*) ће се уписати идентификатор активности и идентификатор корисника који је додао активност. Дохватање детаљних информација о активности се ради коришћењем Django Cron посла који ће на сваких сат времена у тридесетом минуту проверити да ли у бази постоје активности које нису попуњене, тј. садрже само идентификатор активности и страни кључ ка StravaAthlete бази.

5.1.3 Формирање скупа података са резултатима трка

Скуп података са резултатима трка је формиран на основу резултата са сајтова runtrace.net, trka.rs и bgdmarathon.org за које је добијена потврда да се јавно доступни подаци могу користити у истраживачке сврхе. Подаци са ових сајтова су обрађени *техником кроловања (енг. crawling)* и *екстракцијом података (енг. scraping).*

Преузимање података од стране програма на било који други начин осим директном комуникацијом програма са API-ем странице представља технику екстракције података. Коришћење екстракције података уместо API-ја се саветује када API није доступан или онда када јесте доступан, али формат података који пружа није одговарајући или постоји ограничење у обиму и учесталости захтева који се могу послати. Екстрактори веб података су ефикасан алат за прикупљање и обраду великих количина података са различитих веб страница. [10] Кроловање података представља аутоматско извршавање програма за систематску претрагу веб страница и индексирање њиховог садржаја. Индексирање омогућава претраживачима да брзо и ефикасно пронађу релевантне информације у зависности од употребе. [11]

Кроловање и екстракција података са веб страница у *Train Wiser* апликацији су одрађени коришћењем оквира Scrapy. Scrapy је оквир отвореног кода програмског језика Python за веб кроловање и екстракцију структуираних података са HTML/XML страница. Често се користи за анализу и процесирање података, надгледање активности, аутоматизацију тестирања и историјско архивирање. Главна одлика овог алата је да захтеве распоређује и процесира

асинхроно што му омогућава веома брзо кроловање у односу на секвенциони приступ будући да се више захтева може процесирати у исто време уз постојање толеранције грешака.

Овај оквир пружа једноставан API са доста могућности који је лако прилагодљив потребама програмера. Екстракција података се ради коришћењем проширених CSS селектора и XPath израза, док су JSON, CSV и XML формат на располагању на излазу. [12] За потребе *Train Wiser* апликације, екстрактовани подаци су експортовани у JSON формат.

5.2 Архитектура и дизајн апликације

(In progress) За архитектуралну организацију апликације је одабран клијент-сервер модел. Клијентска апликација (енг. *frontend*) је задужена за комуникацију са корисником и прослеђивање захтева серверској апликацији (енг. *backend*) која је задужена за унутрашњу логику.

5.3 Могућа унапређења

Глава 6

Коришћење апликације *Train Wiser*

Глава 7

Имплементација модела машинског учења

7.1 Прикупљање података

(In progress) Поступак формирања скупа података са резултатима трка описан је у подпоглављу 5.1.3. Број укупно сакупљених података са сајтова дат је у табелици 7.1:

Сајт	**Укупан број резултата релевантних трка**	**? - ТОДО**
runtrace.net	19708	
trka.rs	37491	
bgdmarathon.org	50275	

Табела 7.1: Број података са сваког од сајта са резултатима трка

7.2 Евалуација модела

7.3 Могућа унапређења

Глава 8

Закључак

ТОДО: Описати укратко шта је одрађено у раду и шта је закључак

Библиографија

[1] Adrian Holovaty and Jacob Kaplan-Moss. *The Definitive Guide to Django: Web Development Done Right*. Apress, 2009.

[2] Ивона Милутиновић. Апликација Train Wiser - имплементација. `https://github.com/ivonamilutinovic/MasterThesis/train_wiser`.

[3] Charles Bihis. *Mastering OAuth 2.0*. Packt Publishing, 2015.

[4] Getting Started with the Strava API - How to Authenticate. `https://developers.strava.com/docs/getting-started/#oauth`. Приступљено: 17. јун 2024.

[5] django-crontab. `https://pypi.org/project/django-crontab/`. Приступљено: 12. јун 2024.

[6] Daniel J. Barrett. *Linux Pocket Guide*. O'Reilly Media, Inc., 2012.

[7] CronJob. `https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/`. Приступљено: 12. јун 2024.

[8] What Are Webhooks And How Do They Work. `https://hookdeck.com/webhooks/guides/what-are-webhooks-how-they-work`. Приступљено: 3. јун 2024.

[9] Webhook Events API. `https://developers.strava.com/docs/webhooks/`. Приступљено: 4. јун 2024.

[10] Ryan Mitchell. *Web Scraping with Python*. O'Reilly Media, Inc., 2015.

[11] What Is a Web Crawler? `https://www.akamai.com/glossary/what-is-a-web-crawler`. Приступљено: 9. јун 2024.

[12] Scrapy 2.11 documentation. `https://doc.scrapy.org/en/latest/`. Приступљено: 8. јун 2024.

Биографија аутора