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, ванредни професор
Универзитет у Новом Саду, Факултет техничких наука

Датум одбране: септембар 2024.

Захваљујем се својој породици и пријатељима на подршци и мотивацији, и свом ментору, Ивану Чукићу, који је допринео мом професионалном развоју кроз корисна и занимљива предавања у току студија и предано пратио израду овог рада уз конструктивне савете.

Овај рад посвећујем својој породици и баки.

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

Резиме: Рад са корисничким подацима представља важан приступ развоја савремених апликација јер омогућава алгоритмима да дају боље резултате увидом у понашање корисника. У домену спортских апликација оваква решења играју битну улогу у напретку вежбача. Како је најпогоднији начин коришћења оваквих апликација мобилна апликација, рад се бави израдом апликације *Train Wiser* која на клијентској страни користи оперативни систем *Android*, док је на серверској страни направљена *Python* апликација за имплементацију унутрашње логике и обраду корисничких података која користи оквире *Django, Django REST Framework* и *Scrapy*. Подаци се добијају од стране спортске апликације *Strava* са којом је кориснички налог увезан и екстракцијом података са сајтова за резултате трка. Развијена апликација нуди корисницима могућност израде плана тренинга, увид у месечну статисику тренинга и предикцију резултата на тркама. Циљ рада је да да темељан увод у изградњу *Android* и *Django* апликација и да детаљно опише начин рада и имплементацију изграђене апликације. Рад даје смернице за даљи развој и унапређења апликације, као и употребу описаних концепата на другим решењима.

Кључне речи: оперативни систем *Android*, *Retrofit*, *Django*, *Django REST Framework*, *Scrapy*, *OAuth 2.0*, план и статистика тренинга, трчање

Садржај

Глава 1

Увод

Укључивање корисничких података у алгоритме апликација постаје све популарнији приступ у развоју апликација који доприноси побољшању корисничког исткуства. Кориснички подаци се често користе за унапређење алгоритама препорука, анализу корисничког понашања, као и за персонализацију садржаја. Растом броја корисника, апликација повећава количину и разноликост доступних података што омогућава повећање тачности и ефикасности алгоритама који их обрађују. Потреба за коришћењем корисничких података у спортским апликацијама постаје све израженија како расте значај персонализованог приступа тренинзима. Могућност да корисници долазе до нових важних информација како кроз алгоритме које раде искључиво над њиховим подацима, тако и над активностима других корисника, може допринети побољшању спортског напретка корисника.

Најпогоднији тип оваквих апликација су апликације за мобилне уређаје. Оперативни систем *Android* има подршку за велики број уређаја што представља предуслов да апликација досегне довољан број корисника како би њени алгоритми радили ефикасно и уз што мање грешака. Отвореност кода олакшава развој програмерима пружајући приступ алатима и библиотекама за имплементацију апликација, док графички кориснички интерфејс који *Android* пружа олакшава и чини пријатнијим корисничко искуство.

Одвајање унутрашње логике и обраде података од презентационог слоја је устаљена добра пракса архитектуралне организације јер се тиме постиже јасна подела одговорности и олакшава одржавање кода. Стога се најчешће користи модел клијент-сервер, где је серверска апликација задужена за обраду захтева и управљање подацима, док клијентска приказује одгворе корисници-

ма и прослеђује захтеве серверској. Оквир *Django* је оквир програмског језика *Python* за развој веб апликација који омогућава једноставно и ефикасно креирање сложене серверске логике, док оквир *Django Rest Framework* омогућава једноставну изградњу *RESTful API*-ја за комуникацију између клијентске и серверске апликације.

Специфичан циљ рада је развој апликације са архитектуралним моделом клијент-сервер која ради са подацима са тренинга корисника и њиховим резултатима. Апликација је названа *Train Wiser*, а њена намена је да корисницима омогући детаљнију анализу тренинга кроз статистичке извештаје, предлагање плана тренинга и предикцију резултата. Клијентска страна апликације, развијена за оперативни систем *Andorid*, захтеве корисника прослеђује серверској апликацији развијеној у програмском језику *Python* уз оквире *Django* и *Django REST Framework*. Кориснички подаци о тренинзима се допремају са апликације *Strava* на којој корисник дозвољава ауторизацију апликације *Train Wiser*. Такође, апликација располаже и са резултатима трка одржаним у Републици Србији на основу којих даје предикцију резултата корисника.

У Глави 2 је описан оперативни систем *Android* кроз његов историјат, концепте развоја апликација на овом оперативном систему и животни век *Android* активности. У Глави 3 је представљен увод у оквир *Django* и дати су детаљни кораци постављања нове *Django* апликације. Такође, дат је осврт на оквир *Django REST Framework* и како његово укључивање у традиционалну *Django* апликацију доприноси проширењу апликације *RESTful API*-јем, олакшавајући комуникацију између система и омогућавајући интеграцију са различитим клијентским апликацијама. Серверски део имплементиране апликације је описан у Глави 4, дати су прегледи додатно употребљених технологија, опис архитектуре и спецификација јавног интерфејса за програмирање апликација. Имплементација и употреба клијентске апликације је представљена у Глави 5 кроз њену архитектуру, ток рада и случајеве употребе, док су могућа унапређења апликације изложена у Глави 6. У последњем поглављу, у Глави 7, изведен је закључак где су сумирани доприноси рада и дати предлози за нове шире могућности развоја и унапређења представљеног решења.

Глава 2

Оперативни систем *Android*

Оперативни систем *Android* је популаран оперативни систем отвореног кода за развој апликација за мобилне уређаје, таблете и паметне телевизоре. *Android* омогућава програмерима да развијају решења које задовољавају разноврсне потребе корисника, од забаве и продуктивности, до апликација за здравље и физичке активности. Званични сервис за дистрибуцију апликација за *Android*-уређаје је *продавница Google Play (енг. Google Play store)* на којој је доступно преко преко три и по милиона апликација[1]. Овај сервис омогућава програмерима да једноставно дистрибуирају своје апликације глобално, уз подршку за различите методе монетизације.

Апликације за *Android* се могу развијати коришћењем разноврсних програмских језика, међу којима су најчешћи програмски језици *Java* и *Kotlin*[2]. Званично развојно окружење за развој апликација за *Android* је *Android Studio* [1] који нуди богат скуп функционалности за побољшање продуктивности програмера који развијају апликације за *Android*, попут напредног едитора кода, визуелног дизајнера графичког корисничког интерфејса (енг. *Graphical User Interface, GUI*), алата за дебаговање (енг. *debugging*), профајлере за анализу перформанси (енг. *profilers*), као и интеграцију са системом за контролу верзија. Захваљујући овом окружењу значајно се олакшава процес развоја и тестирања, чиме се скраћује време потребно за израду апликација.

У наставку ћемо сазнати нешто више о историји *Android*-а, основним концептима развоја апликација за *Android* и животном веку *Android* активности.

[1] Информација из јула 2024. године

[2] У даљем тексту ћемо се усмерити на развој апликација за *Android* уз програмски језик *Java*

2.1 Историјат

Развој *Android-a* је започела америчка компанија *Android Inc.* 2003. године са циљем да се направи оперативни систем за дигиталне камере. Међутим, 2004. циљ развоја се преусмерава на паметне телефоне. *Google Inc.* купује компанију *Android Inc.* 2005. године и одлучује да базира *Android* на оперативном систему *Linux*.

У новембру 2007. године, *Google* најављује оснивање *Open Handset Alliance*-а, конзорцијума од неколико компанија за мобилне телефоне са намерном да се *Android* развија и промовише као бесплатан оперативни систем отвореног кода са подршком за спољне апликације (енг. *third-party applications*). Први мобилни телефон који је користио овај оперативни систем био је *T-Mobile G1*, који је изашао 22. октобра 2008. године. Године 2012. *Android* надмашује *iOS* и постаје најпопуларнији оперативни систем за мобилне уређаје [2].

Android наставља да редовно додаје нове функционалности и побољшава постојеће које објављује кроз нове верзије. Верзије *Android*-а имају јединствену бројну верзију и кодно име које носи назив одређеног слаткиша.

2.2 Основни концепти развоја апликација за *Android*

Android SDK (енг. *Software Development Kit*)[3] представља развојни комплет који садржи библиотеке и алате за развој апликација за *Android*. При прављењу нове апликације, програмери ће одабрати *SDK* верзију која је подржана уз одређену *Android* верзију. Апликација ће се изршавати на уређајима који имају одабрану или вишу *SDK* верзију.

Свака апликација за *Android* представља колекцију екрана који се састоје од једног *распореда (енг. layout)*, једне *активности (енг. activity)* и *ресурса (енг. resources)*. Распореди описују изглед апликације, активности дефинишу шта апликација ради и њен начин комуникације са корисником, док ресурси представљају слике и податке апликације.

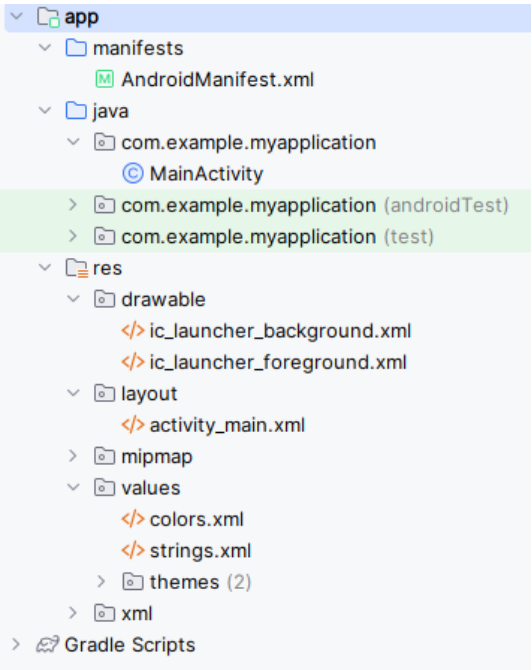
Распоред се чува у *XML* фајлу и може садржати *GUI* компоненте попут дугмади, лабела, поља за унос текста и сл. Распоред можемо да дефинишемо мењајући *XML* фајл ручно или употребом едитора дизајна који је доступан у

[3]У даљем тексту SDK

Android Studio-у. Едитор дизајна омогућава коришћење доступних *GUI* компонената и сређивање њиховог распореда, док ће се *Android Studio* у позадини побринути за унос ових промена у *XML* фајл.

Активности представљају основу апликација за *Android*, то су *Java* класе које одређују који распоред ће се користити и дефинишу реакције апликације на поступке корисника. На пример, уколико је у распореду дефинисано дугме или текстулно поље, активност дефинише како се апликација понаша када корисник кликне на то дугме или унесе текст у текстуално поље.

На слици 2.1 је приказана основна структура *Android* пројекта са једном активношћу `MainActivity.java` и одговарајућим распоредом `activity_main.xml`. Активности се чувају у директоријуму `java`, а распореди на путањи `res/layout`. Ресурсе у виду слика треба чувати у директоријуму `res/drawable/`, боје које се користе у распоредима у фајлу `res/values/colors.xml` и подразумеване текстуалне вредности у фајлу `res/values/strings.xml`. Поред наведених фајлова, од значаја је поменути фајл `AndroidManifest.xml` без кога апликација за *Android* не може да функционише. Овај фајл садржи све важне информације о апликацији - компоненте од којих се састоји, списак потребних библиотека и друге декларације. У овом фајлу је такође одређена главна активност која се прва иницира при покретању апликације.



Слика 2.1: Основна структура *Android* пројекта

Да би се побољшала брзина апликације и смањила потрошња батерије, *Android*-уређаји користе оптимизоване формате за преведени код уместо извршних фајлова језика на коме је код писан. У случају програмског језика *Java*, изворни *Java* код ће се превести у бајткод, а паковање апликације за *Android* у *APK* фајл ће, поред преведеног *Java* кода, садржати библиотеке и ресурсе потребне за рад апликације. *APK* фајл се може извршити на физиком уређају или на виртуелном уређају са *Android*-ом (енг. *Android virtual device, AVD*) који обезбеђује *Android Studio*.

Преведен *Android* код се извршава у извршном *Android* окружењу *ART* (енг. *Android runtime*) директно на процесору уређаја са *Android*-ом обезбеђујући већу уштеду батерије и брже извршавање. Свака апликација се извршава у сопственом процесу чиме се спречава да једна апликација приступи ресурсима друге апликације и уколико нека од апликација престане да ради услед одређене грешке, то неће утицати на друге апликације. На овај начин се доприноси безбедности и сигурности апликација за *Android*. При покретању нове активности апликације, *Android* проверава да ли постоји процес за апликацију којој та активност одговара, ако постоји, користиће њега, у супротном ће направити нови процес. Корисников интерфејс може да ажурира једино главна *Android* нит [3].

2.3 Животни циклус *Android* активности

Android активности наслеђују класу `android.app.Activity` чије методе животног циклуса подразумевано користе. Ове методе можемо надјачати (енг. *override*) уколико потребе апликације коју развијамо то захтевају, уз првобитно обавезно позивање методе наткласе. Животни циклус активности се креће из стања *покренута (енг. launched)*, у стање *извршава се (енг. running)* и завршава у стању *уништена (енг. destroyed)*.

При прављењу активности, позива се њена метода `onCreate()`. Да бисмо обавестили *Android* који распоред активности треба да користи, ову методу треба надјачати додавањем методе `setContentView()`. Овде такође додајемо сва подешавања активности које желимо да се одраде у току њеног стварања. Разлог зашто се за прављење активности не користи конструктор, већ метода, је што *Android* пре прављења активности треба да постави окружење за њу. `onCreate()` мора да се заврши да би се распоред појавио на уређају, стога у

овој методи не смемо имати бесконачне петље.

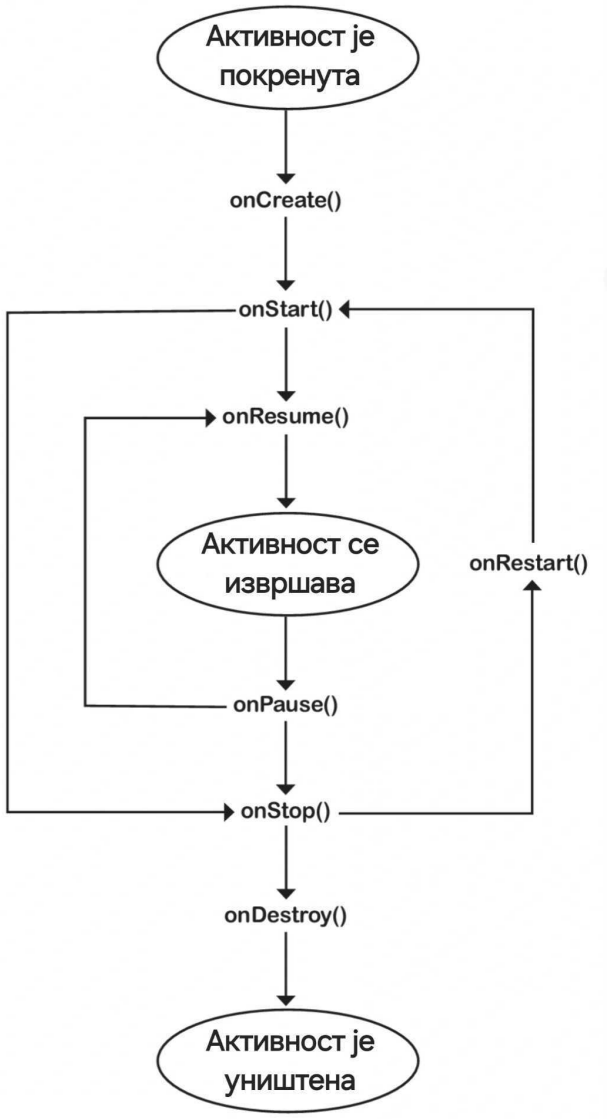
Активност проводи највише времена у стању извршавања и тада је у првом плану на екрану. Активност може бити прекинута наредбом прекида, ако *Android* донесе одлуку да је потребна уштеда простора или да би се поново креирала након промене конфигурације уређаја, попут орјентације или величине екрана. На пример, можда је потребно променити распоред ако је положај екрана уређаја промењен у хоризонталан положај. Ресурси потребни за апликацију могу зависити од конфигурације уређаја, нпр. уколико се промени језик који се користи на телефону, користиће се други скуп ресурса ниски. У случају промене конфигурације уређаја, *Android* ће уништити тренутну активност и поново је направити са ресурсима потребним за нову конфигурацију уређаја. Метод `onDestroy()` се позива непосредно пре уништења активности. Уколико је потребно, ова метода се може надјачати тако да се ослободе ресурси или обаве друга завршна чишћења. Пре позива `onDestroy()`, текуће стање активности се може сачувати позивом методе `onSaveInstanceState()` која као аргумент прима `Bundle` објекат у коме чува стање активности. Како метода `onCreate()` такође има овај аргумент, претходно стање активности се може ресетовати када се активност поново буде правила.

Након методе `onCreate()`, позива се метода `onStart()` која обезбеђује да активност постане видљива кориснику и пређе у стање извршавања. Активност може прећи у стање *заустављена (енг. stopped)* ако се у потпуности сакрије другом активношћу и престане да буде видљива кориснику. У овом случају, активност ће и даље постојати у позадини и задржати све информације стања. Овај прелазак обезбеђује метода `onStop()` која се такође позива и пре методе `onDestroy()` ако ће активност бити уништена. У овом случају, потребно је позвати методу `onSaveInstanceState()` пре позива `onStop()` методе. Метода `onRestart()` се позива када активност постаје поново видљива након што је била у стању заустављена.

Уколико активност изгуби фокус, тј. престане да буде у првом плану, али је и даље видљива, онда се она налази у стању *паузирана (енг. paused)*. У овом случају, активност и даље ради и задржава све информације о стању. Активност прелази из стања извршавања у ово стање методом `onPause()`, а враћа се у стање активности методом `onResume()`. Ова метода се позива и након `onStart()` методе како би активност прешла у први план. Ако је активност видљива, а никада није у првом плану, онда ће се методе `onPause()`

и `onResume()` прескочити. Такође, када долази до уништавања активности, метода `onPause()` ће се позвати пре методе `onStop()`.

Целокупан описан процес је представљен на слици 2.2 [3]. Опис израде сложеније апликације за *Android* ће бити приказан у поглављу 5.



Слика 2.2: Животни циклус *Android* активности

Глава 3

Оквири *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 инжењера 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 – CGI скрипте су морале да имају доста поновљеног шаблонског кода, што је усложњавало његово вишеструко искоришћење и отежавало програмерима разумевање пројеката 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-a*. 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 несистематичности и дуплирања 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 [4].

3.1 Основни концепти

Django [4, 5] је веб оквир програмског језика *Python* који је први пут објављен 2005. године. Заснива се на филозофији 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. Његова главна предност је управљање свим захтевнијим деловима израде веб сајта као што је аутентификација, повезивање са базом, безбедност итд.

Django такође следи принцип *"без понављивања" (енг. "Don't Repeat Yourself", DRY)*. Ово је основни принцип програмског развоја који промовише избегавање дуплирања кода. Како би подстакао програмере да пишу код који није редудантан, *Django* има неколико уграђених механизама попут система шаблона, наслеђивања модела, међуслојног оквира (енг. *middleware*) који омогућава дефинисање поновно употребивих компоненти за обрађивање *HTTP* захтева и одговора и др.1425

Овај оквир је отвореног кода и доступан је на репозиторијуму за додатке за програмски језик *Python*, *Python Package Index (PyPI)*[1] као и сви његови додатни пакети који су на располагању програмерима. У наставку су детаљније описане главне карактеристике овог оквира.

3.1.1 Постављање и конфигурација *Django* пројекта

Нови *Django* пројекат се креира командом

[1] https://pypi.org

```
django-admin startproject $project_name $storing_dir
```

Први аргумент команде `startproject` је име које желимо да дамо новом пројекту. Други аргумент дефинише директоријум који ће садржати основне фајлове и структуру пројекта. Уколико га не дефинишемо, додатни директоријум са именом пројекта ће се креирати у тренутном радном директоријуму.

У случају да корисник позове претходну команду на следећи начин:

```
django-admin startproject django_project_name .
```

Django ће у тренутном радном директоријуму креирати наредну структуру пројекта:

```
|-- django_project_name
|    |-- __init__.py
|    |-- asgi.py
|    |-- settings.py
|    |-- urls.py
|    |-- wsgi.py
|-- manage.py
```

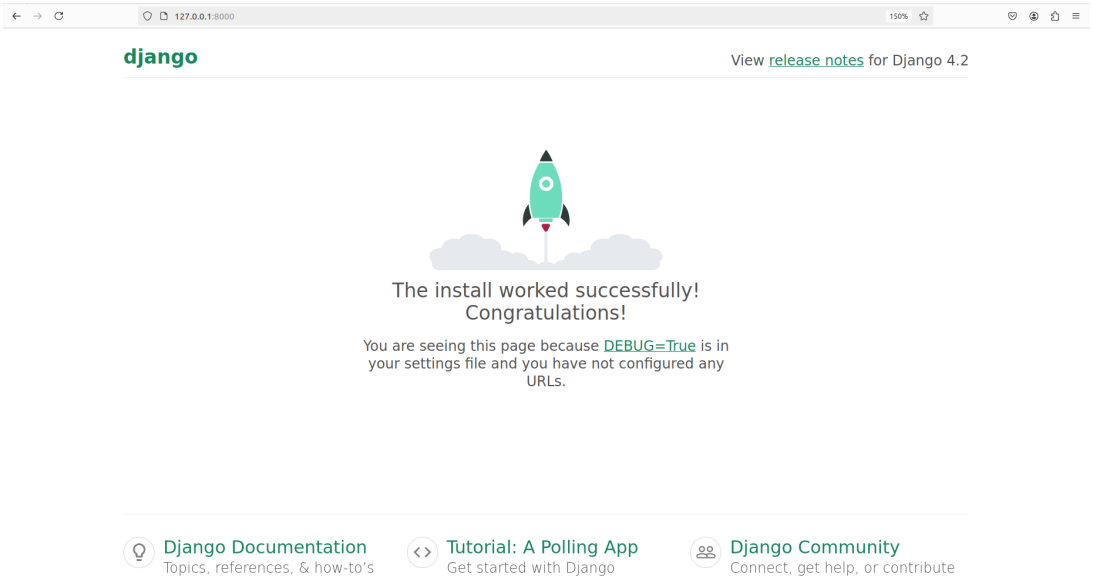
Од највеће важности за стандардне пројекте су наредни фајлови:

- `settings.py` – контролише општа подешавања *Django* пројекта,
- `urls.py` – служи за рутирање *HTTP* захтева и
- `manage.py` – скрипта намењена за извршавање различитих *Django* команди.

Коришћењем `manage.py` скрипте, *Django* пројекат се покреће на локалном веб серверу на следећи начин:

```
python manage.py runserver
```

Да би се потврдило да је пројекат успешно инсталиран, потребно је отворити адресу `http://127.0.0.1:8000` у веб прегледачу. У случају успешног покретања, корисник ће добити страницу са изгледом приказаним на слици 3.1.



Слика 3.1: Изглед веб странице иницијално постављеног *Django* пројекта

Традиционалан *Django* веб сајт се састоји од једног пројекта са више апликација, где свака апликација представља одвојену функционалност. За прављење нове апликације унутар пројекта се користи наредна команда:

```
python manage.py startapp $app_name
```

где је `$app_name` име апликације која се креира. У случају конкретног позива

```
python manage.py startapp new_app
```

креираће се директоријум `new_app` са следећом структуром:

```
|-- new_app
|     |-- __init__.py
|     |-- admin.py
|     |-- apps.py
|     |-- migrations
|           |-- __init__.py
|     |-- models.py
|     |-- tests.py
|     |-- views.py
|-- django_project (prethodno kreiran)
```

где су:

– `admin.py` – конфигурациони фајл за уграђену апликацију *Django Admin*,

– `apps.py` – конфигурациони фајл за новокреирану апликацију,

– `migrations/` – директоријум за чување миграционих фајлова за промене у бази података,

– `models.py` – модул за дефинисање модела базе података,

– `tests.py` – модул за тестове новокреиране апликације, а

– `views.py` – модул за обраду логике *HTTP* захтева и одговора.

Додатно, потребно је креирати фајл `urls.py` за рутирање на нивоу апликације.

Да би *Django* препознао и учитао апликацију, неопходно је да се региструје у конфигурацији `INSTALLED_APPS` у конфигурационом фајлу `django_project/settings.py`:

```
# django_project/settings.py

INSTALLED_APPS = [
    ...
    "new_app.apps.NewAppConfig"
    ...
]
```

3.1.2 *MVT* архитектура

Архитектура оквира *Django* се темељи на обрасцу *Модел-Поглед-Шаблон (енг. Model-View-Template, MVT)* који омогућава јасно раздвајање између управљања подацима (модели), пословне логике (погледи) и презентације (шаблони). За моделе се користи фајл `models.py`, за погледе `views.py`, а за шаблоне директоријум `templates/`. У наставку су детаљније описане компоненте ове архитектуре.

3.1.2.1 Модели

Модели представљају апстракцију базе података и омогућавају рад са подацима на високом нивоу кроз интеракцију са базом података, без директног коришћења *SQL* упита. Модели дефинишу структуру табела у бази података кроз *Python* класе. Атрибути класе представљу колоне у табели, док инстанце класе представљу редове, тј. записе. Атрибути класе модела, познатији као *поља (Fields)*, дефинишу тип и карактеристике података у бази. На располагању су разноврсни типова поља, укључујући текстуални `(CharField)`, целобројни `(IntegerField)`, датумски `(DateField)`, тип података за електронску пошту `(EmailField)`, *URL* адресе `(URLField)` и др. Списак свих доступних типова као и њихови детаљни описи су доступни у званичној *Django* документацији о типовима поља[2].

Рад са базом података без коришћења *SQL* упита је омогућен захваљујући *Објектно-Релационом мапирању (енг. Object-Relational Mapping, ORM)* које прави мост између објектно-орјентисаног програмирања и релационих база података. У оквиру *Django*, *ORM* обезбеђује једноставно и ефикасно извођење операција креирања, читања, ажурирања и брисања *(енг. CRUD - create, read, update, delete)* над базом података.

На пример, за креирање табеле са ауторима која садржи колоне са јединственим идентификатором, именом, презименом и датумом рођења аутора, потребно је креирати класу са наведеним атрибутима у фајлу `models.py`.

```
# django_project/models.py

from django.db import models

class Author(models.Model):
    id = models.PositiveBigIntegerField(primary_key=True)
    first_name = models.CharField()
    last_name = models.CharField()
    birth_date = models.DateField()
```

Креирање новог уноса у табелу се ради инстанцирањем направљене класе и коришћењем методе `save()` које је неопходна за ажурирање базе података.

[2]https://docs.djangoproject.com/en/5.0/ref/models/fields/

```
author = Author(id=1,
                first_name="John",
                last_name="Doe",
                birth_date="1980-01-01")
author.save()
```

У наредном коду су приказани примери читања свих аутора и аутора са идентификатором вредности 1 из базе.

```
authors = Author.objects.all()
author = Author.objects.get(id=1)
```

Наредни код мења презиме аутора са идентификатором вредности 1.

```
author = Author.objects.get(id=1)
author.last_name = "Smith"
author.save()
```

Брисање записа из базе се ради коришћењем методе `delete()` над конкретном инстанцом одговарајуће класе.

```
author = Author.objects.get(id=1)
author.delete()
```

Једна од важних могућности модела је **валидација података**. *Django* пружа уграђене механизме за валидацију кроз поља модела. На пример, `CharField` може имати ограничење за максималну дужину текста:

```
first_name = models.CharField(max_length=30)
last_name = models.CharField(max_length=30)
```

Такође, могуће је дефинисати методе за валидацију унутар модела. У наредном примеру је додато ограничење да датум рођења аутора мора бити мањи од тренутног.

```
# django_project/author/models.py

from django.db import models
from django.core.exceptions import ValidationError
from datetime import datetime
```

```
class Author(models.Model):
    id = models.PositiveBigIntegerField(primary_key=True)
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    birth_date = models.DateField()

    def check_birth_date(self):
        if self.birth_date >= datetime.date.today():
            raise ValidationError("Birth date cannot be
                                   future date.")
```

Миграције у *Django*-у [6] су механизам за примењивање извршених промена насталих у моделима у бази података. Овај процес је веома важан за одржавање конзистентности и интегритета података у апликацији. Свака промена у моделима у *Django* апликацији захтева креирање миграције. Миграције треба правити приликом додавања новог модела, измена постојећег или брисања модела. *Django* пружа неколико команди које олакшавају рад са миграцијама:

- `migrate` – за примењивање и повлачење миграције у базу,
- `makemigrations` – креира нове миграције на основу промена у моделима,
- `sqlmigrate` – приказује *SQL* упите који ће бити извршени за одређену миграцију и
- `showmigrations` – приказује листу миграција и њихов статус (да ли су примењене или нису).

Наведене команде се покрећу над фајлом `manage.py`. Прво је потребно направити миграциони фајл са командом `makemigrations`,

```
python manage.py makemigrations
```

а затим применити све непримењене миграције командом `migrate` која ће синхронизовати базу података са тренутним стањем модела.

```
python manage.py migrate
```

3.1.2.2 Погледи

Функција погледа (енг. view function) [7] је *Python* функција која као улазну вредност добија веб захтев, а на излазу враћа веб одговор. Одговор може бити садржај *HTML* странице, преусмеравање, грешка, документ и сл.

Како би се одговор специфичне поглед функције приказао, потребно је конфигурисати фајл `urls.py` који је задужен за рутирање у *Django* апликацијама. Овај фајл дефинише *URL* руте и повезује их са одговарајућим поглед функцијама. Када корисник пошаље захтев на одређену *URL* адресу, *Django* на основу овог фајла проналази одговарајућу поглед функцију која ће обрадити захтев и вратити одговор. За дефинисање рутирања, обично се користи функција `path()` која као први аргумент узима назив *URL* шаблона у облику ниске, други аргумент представља одговарајућу поглед функцију, док опционо може имати трећи аргумент за именовање рута које се може користити у шаблонима.

Наредни пример приказује све ауторе из базе података на рути `"authors/"` користећи шаблоне који су објашњени у наставку.

```
# django_project/author/urls.py

from django.urls import path
from .views import home_view, authors_view

urlpatterns = [
    path('authors/', authors_view, name='authors'),
]
```

```
# django_project/author/views.py

from django.shortcuts import render
from .models import Author

def authors_view(request):
    authors = Author.objects.all()
    return render(request, 'authors.html',
                  {'authors': authors})
```

3.1.2.3 Шаблони

Шаблони у *Django*-у представљају концепт који омогућава програмерима да одвоје презентациони од логичког слоја апликације, чиме се постиже боља организација и одржавање пројекта. *Језик Django шаблона (енг. Django template language, DTL)* [8] омогућава укључивање динамичких елемената у статичке *HTML* шаблоне. Ови елементи се називају *тагови* и понашају се слично као програмске контролне структуре (попут наредби `if`, `for`...), али се не извршавају као *Python* код. Стога, није могуће навести обичан *Python* код, већ само оно што *DTL* подржава. Шаблон садржи промењљиве које се евалуирају у вредности и тагове који контролишу логику. Тагови се увек наводе у формату `{% tag %}`. За апликацију `author`, *Django* ће подразумевано тражити шаблон на путањи `author/templates/`.

Конвенција је да се унутар директоријума `templates` креира поддиректоријум са истим именом као апликација у коме ће се сместити шаблони. У наредном примеру је приказан шаблон за пролазак кроз све ауторе у бази и њихово приказивање.

```
<!-- author/templates/author/author_list.html -->

<h1>All authors</h1>
{% for author in author_list %}
<ul>
    <li>Id: {{ author.id }}</li>
    <li>Name: {{ author.first_name }}</li>
    <li>Surname: {{ author.last_name }}</li>
    <li>Birth date: {{ author.birth_day }}</li>
</ul>
{% endfor %}
```

3.2 *Django REST Framework*

Скуп протокола и инструкција које одређују како ће две програмске компоненте остварити директну комуникацију представља *програмски интерфејс апликације (енг. Application Programming Interface, API)*. Када је *Django* настао, већина веб сајтова је била заснована на апикацијама са монолитном

архитектуром у којима су презентациони слој, логички слој као и база података интегрисани унутар једне самосталне целине. Данас се често користи *приступ који даје приоритет API-ју (енг. API-first approach)* где се апликација раздваја на две компоненте, клијентску (енг. *frontend*) и серверску (енг. *backend*) апликацију, које комуницирају путем *API*-ја. Клијентска апликација је задужена за презентациони слој и прослеђивање захтева серверској апликацији која управља логичким слојем и базом података. Један *API* може истовремено подржати више различитих клијентских апликација развијених у различитим програмским језицима и оквирима. Такође, уколико се клијентска апликација ажурира новијим верзијама алата, *API* може остати исти.

Код веб *API*-ја, преовладава архитектурни образац *REST (REpresentational State Transfer)* [9] чији је кључни појам ресурс. Ресурс може бити било који ентитет довољно битан да може бити референциран. Сваки ресурс мора имати свој *URL*. Да би *API* био *RESTful API*, неопходно је да испуњава наредна три својства:

1. Да нема стања (енг. *stateless*) као и *HTTP* протокол над којим је *RESTful API* изграђен[3]
2. Подржава уобичајене *HTTP* методе (*GET*, *POST*, *PUT*, *DELETE*, итд.)[4]
3. Враћа податке у *JSON* или *XML* формату

Оквир *Django REST Framework (DRF)* омогућава програмерима да постојећи *Django* пројекат трансформишу у веб *API*. Главни недостатак приступа који даје приоритет *API*-ју је што захтева више конфигурације него традиционална *Django* апликација. Међутим, *DRF* поседује механизме који олакшавају имплементацију веб *API*-ја. Да би се постојећа традиционална *Django* апликација трансформисала у веб *API*, потребно је ажурирати фајл `urls.py`, додати *DRF* погледе и креирати серијализатор (енг. *serializer*). Серијализатор претвара комплексне податке (нпр. *Django* моделе) у формате за слање серијализованих података које је једноставно користити, најчешће у *JSON* или *XML* формат и прима и валидира податке у овим форматима пре чувања у бази.

[3]Сваки пар *HTTP* захтев -> *HTTP* одговор је независан од претходног и не постоји меморија која чува стање између захтева

[4]Метода *HTTP GET* се користи за добијање ресурса, *POST* за креирање нових ресурса, *PUT* за ажурирање постојећих ресурса, *DELETE* за брисање ресурса, итд.

Да би се *DRF* користио у пројекту, потребно га је инсталирати користећи *PyPI* и регистровати у конфигурацији `INSTALLED_APPS` у конфигурационом фајлу `django_project/settings.py`:

```
# django_project/settings.py

INSTALLED_APPS = [
    ...
    "rest_framework",
    ...
]
```

Уколико се сва *API* логика смешта на рути `"api/"`, потребно је направити апликацију `api` и ажурирати фајл `django_project/urls.py` у складу са претходно наведеним објашњењима за традиционалну *Django* апликацију. Уколико желимо да направимо *API* за приказ свих аутора у *JSON* формату, *URL* фајл за апликацију `apis/urls.py` је потребно ажурирати тако да на рути `""` приказује поглед `AuthorAPIView` који ћемо креирати.

```
# apis/urls.py

from django.urls import path

from .views import AuthorAPIView

urlpatterns = [
    path("", AuthorAPIView.as_view(), name="author_list"),
]
```

Погледи *DRF*-а се разликују од погледа традиционалне *Django* апликације у формату података које враћају, традиционална *Django* апликација ће враћати садржај *HTML* странице, док *DRF* враћа серијализоване податке у медијским форматима попут формата *JSON*, *XML*, итд. Наредни код користи генеричку класу `ListAPIView` за читање свих инстанци аутора. Потребно је дефинисати атрибуте `queryset` и `serializer_class` ове класе коју ће наша класа `AuthorSerializer` која је наслеђује користити. Атрибуту `queryset` додељујемо инстанце свих аутора, док класу за серијализатор, `serializer_class`,

постављамо на `AuthorSerializer` коју ћемо дефинисати у модулу за серијализатор.

```
# apis/views.py

from rest_framework import generics
from books.models import Author
from .serializers import AuthorSerializer

class AuthorAPIView(generics.ListAPIView):
    queryset = Author.objects.all()
    serializer_class = AuthorSerializer
```

Приликом дефинисања серијализатора, потребно је обезбедити модел базе који ће се читати и поља која желимо да прикажемо.

```
# apis/serializers.py

from rest_framework import serializers
from books.models import Author

class AuthorSerializer(serializers.ModelSerializer):
    class Meta:
        model = Author
        fields = ("id",
                  "first_name",
                  "last_name",
                  "birth_date")
```

Уколико су претходни кораци успешно одрађени, након покретања *Django* пројекта на локалном веб серверу, на рути `"api/"` ћемо моћи да видимо списак свих аутора из базе података у *JSON* формату.

Овим смо обухватили основне концепте и функционалности оквира *Django* и *Django REST Framework*, показујући како се користе за развој робусних и скалабилних веб апликација. У поглављу 4 ћемо приказати његову конкретну употребу на сложенијој апликацији.

Глава 4

Серверска страна апликације *Train Wiser*

Употреба технологија описаних у претходним поглављима представљена је у практичном делу рада - креираној апликацији *Train Wiser* [10]. Апликација пружа статистку и асистенцију за тренирање и напредак вежбача на основу корисничких података увезених са апликације *Strava* путем њеног јавног програмског интерфејса и на основу резултата трка одржаних у Републици Србији. Апликација *Train Wiser* је намењена тркачима који желе да детаљније прате своје перформансе и да унапреде резултате.

За архитектуралну организацију апликације одабран је модел клијент-сервер. У овом моделу, клијентска апликација (енг. *frontend*) је задужена за комуникацију са корисником и прослеђивање захтева серверској апликацији (енг. *backend*) која је задужена за унутрашњу логику. У овом поглављу су представљене кључне тачке и додатне технологије коришћене у изради серверског дела апликације, њена архитектура и јавни интерфејс за програмирање апликација.

4.1 Кључни аспекти при развоју апликације

За израду серверског дела апликације, коришћен је програмски језик *Python* уз оквире *Scrapy*, *Django* и *Django REST Framework*. У наставку су кроз кључне тачке при развоју апликације издвојени описи неких од додатно употребљених технологија.

4.1.1 Аутентификација и ауторизација

Аутентификација и ауторизација представљају кључне концепте безбедности система.

Аутентификација је процес доказивања идентитета 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 податке за аутентификацију како би се аутентификовала на друштвеној мрежи, она поред приступа листи пратилаца добија потпун приступ корисничком налогу. Такође, апликација би имала могућност да сачува корисничке податке за аутентификацију за друштвену мрежу, као и да то уради на небезбедан начин. У случају да се апликација компромитује, подаци за аутентификацију 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 података за аутентификацију 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 посредништва корисника.

Принцип рада протокола се заснива на следећим корацима:

1. Регистрација апликације
2. Добијање приступног токена
3. Преузимање захтеваних ресурса посредством приступног токена
4. Коришћење токена за обнову приступног токена

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 податке за аутентификацију 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 случајева 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

Добијање приступног токена

Након успешне регистрације, спољни сервис ће апликацији доделити *приступни токен (енг. access token)*. Приступни токен обезбеђује приступ заштићеним дигиталним ресурсима, ограничавајући друге начине приступа и тиме побољшавајући сигурност. Овај 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.

Уколико спољни сервис подржава обнављављање приступног токена коришћењем *токена за обнову (енг. refresh token)*, овај токен ће бити додатно достављен уз приступни токен. Токен за обнову има дуже време трајања од приступног токена и служи за добијање новог приступног токена када постојећи истекне.

Преузимање захтеваних ресурса посредством приступног токена

Након што се корисник аутентификује на спољни сервис на који га је преусмерила апликација и прихвати 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.

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 апликације које нису од поверења 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 апликација које нису од поверења. 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 апликације која није од поверења, 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 [11].

Апликација *Strava* користи *OAuth 2.0* [12] за аутенификацију са *V3 API*-ем, јавним интерфејсом ове апликације који даје могућност програмерима да приступе подацима ове апликације. Апликација *Train Wiser* је преко овог протокола увезана са апликацијом *Strava* како би имала приступ свим активностима корисника који одобре ауторизацију.

4.1.2 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 табелу `StravaActivity` у бази података. Информације о новим активностима корисника се уписују у ову табелу након што 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 оквира *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.

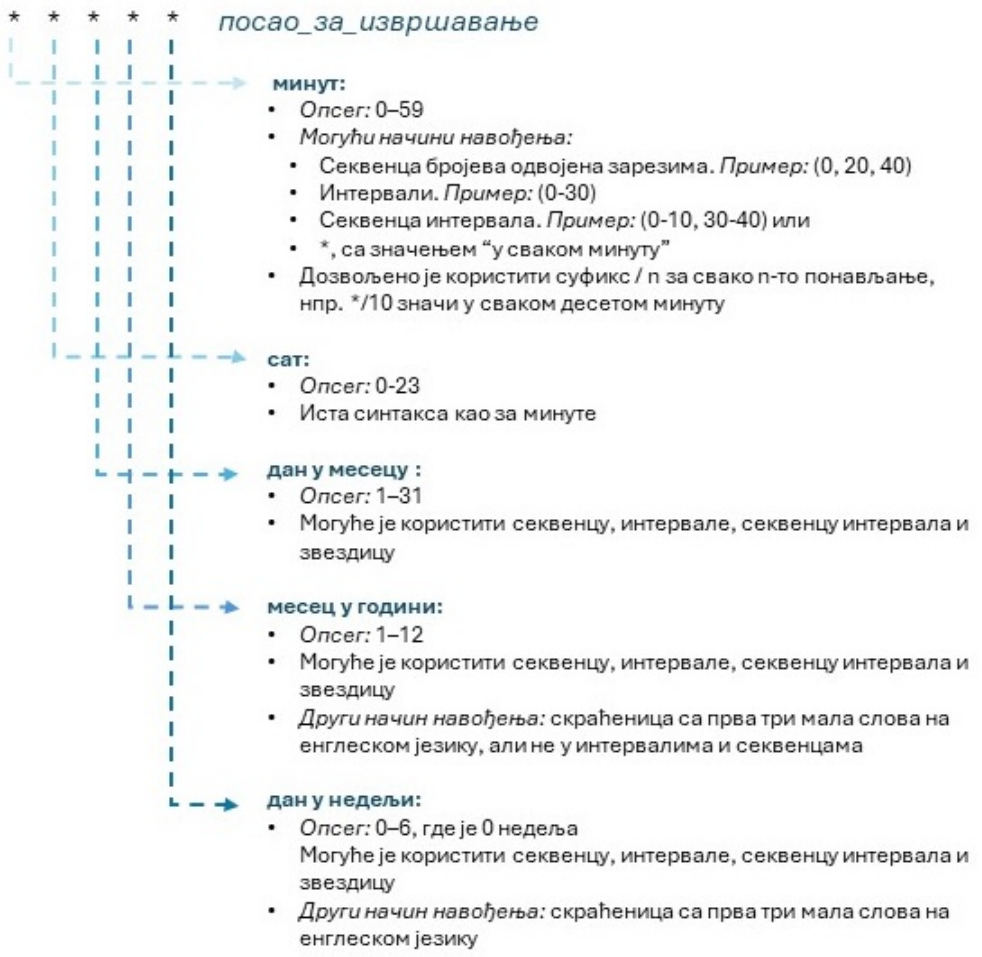
4.1.2.1 Django библиотека *django-crontab*

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 [13]. Један ред ове датотеке представља један *Cron-посао (енг. cronjob)* [14]. Синтакса за навођење *Cron* послова у `crontab` датотеци је приказана на слици 4.1 [13].

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, 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



Слика 4.1: Синтакса конфигурационе датотеке `crontab`

библиотеке *django-crontab* [15] која у позадини користи системски процес *Cron*. *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:

```
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.

4.1.2.2 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 [16, 17].

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. податке за аутентификацију апликације за приступ *Strava* ресурсима, поменуте у потпоглављу 4.1.1, 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 (`hub.verify_token`) који ће апликација *Strava* узвратити слањем верификационог *GET* захтева ка `callback` 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 апликацији *Strava*. Након слања одговора, апликација ће на рути са које је захтев за регистрацију послат добити `subscription_id` који ће служити за проверу валидности приликом слања догађаја мрежних кука чиме се затвара процес регистрације [17].

Регистрација за мрежне куке апликације *Strava* у апликацији *Train Wiser* се покреће једнократно када апликација добије *POST* захтев ка рути `webhook_subscription`. Као што је наведено у потпоглављу 3.1.2.2, функције за обраду *HTTP* захтева који се упућују ка одређеној рути се наводе у фајлу `urls.py`. У случају `webhook_subscription` руте, одговарајућа функција је `webhook_subscription` дефинисана у `views.py`. Ова функција ће проверити методу *HTTP* захтева, и у случају *POST* методе, послати *POST* захтев ка адреси `https://www.strava.com/api/v3/push_subscriptions` са горенаведеним неопходним параметрима.

Функција `webhook_callback` је задужена за обраду захтева са `callback` адресе. Према наведеном објашњењу, на ову адресу се шаљу верификациони *GET* захтев за регистрацију апликације и догађаји мрежних кука у виду *POST* захтева. У случају *GET* захтева, функција ће проверити да ли су прослеђени параметри `hub.mode` и `hub.challenge`, као и да ли је верификациони токен једнак токену који је апликација проследила. Функција ће у одговору апликацији *Strava* проследити параметар `hub.challenge` чиме успешно за-

твара процес верификације. Комплетирање процеса регистрације, након што *Strava* пошаље параметар `subscription_id` на руту `webhook_subscription` ће одрадити функција `webhook_subscription` чувањем идентификатора регистрације у табели `StravaSettings` у бази података. Осигуравање да се регистрација не ради више од једанпут се ради провером вредности овог параметра у табели на почетку позива функције. Регистрација ће се радити једино у случају да параметар већ није уписан, тј. уколико је вредност у табели једнака `NULL`.

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

Догађаји мрежних кука се обрађују на `callback` адреси за коју је задужена фјункција `webhook_callback`. Након што корисник постави нову активност и апликација *Train Wiser* добије *POST* захтев, ова функција ће проверити `subscription_id`, и у случају једнакости са вредношћу из `StravaSettings` табеле, уписати идентификаторе активности и вежбача чија активност је детектована у табелу за чување активности, `StravaActivity`. 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 табели `StravaAthlete`.

4.1.3 Формирање скупа података са резултатима трка

Скуп података са резултатима трка је формиран на основу резултата тркача доступних на сајтовима *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 *екстракције 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 [18].

Екстракција података са веб страница у апликацији *Train Wiser* је одрађена коришћењем оквира *Scrapy*. *Scrapy* је оквир отвореног кода програмског језика *Python* за обилазак 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 претраживање страница 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 [19].

За потребе *Train Wiser* апликације, екстрактовани подаци су експортовани у *JSON* формат. На основу излазних фајлова екстрактора, оформљен је јединствен *JSON* фајл који садржи речник чији су кључеви имена тркача, а вредности речници чији кључеви представљају могуће дистанце трка (5, 7, 7.7, 10, 21 и 42 километара) и вредности листа са информацијама о резултатима тог учестника на свим тркама са одговарајућом километражом. Број укупно сакупљених података са сајтова приказан је у табели 4.1, док је укупан број тркача који имају информацију о учешћу на барем једној трци одржаној у Републици Србији 44077.

Сајт	**Укупан број резултата релевантних трка**
runtrace.net	20908
trka.rs	42833
bgdmarathon.org	49742

Табела 4.1: Број података са сајтова за резултате трка

4.2 Архитектура апликације

Серверска страна апликације је подељена у више модуларних компоненти (*Django* апликација) задужених за обраду унутрашње логике. У наредном тексту је објашњен начин рада и имплементације серверске апликације кроз компоненте које садржи.

4.2.1 Компонтента за рад са корисничким налозима

За рад са корисничким налозима је задужена имплементирана компонента `users`. За креирање модела корисника одабрана је класа `AbstractUser` из библиотеке `django.contrib.auth.models` која служи као основа за креирање корисничког модела. У апликацији *Train Wiser* је направљен модел `CustomUser` надоградњом овог модела додавањем страног кључа ка табели `StravaAthlete` и додатним ограничењима за атрибуте модела.

Коришћењем библиотеке *Django OAuth Toolkit* [20] која обезбеђује све *крајње тачке (енг. endpoints)*, податке и логику потребне за додавање *OAuth2* функционалности *Django* пројектима се омогућава пријављивање са клијентске стране на серверску преко протокола *OAuth2*. Коришћен је тип одобрења (енг. *grant type*) *password* уз који клијентска апликација на основу корисничког имена и његове лозинке добија приступни токен који може користити у другим захтевима ка серверској апликацији намењеним за тог корисника. Поред приступног токена, апликација добија токен за обнову и време трајања приступног токена. Тип одобрења са разменом лозинке је погодан за коришћење само када су клијентска и серверска страна у истом власништву, док се за приступ апликацијама треће стране саветује коришћење поузданијих типова одобрења.

Библиотека *Django OAuth Toolkit* има уграђене крајње тачке за пријављивање и одјављивање корисника, док је за регистрацију корисника потребно направити нову крајњу тачку апликације. *API* који компонента `users` пружа, поред регистровања корисника, омогућава једноставно добијање информација о ауторизованом кориснику, мењање параметара корисничког профила и брисање корисничког налога коришћењем класе `generics.RetrieveUpdateDestroyAPIView` оквира *Django REST Framework*.

4.2.2 Компонтента за руковање захтевима ка апликацији *Strava*

Компонента `strava_gateway` садржи имплементацију за три модела, `StravaAthlete`, `StravaActivity` и `StravaSettings`. Модел `StravaAthlete` садржи идентификатор корисника на апликацији *Strava* и податке за приступ овој апликацији које чине приступни токен, токен за обнову и време истека приступног токена. Додатно, у овом моделу се чувају подаци о границама за сваку зону срчаног пулса корисника који се добијају засебним *GET* захтевом и поље `backfill_progress` за чување информације докле се стигло са допремањем претходних активности корисника у базу након ауторизације са апликацијом *Strava*. Ово поље је иницијално постављено на 0, а претходне активности корисника ће се допремати преко *Cron* задатка све док вредност овог поља не постане `NULL` након што су све активности сачуване у бази. Ако се у току једне итерације позивања *Cron* задатка не допреме све активности, `backfill_progress` се поставља на вредност датума почетка последње допремљене активности како би *Cron* задатак знао одакле да настави при следећем позиву.

Модел `StravaActivity` садржи све информације о активностима које су иницијално доступне за активности на *Strava API*-ју и додатна поља попут просечне зоне срчаног пулса, поље за чување информације да ли је активност трка и заокружену вредност дистанце трке. Поље за просечну зону срчаног пулса у току активности се израчунава на основу граница зона срчаног пулса из модела `StravaAthlete` и вредности просечног срчаног пулса у току активности. Ово поље је додато како би се лакше разликовао тип тренинга и јер се границе зона разликују од особе до особе, те је и приликом увида у туђе активности јасније колико је активност захтевна. Табела `StravaActivity` садржи страни кључ ка табели `StravaAthlete` који је повезан преко поља за индентификацију корисника.

Модел `StravaSettings` има два поља, `setting_key` и `setting_value`. У овој табели се чува `subscription_id` потребан за мрежне куке апликације *Strava*, а табела се може допунити другим подешавањима по потреби.

Ова компонента не пружа *API* за директан рад са клијентском апликацијом већ обезбеђује доступност прилагођених података са апликације *Strava* другим компонентама и функције за обраду захтева ка овој апликацији које

смо описали у потпоглављима 4.1.1 и 4.1.2. Такође, у оквиру ове компоненте је направљена *Django* команда за експортовање свих активности корисника у *CSV* фајл што омогућава њихову прилагоднију употребу за различите анализе и развој нових функционалности. Команда се позива са:

```
python manage.py export_to_csv
```

4.2.3 Компонтента за генерисање планова тренинга

Компонента `trainings` пружа *API* који на улазу прима дистанцу трке и жељени временски резултат корисника, а враћа до највише пет планова тренинга који одговарају тренинзима корисника апликације који су остварили најприближније резултате траженом. За дистанце дужине 5km, компонента враћа шестонедељне планове, за дистанце дужине 7km, 7.7km и 10km осмонедељне, 21km дванестонедељне и за 42km шеснаестонедељне. Планови тренинга се генеришу на основу табеле `StravaActivity` из базе података тако што се посматрају сви тркачки резултати на тркама жељене дистанце и узима се пет најприближнијих резултата циљаном. Затим се израчунава број дана трајања тренинг плана за тражену дистанцу и за корисника који је остварио тај резултат се узимају сви његови тренинзи у оквиру тог периода.

Осим постигнутог резултата на трци, алгоритам враћа и процентуалну разлику у границама зона срчаног пулса корисника за кога се генерише план тренинга у односу на зоне корисника коме припадају тренинзи на основу којих се тренинг план предлаже. Ова разлика се доставља ако су оба корисника повезала апликацију *Train Wiser* са апликацијом *Strava* и имају доступну информацију о границама зона срчаног пулса.

API компоненте као одговор враћа речник чији кључеви представљају резултат на трци уз опциону разлику у зонама срчаног пулса, а вредности тренинзи који су претходили резултату одређеног корисника апликације.

4.2.4 Компонтента за статистичке извештаје о тренинзима

Компонента `stats` пружа *API* са извештајима о тренинзима за задати месец и годину корисника која се прослеђује у оквиру *URL* путање. Ова компонента израчунава које недеље у години су покривене одабраним месецом

и за њих даје информације о свим тренинзима које укључују тип тренинга, пређену дистанцу, трајање тренинга, просечну зону срчаног пулса и датум тренинга. Такође, за сваки тип тренинга се прави месечни извештај са укупним трајањем, укупном дистанцом и просечном зоном срчаног ритма свих тренинга тог типа у оквиру одабраних недеља.

Информације које компонента пружа се израчунавају и генеришу на основу података из табеле `StravaActivity`. Стога, да би клијентска апликација могла да користи *API* ове компоненте за тренутног корисника, потребно је да је корисник аутентификован и да има налог на апликацији *Strava* коју је повезао са апликацијом *Train Wiser*.

4.2.5 Компонтента за предикцију резултата на наредној трци

Предикција резултата на трци, за коју је задужена компонента `results_predictor`, је одрађена коришћењем алгоритма *Експоненцијално пондерисана покретна средина (енг. Exponentially Weighted Moving Average, EMWA)* [21] који се користи за моделовање и опис временских серија.

Основна идеја овог алгоритма је да се свака вредност у серији пондерише фактором који експоненцијално опада како се удаљава од тренутног временског тренутка. Формула на којој се *EWMA* заснива је:

$$\mathrm{EWMA}_t = \alpha \cdot x_t + (1 - \alpha) \cdot \mathrm{EWMA}_{t-1}$$

Потребно је да корисник обезбеди *параметар изглађивања* α *(енг. smoothing factor)* који узима вредности између 0 и 1.

- Када је $\alpha > 0.5$ – Алгоритам брже реагује на промене у подацима
- Када је $\alpha < 0.5$ – Брзина опадања тежине старијих података је мања у односу на случај када је α већа од 0.5

Уколико развијемо рекурзивну *EWMA* формулу,

$$
\begin{aligned}
\mathrm{EWMA}_t &= \alpha \cdot x_t + (1-\alpha) \cdot \mathrm{EWMA}_{t-1} \\
&= \alpha \cdot x_t + (1-\alpha) \cdot [\alpha \cdot x_{t-1} + (1-\alpha) \cdot \mathrm{EWMA}_{t-2}] \\
&= \alpha \cdot x_t + \alpha \cdot (1-\alpha) \cdot x_{t-1} + (1-\alpha)^2 \cdot \mathrm{EWMA}_{t-2} \\
&\vdots \\
&= \alpha \cdot x_t + \alpha \cdot (1-\alpha) \cdot x_{t-1} + \alpha \cdot (1-\alpha)^2 \cdot x_{t-2} + \ldots \\
&\quad + \alpha \cdot (1-\alpha)^k \cdot x_{t-k} + \ldots
\end{aligned}
$$

можемо закључити да тежина за податак x_{t-k} износи $\alpha \cdot (1-\alpha)^k$, а како је α између 0 и 1, тежина постаје мања што је k веће, тј. тежина података има мању вредност што су подаци старији.

За компоненту `results_predictor` је одабрана имплементација овог алгоритма библиотеке *Pandas* са параметром $\alpha = 0.9$. Компонента пружа *API* који на основу имена тркача (које добија из модела `CustomUser` на основу приступног токена) и дистанце трке даје процену за резултат на наредној трци на основу историјских података о учешћу на тркама исте дистанце у Републици Србији чије преузимање смо описали у поглављу 4.1.3.

4.3 Јавни интерфејс за програмирање апликација

Спецификација јавног интерфејса за програмирање апликација који је имплементиран на серверској страни за приступ од стране клијентске апликације је дата у табели 4.2. За све захтеве из табеле почев од захтева за добијање података о кориснику је потребно проследити приступни токен корисника кроз заглавље захтева на следећи начин:

```
"Authorization": "Bearer " + token
```

Регистрација новог корисника			
HTTP метода	**Путања ресурса**	**Параметри тела захтева**	**Тело одговора**
POST	`api/users/register/`	· *username*: String · *email*: String · *first_name*: String · *last_name*: String · *password*: String · *birth_date*: String	· *user_id*: Integer
Пријављивање корисника			
HTTP метода	**Путања ресурса**	**Параметри тела захтева**	**Тело одговора**
POST	`o/token/`	· *username*: String · *password*: String · *client_id*: String · *client_secret*: String · *grant_type*: String	· *access_token*: String · *expires_in*: long · *token_type*: String · *scope*: String · *refresh_token*: String
Одјављивање корисника			
HTTP метода	**Путања ресурса**	**Параметри тела захтева**	**Тело одговора**
POST	`o/revoke_token/`	· *client_id*: String · *client_secret*: String · *token*: String	/
Добијање података о кориснику			
HTTP метода	**Путања ресурса**	**Параметри захтева**	**Тело одговора**

GET	`api/users/me/`	/	· *strava_athlete_id*: `Integer` · *username*: `String` · *email*: `String` · *first_name*: `String` · *last_name*: `String` · *password*: `String` · *birth_date*: `String`
Мењање корисничких података			
HTTP метода	**Путања ресурса**	**Параметри тела захтева**	**Тело одговора**
PATCH	`api/users/me/`	Мапа типа `Map<String, Object>` са пољима које је потребно поставити на нове вредности. Вредности кључева који се могу користити одговарају вредностима из наредне колоне.	· *strava_athlete_id*: `Integer` · *username*: `String` · *email*: `String` · *first_name*: `String` · *last_name*: `String` · *password*: `String` · *birth_date*: `String`
Брисање корисничког налога			
HTTP метода	**Путања ресурса**	**Параметри захтева**	**Тело одговора**
DELETE	`api/users/me/`	/	/
Предлагање планова тренинга за остваривање одређених резултата			
HTTP метода	**Путања ресурса**	**Параметри захтева**	**Тело одговора**

GET	`api/trainings/`	· *goal_time*: `int` · *race_distance*: `float`	Мапа типа `Map<String, List<List<List< TrainingPlan>>>>` где је `TrainingPlan` речник са пољима следећих типова: · *activity_type*: `String` · *distance*: `float` · *duration*: `int` · *avg_hr_zone*: `int`
Предикција резултата на наредној трци			
HTTP метода	**Путања ресурса**	**Параметри захтева**	**Тело одговора**
GET	`api/result_prediction/`	· *race_distance*: `Sting`	· *race_result*: `Sting`
Месечна статистика тренинга			
HTTP метода	**Путања ресурса**		**Тело одговора**
GET	`api/stats/<int:year>/<int:month>/`		· *training_weeks*: `Map<String, WeekSummary*>`

Табела 4.2: Јавни интерфејс за програмирање апликација

*Опис за `WeekSummary`:

```
WeekSummary =  {
    "activities": List<Activity>,
    "summary": Map<String, ActivitySummary>
};

Activity = {
    "activity_type": String,
```

```
        "distance": float,
        "duration": int,
        "avg_hr_zone": int,
        "start_date": String
    };

    ActivitySummary = {
        "total_duration": int,
        "total_distance": float,
        "avg_hr_zone": int
    };
```

Глава 5

Клијентска страна апликације *Train Wiser*

Функционалности имплементиране на серверској страни, клијентска апликација чини погодним за коришћење од стране корисника кроз графички кориснички интерфејс. У овом поглављу ћемо се детаљније упознати са њеном архитектуром, радним током и представити случајеве употребе.

5.1 Архитектура апликације

За израду клијентске стране апликације, коришћен је програмски језик *Java* уз развојни оквир *Android*. Апликација садржи *Android* активности и њихове распореде за сваки екран који се приказује кориснику, класе за комуникацију са *API*-јем и класе за додатне функционалности.

Комуникација са серверском апликацијом је остварена коришћењем библиотеке *Retrofit*. *Retrofit* [22] је библиотека за типизоване *HTTP* клијенте у *Android* и *Java* апликацијама, коју је развила компанија *Square*. Омогућава лако и ефикасно обављање *REST API* позива, претварајући *HTTP API* у *Java* интерфејс са прецизно дефинисаним типовима података.

Као што је наведено у потпоглављу 4.2.1, клијентска апликација на основу прослеђеног корисничког имена и лозинке корисника, добија приступни токен, токен за обнову и време истицања приступног токена за тог корисника. Да би корисник остао пријављен на апликацији и након што изађе из ње, потребно је сачувати ове податке и онда када апликација није активна. За ово је одабран *SharedPreferences API* [23] који је погодан за чување једноставних података.

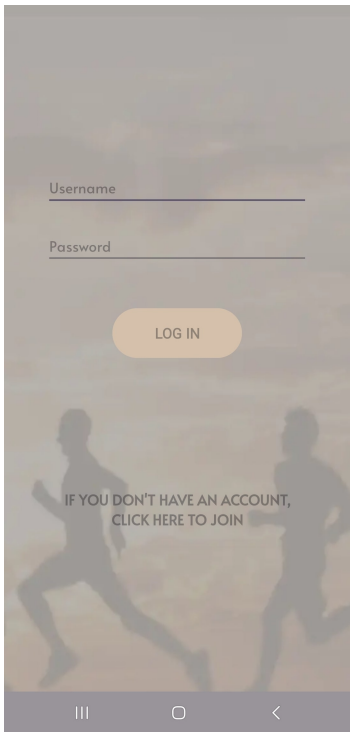
Подаци се складиште у паровима кључ-вредност и чувају у *XML* датотеци у меморији за податке апликације на уређају корисника, а *SharedPreferences API* обезбеђује методе за читање и упис сачуваних података.

Како би подаци о кориснику били доступни у свим тренуцима рада апликације, направљена је `Singleton` класа `ProfileSingleton` за чување параметара корисничког профила који се допремају преко *API*-ја. Приликом покретања апликације, атрибути ове класе се постављају учитавањем најновијег стања са серверске апликације, а освежавају и постављају на иницијалну празну вредност у другим фазама рада апликације описаним у наредном потпоглављу.

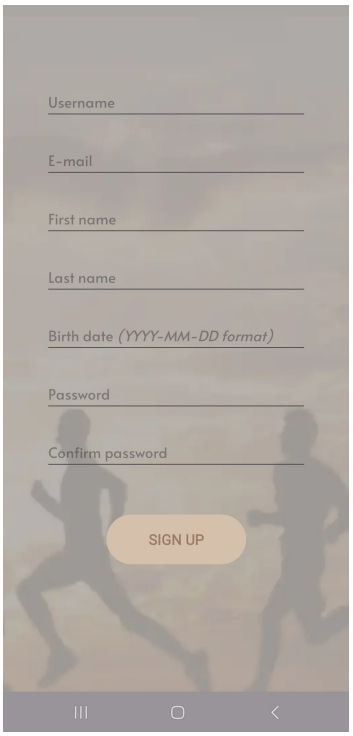
5.2 Радни ток апликације

Ток рада апликације креће од иницијалне *Android* активности која проверава да ли је корисник већ пријављен и, уколико није, преусмерава га на екран за пријављивање на апликацију. На овом екрану се налазе поља за унос података за аутентификацију и опција за преусмерење на екран за регистрацију уколико корисник нема креиран налог. Изглед активности за пријављивање корисника је приказан на слици 5.1. Активност за регистрацију садржи уносе за корисничко име, лозинку, потврду лозинке, *e-mail* адресу и датум рођења, приказана је слици 5.2.

Уколико је корисник већ пријављен, као и након пријављивања, биће преусмерен на екран са главним менијем за избор услуга. Главни мени је приказан на слици 5.3 и садржи групу примарних опција за генерисање плана тренинга, предикцију резултата на наредној трци и приказивање месечне статистке тренинга. Дугме за генерисање плана тренинга отвара нови екран који за одређену дистанцу и резултат, посредством *API* позива ка серверској страни, враћа избор за до највише пет типова вишенедељних планова тренинга које корисник може прегледати кликом на дугме које одговара плану тренинга са резултатом за који је заинтересован (слике 5.4). Снимци екрана за предикцију резултата у случају различитих одговора од стране сервера су приказани на слици 5.15. У оквиру ове *Android* активности, апликација ће кориснику пружити предикцију његовог резултата на одабраној дистанци или информацију да његови резултати, односно резултати за ту дистанцу, нису пронађени међу доступним резултатима у оквиру апликације. Изглед екрана за месечну статистику тренинга је приказан на сликама 5.5.



Слика 5.1: Пријављивање корисника на апликацију *Train Wiser*

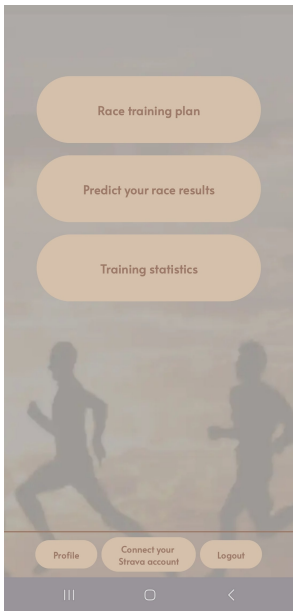


Слика 5.2: Регистрација корисника на апликацију *Train Wiser*

Поред описаних примарних опција, екран са главним менијем нуди могућност за преусмерење на активност са подацима о налогу корисника кликом на дугме *Profile*, дугме за ауторизацију са апликацијом *Strava* и дугме за одјављивање корисника.

Уколико корисник одлучи да повеже свој налог са налогом на апликацији *Strava*, биће преусмерен на веб сајт за одобрење ауторизације ове апликације. Након тога, клијентска апликација позива захтев за добијање података о кориснику како би проверила да ли се у бази уписао идентификатор ка налогу ове апликације, и уколико јесте, чува га у класи `ProfileSingleton` како би се након враћања корисника на главни мени приказало да је налог увезан.

Поред прегледа профила, екран за профил корисника (слика 5.6), има опцију за мењање корисничких података и брисање налога. Уколико корисник промени своје податке и сачува промене кликом на дугме *Save changes*, клијентска апликација шаље захтев серверској за промену података, и у случају успешног одговора, ажурира атрибуте класе `ProfileSingleton` у складу са променама. Подаци из ове класе, као и токени за приступ серверској апли-



Слика 5.3: Изглед главног менија апликације *Train Wiser*

Слика 5.4: Изглед екрана са планом тренинга (хоризонтално померање садржаја табелице са тренингом)

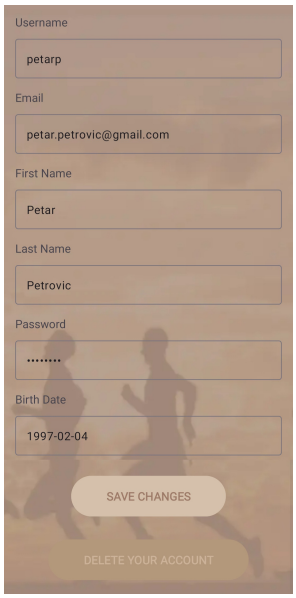
кацији се бришу у случају да корисник одлучи да обрише свој налог или да се одјави са апликације. У случају одјављивање се додатно шаље захтев за опозивање токена ка серверској страни апликације. У оба случаја, корисник ће након слања захтева серверској апликацији бити преусмерен на екран за пријаву на апликацију.

Слика 5.5: Изглед месечне статистике тренинга

5.3 Случајеви употребе

У овом потпоглављу је детаљно изложено како се апликација *Train Wiser* користи кроз токове случајева употребе. Уколико дође до грешке при раду апликације, грешка ће бити приказана као *Toast* порука на дну екрана. Приликом финалног тестирања апликације, интерне грешке на серверској страни се нису јављале. Како би се демонстрирали алтернативни токови за случај интерне грешке на серверу, симулирана је намерна грешка на серверској страни и приказана на слици 5.7. Такође, у току финалног тестирања апликације није долазило до неауторизованих захтева са клијентске стране, симулација грешке је приказана на слици 5.8.[1] Ради веће прегледности, приказ генералних грешака поменутих типова приликом захтева је изостављен на илустрацијама случајева употребе.

[1]Интерне грешке на серверској страни апликације и неаутризовани захтеви од стране клијентске су обрађене на исти начин на нивоу клијентске апликације, на приказаним примерима је илустрован случај са активношћу за генерисање плана тренинга



Слика 5.6: Приказ профила корисника апликације *Train Wiser*

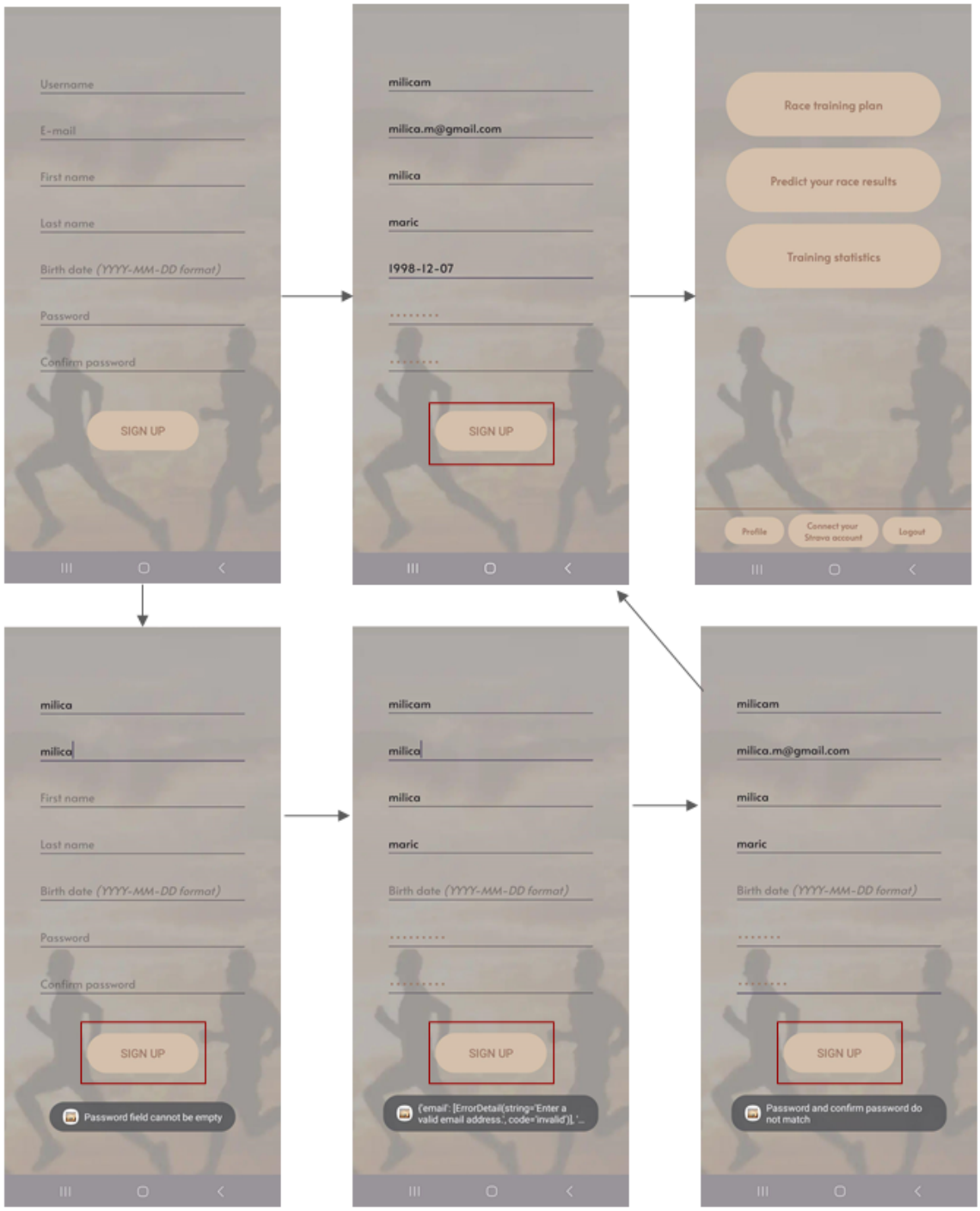
Слика 5.7: Приказ симулиране интерне грешке на серверској страни апликације *Train Wiser*

Слика 5.8: Приказ симулиране грешке приликом неауторизованог захтева са клијентске стране апликације *Train Wiser*

Регистрација корисника

Актери	Корисник, систем (клијентска и серверска апликација)
Предуслови	Корисник нема налог за апликацију
Постуслови	Корисник је регистровао свој налог за апликацију
Главни ток	1. Корисник на екрану за пријављивање на апликацију бира опцију за преусмерење на регистрацију кликом на одговарајућу лабелу 2. Корисник попуњава сва потребна поља и притиска дугме *SIGN UP* 3. Клијентска апликација шаље захтев за регистрацију новог корисника серверској 4. Серверска апликација чува податке о кориснику у бази 5. Клијентска апликација преусмерава корисника на екран за пријављивање
Алтернативни ток 1	Уколико садржај поља за унос није прошао валидацију, корисник се обавештава о грешци и случај употребе се наставља у тачки 2.
Алтернативни ток 2	Уколико је приликом захтева настала грешка која није везана за невалидност унетих параметара, корисник се обавештава о грешци и случај употребе се завршава

Ток случаја употребе је приказан на слици 5.9.

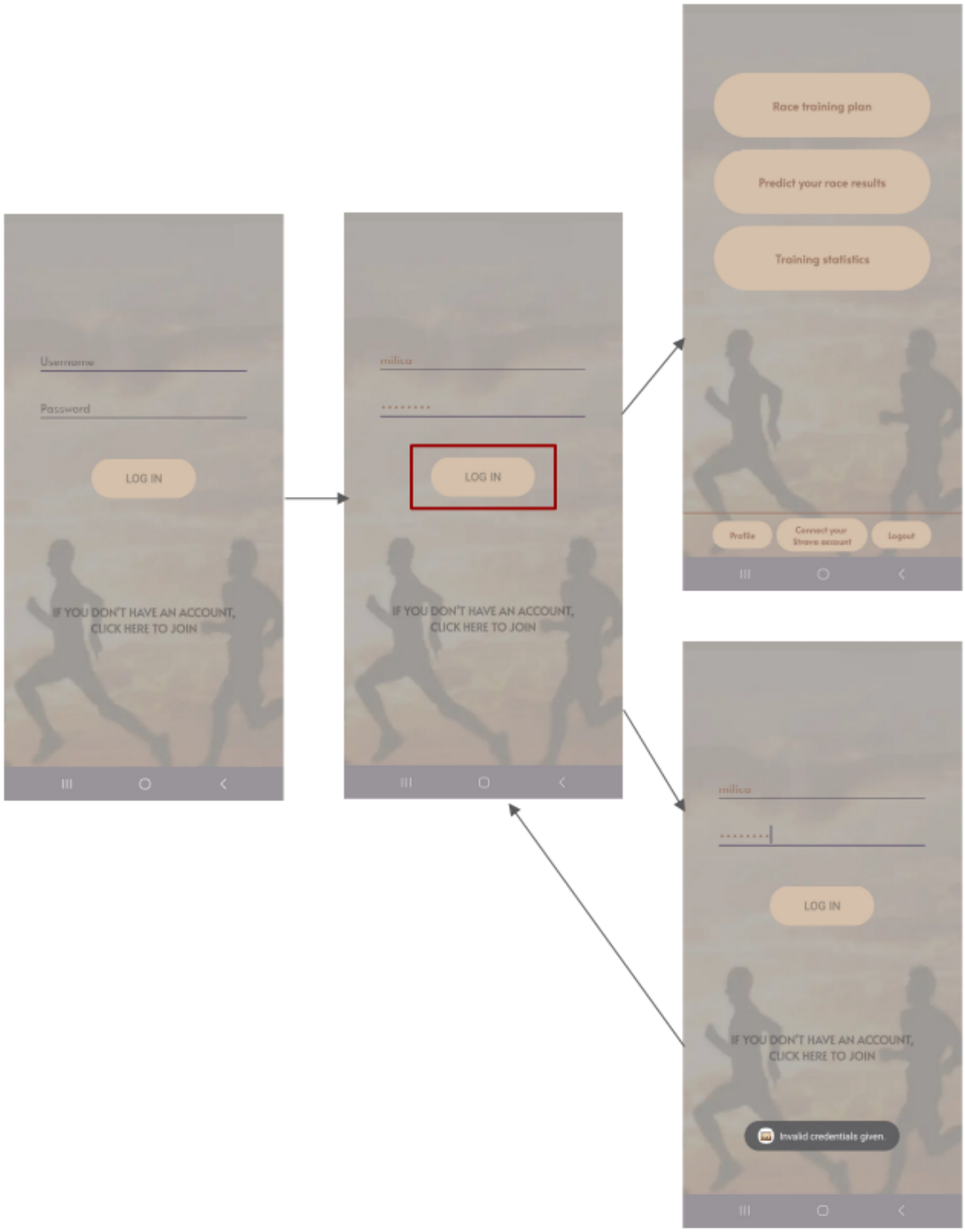


Слика 5.9: Приказ тока случаја употребе за регистрацију корисника

Пријављивање корисника

Актери	Корисник, систем (клијентска и серверска апликација)
Предуслови	Корисник нема сачуване токене за приступ серверској апликацији на уређају, тј. није пријављен на апликацију
Постуслови	Кориснички је пријављен на апликацију
Главни ток	1. Будући да корисник није пријављен, након уласка у апликацију, систем га преусмерава на екран за пријаву 2. Корисник попуњава сва потребна поља и притиска дугме *LOG IN* 3. Клијентска апликација шаље захтев серверској за пријаву корисника 4. Серверска апликација узвраћа токене за пријаву 5. Клијентска апликација чува токене 6. Клијентска апликација преусмерава корисника на екран са главним менијем
Алтернативни ток 1	Уколико је корисник унео невалидне податке за аутентификацију, систем га обавештава о грешци и случај употребе се завршава
Алтернативни ток 2	Уколико је настала грешка приликом захтева, корисник се обавештава о грешци и случај употребе се завршава

Ток случаја употребе је приказан на слици 5.10.



Слика 5.10: Приказ тока случаја употребе за пријављивање корисника

Одјављивање корисника

Актери	Корисник, систем (клијентска и серверска апликација)
Предуслови	Корисник је пријављен на апликацију
Постуслови	Корисник је одјављен са апликације
Главни ток	1. Уколико корисник није на главном менију, корисник се пребацује на главни мени 2. Корисник притиска дугме *Logout* за одјављивање 3. Систем позива захтев за опозивање приступног токена 4. Систем брише токене за приступ серверској апликацији са уређаја корисника 5. Систем преусмерава корисника на екран за пријављивање на апликацију
Алтернативни ток	Уколико се десила грешка приликом захтева за опозивање токена, случај употребе се наставља у тачки 4., а корисник приликом следеће пријаве добија нови приступни токен

Ажурирање корисничких података

Актери	Корисник, систем (клијентска и серверска апликација)
Предуслови	Корисник је пријављен на апликацију
Постуслови	Корисник је ажурирао жељене податке о свом профилу

Главни ток	1. Уколико корисник није на главном менију, корисник се позиционира на главни мени 2. Корисник бира опцију за преглед профила кликом на дугме *Profile* 3. Корисник мења све приказане податке свог профила које жели да промени 4. Корисник притиска дугме *SAVE CHANGES* за чување промена 5. Систем проверава која поља су ажурирана и шаље захтев са промењеним параметрима профила 6. Систем обавештава корисника да су промене профила успешно сачуване 7. Систем преусмерава корисника на главни мени
Алтернативни ток 1	Уколико садржај поља за унос није прошао валидацију, корисник се обавештава о грешци и случај употребе се наставља у тачки 2.
Алтернативни ток 2	Уколико је приликом захтева настала грешка која није везана за невалидност унетих параметара, корисник се обавештава о грешци и случај употребе се завршава

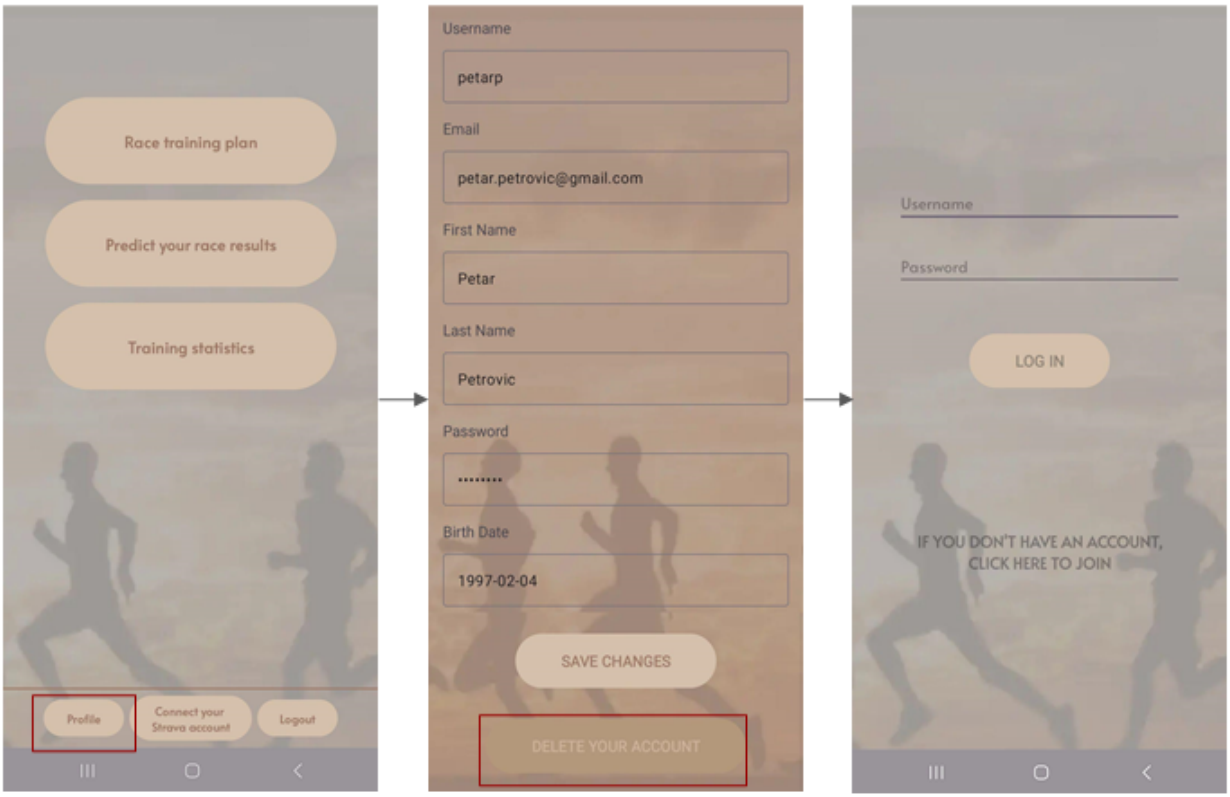
Изглед овог тока случаја употребе је сличан току за регистрацију корисника, будући да су грешке које могу настати при попуњавању поља исте као за регистрацију.

Брисање налога

Актери	Корисник, систем (клијентска и серверска апликација)
Предуслови	Корисник је пријављен на апликацију
Постуслови	Корисник је обрисао налог за апликацију

Главни ток	1. Уколико корисник није на главном менију, корисник се пребацује на главни мени 2. Корисник бира опцију за преглед профила кликом на дугме *Profile* 3. Корисник притиска дугме *DELETE YOUR ACCOUNT* за брисање налога 4. Клијентска апликација позива захтев за брисање налога на серверској апликацији 5. Клијентска апликација брише токене за приступ серверској апликацији са уређаја корисника 6. Систем преусмерава корисника на екран за пријављивање на апликацију
Алтернативни ток	Уколико се десила грешка приликом захтева за брисање налога, систем обавештава корисника о грешци и случај употребе се наставља у тачки 5.

Ток случаја употребе је приказан на слици 5.11.

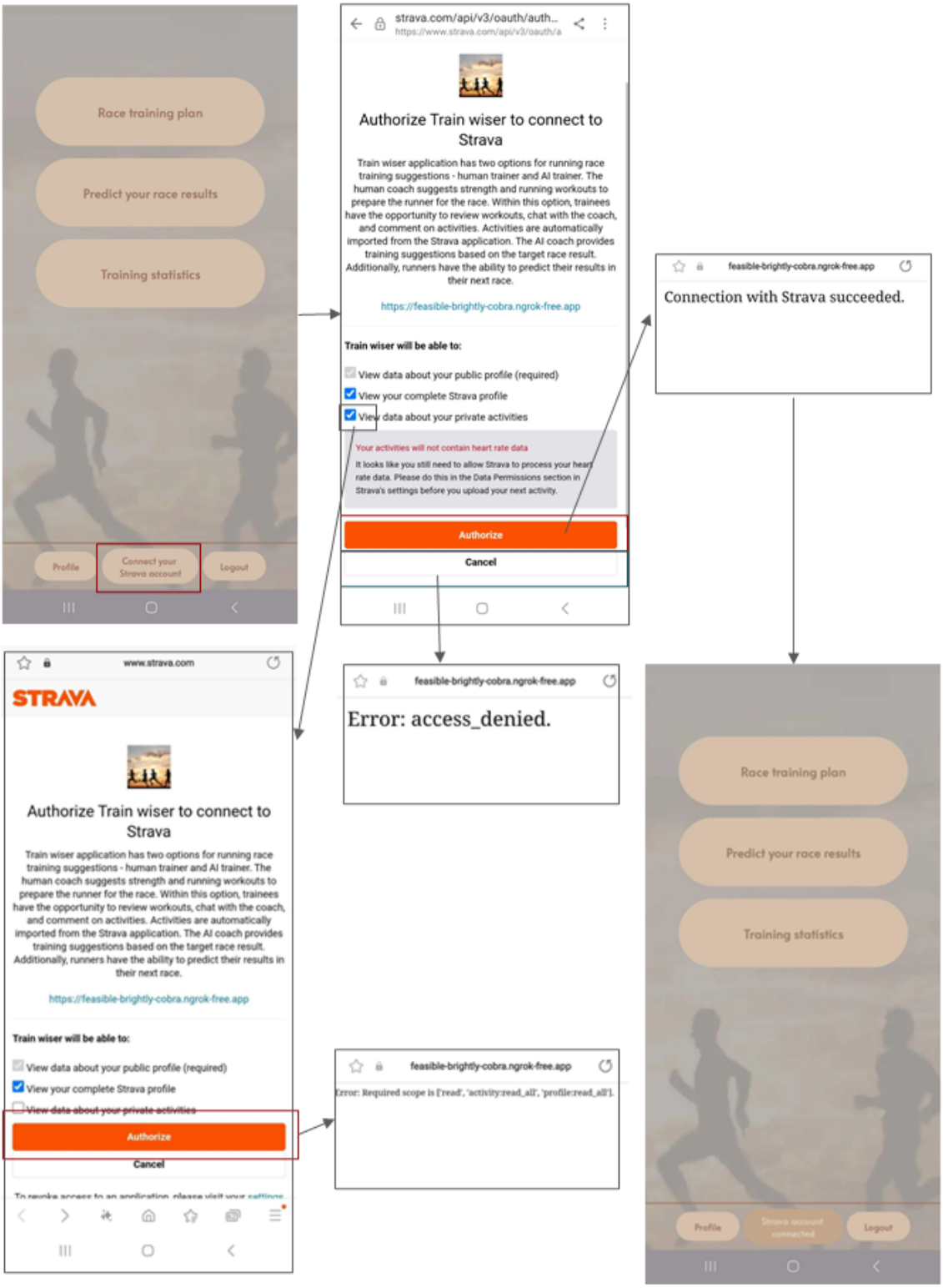


Слика 5.11: Приказ тока случаја употребе за брисање корисничког налога

Ауторизација налога апликације *Train Wiser* са спољним налогом апликације *Strava*

Актери	Корисник, систем (клијентска и серверска апликација)
Предуслови	Корисник је пријављен на апликацију и нема повезан налог са апликацијом *Strava*
Постуслови	Корисник је ауторизовао свој налог са апликације *Train Wiser* на апликацији *Strava*
Главни ток	1. Уколико корисник није на главном менију, корисник се пребацује на главни мени 2. Корисник бира опцију за повезивање са апликацијом *Strava* кликом на дугме *Connect your Strava account* 3. Систем преусмерава корисника на страницу апликације *Strava* (на веб прегледачу) за ауторизацију апликације *Train Wiser* 4. Систем уписује податке о ауторизацији и за највише сат времена покреће преузимање активности корисника 5. Корисник враћа апликацију *Train Wiser* у први план 6. Систем онемогућава клик на дугме за конекцију са апликацијом *Strava*
Алтернативни ток 1	Уколико корисник у тачки 3. није одобрио потребне дозволе апликацији *Train Wiser* или је отказао ауторизацију, добиће одговарајућу поруку са грешком на веб прегледачу и случај употребе се завршава
Алтернативни ток 2	Уколико је приликом захтева настала грешка, корисник се обавештава о грешци и случај употребе се завршава

Ток случаја употребе је приказан на слици 5.12.



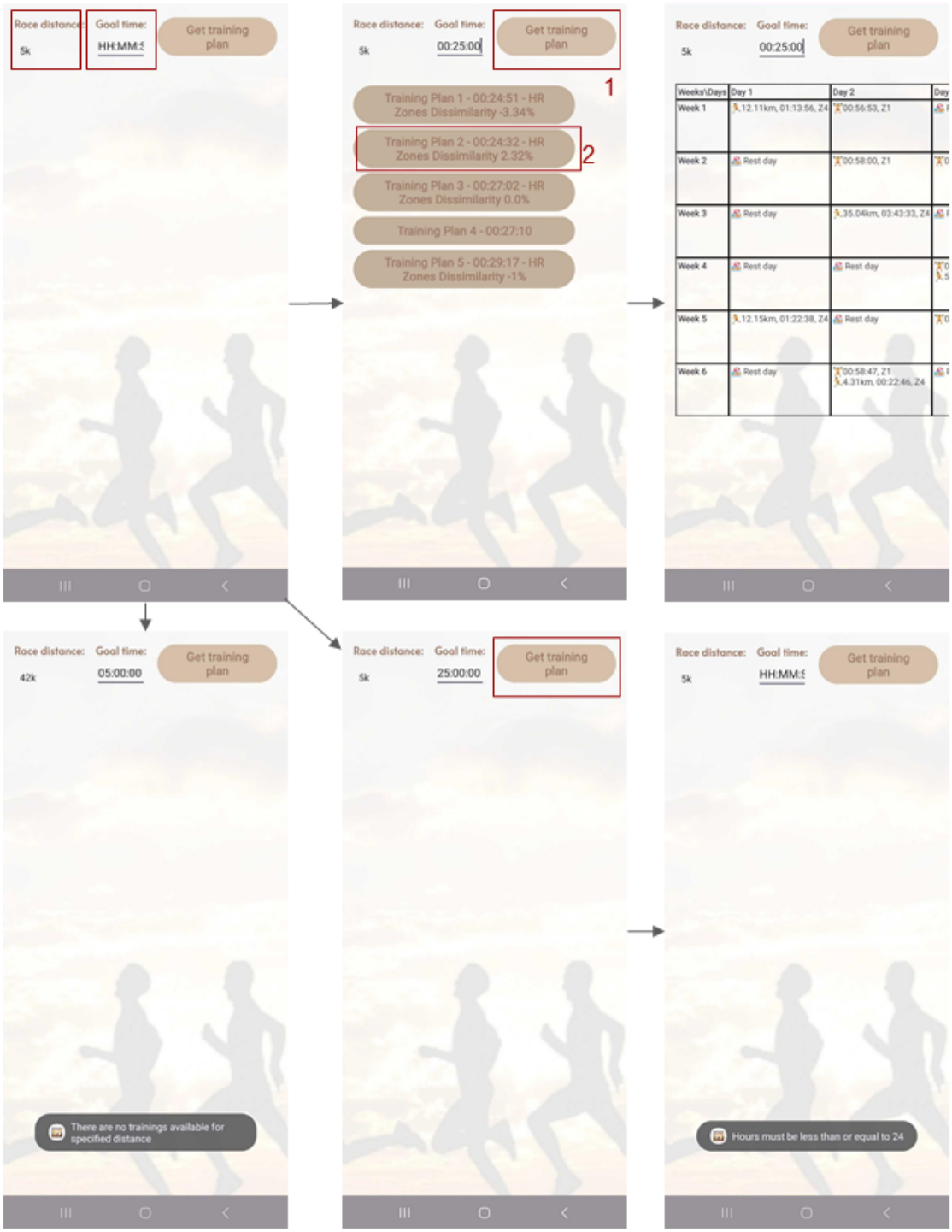
Слика 5.12: Приказ тока случаја употребе за ауторизацију са спољним налогом апликације *Strava*

Предлагање планова тренинга за остваривање одређених резултата

Актери	Корисник, систем (клијентска и серверска апликација)
Предуслови	Корисник је пријављен на апликацију
Постуслови	Корисник је добио до највише пет планова тренинга за одређену дистанцу и циљно време
Главни ток	1. Корисник са главног менија бира опцију за израду плана тренинга кликом на дугме *Race training plan* 2. Корисник бира дистанцу и циљно време 3. Корисник притиска дугме *Get training plan* за генерисање планова тренинга 4. Клијентска апликација шаље захтев серверској за генерисање планова тренинга 5. Клијентска апликација у виду текста на дугмићима приказује кориснику предлоге за највише пет планова тренинга уз које су се остварили резултати најприближнији циљаном уз информације о постигнутом времену са тим типом тренинга и опционалној процентуалној разлици у границама зона срчаног пулса између његових зона и зона корисника на основу чијег тренинга се план генерише 6. Корисник бира једну од понуђених опција кликом на дугме које одговара том плану тренинга 7. Систем приказује план тренинга кориснику
Алтернативни ток 1	Уколико у тачки 3. корисник није унео валидно време, систем обавештава корисника о томе и случај употребе се завршава

Алтернативни ток 2	Уколико у тачки 4. за тражену дистанцу нема доступних тренинга, систем обавештава корисника о томе и случај употребе се завршава
Алтернативни ток 3	Уколико је приликом захтева настала грешка, корисник се обавештава о грешци и случај употребе се завршава

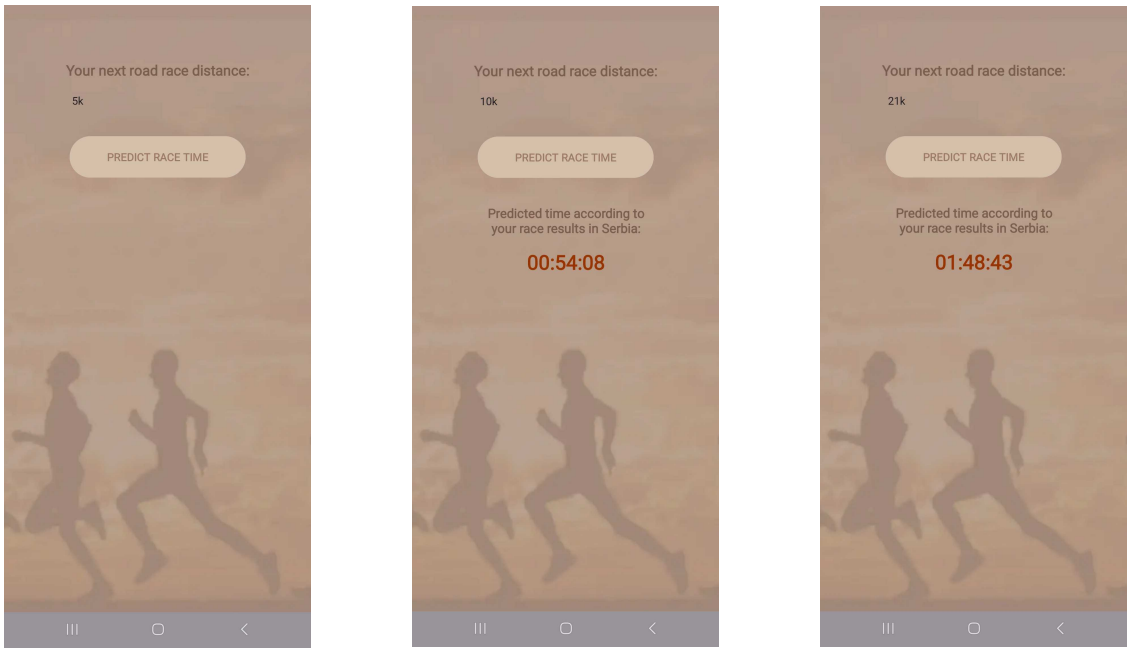
Ток случаја употребе је приказан на слици 5.13, дат је пример грешке у случају да сати имају већу вредност од дозвољене, слично је за минуте и секунде.



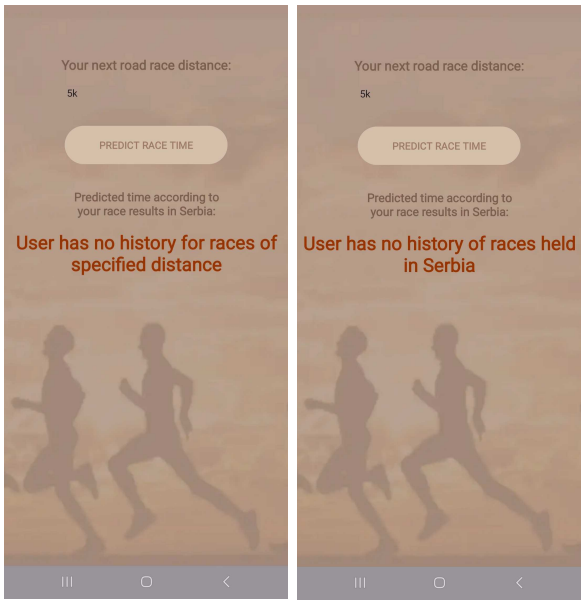
Слика 5.13: Приказ тока случаја употребе за генерисање плана тренинга

Предикција резултата на наредној трци

Актери	Корисник, систем (клијентска и серверска апликација)
Предуслови	Корисник је пријављен на апликацију
Постуслови	Корисник је добио предикцију резултата на следећој трци или одговарајућу поруку
Главни ток	1. Корисник са главног менија бира опцију за приказивање предикције времена на следећој трци кликом на дугме *Predict your race results* 2. Корисник бира дистанцу трке 3. Корисник притиска дугме *PREDICT RACE TIME* 4. Клијентска апликација шаље захтев серверској за добијање предикције времена на следећој трци 5. Систем приказује предикцију времена на следећој трци или поруку да корисник нема доступне информације на тркама у Републици Србији или доступну информацију за одабрану дистанцу
Алтернативни ток	Уколико је приликом захтева настала грешка, корисник се обавештава о грешци и случај употребе се завршава



Слика 5.14: Изглед екрана за предикцију резултата: Иницијални изглед екрана, предикција резултата на трци од 10km и од 21km редом

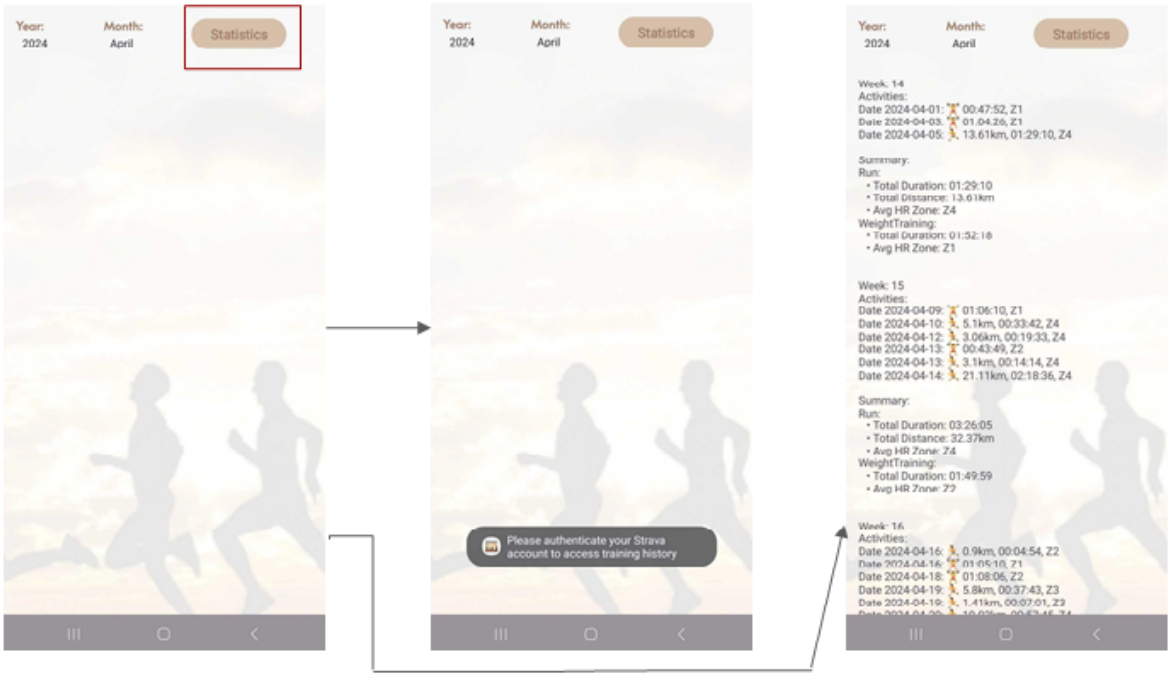


Слика 5.15: Изглед екрана за предикцију резултата када корисник нема доступне податке са резултатима на одабраној дистанци и када га нема у бази са резултатима

Месечна статистика тренинга

Актери	Корисник, систем (клијентска и серверска апликација)
Предуслови	Корисник је пријављен на апликацију
Постуслови	Корисник је добио месечни извештај о тренинзима за одабрану годину и месец
Главни ток	1. Корисник са главног менија бира опцију за приказивање месечне статистике кликом на дугме *Training statistics* 2. Корисник бира годину и месец 3. Корисник притиска дугме *Statistics* за генерисање месечне статистике тренинга 4. Клијентска апликације шаље захтев серверској за добијање месечне статистике 5. Систем приказује месечну статистику кориснику
Алтернативни ток	Уколико је приликом захтева настала грешка, корисник се обавештава о грешци и случај употребе се завршава

Ток случаја употребе је приказан на слици 5.16.



Слика 5.16: Приказ тока случаја употребе за генерисање месечне статистике тренинга

Глава 6

Могућа унапређења апликације *Train Wiser*

Имплементирана апликација садржи све неопходне карактеристике да би била функционална, у наставку овог поглавља су описани могући начини унапређења како би се њена употреба проширила надградњом постојећих и додавањем нових функционалности.

Додавање личног *online* тренера Додавање опције да апликацији могу приступити и лични тренери који би имали могућност да дају план тренинга корисницима и прате њихов напредак кроз приступ тренинзима би допринело проширењу употребе апликације. Било би корисно да апликација олакшава рад тренерима и кроз праћење испуњености задатог тренинга, као и да пружа могућност четовања корисника са тренером и остављања коментара на активностима.

Верификација креираног корисничког налога Верификација налога путем *e-mail* адресе или броја телефона би допринела побољшању безбедности и поузданости корисничких налога. На овај начин би се утицало на спречавање неовлашћеног приступа и унапређење заштите приватних података корисника, што додатно побољшава корисничко искуство и повећава поверење у апликацију.

Увођење верификације путем *e-mail* адресе или броја телефона такође омогућава бржу и лакшу комуникацију са корисницима, било да је реч о постављању нове лозинке у случају да је тренутна заборављена, слању важних

обавештења или обезбеђивању додатних слојева сигурности путем двофакторске аутентификације.

Ограничење броја захтева по кориснику у одређеном временском периоду Да би се спречило преоптерећење серверске апликације захтевима корисника, корисно је увести ограничење количине захтева у одређеном временском периоду. Ово доприноси одржавању стабилности система, што резултира бржим и стабилнијим одговором на захтеве корисника. Ограничење броја захтева спречава злоупотребе као што су *DDoS* напади[1] *(Distributed Denial of Service)* чиме се побољшава безбедност апликације. Такође, овакво ограничење олакшава праћење и анализу корисничког понашања, идентификацију аномалија и благовремено реаговање на потенцијалне проблеме.

Прављење статистике са пласманом за трке Подаци са резултатима трка у Републици Србији би такође могли да се искористе за праћење потребног времена за одговарајући пласман одређене трке кроз године и процену које место би корисник заузео са тренутним најбољим личним резултатом или за резултат који планира да проба да оствари на наредној трци.

Увођење алгоритма машинског учења за предикцију резултата на трци и предлагање тренинга У оквиру процеса израде апликације, направљен је *LSTM* модел који ради над подацима са резултатима трка у Републици Србији. Модел је на улазу добијао претходне три трке корисника и закључивао резултат на четвртој исте дистанце. Међутим, како је просечан број трка по кориснику недовољан за обуку оваквог модела, модел је давао велике грешке. Са оваквим решењем се може пробати у случају доступности већег броја података или применом другачијих алгоритама.

Такође, направљен је и *LSTM* модел за предлагање тренинга за остваривање очекиваног резултата на одређеној дистанци који на основу дистанце и циљног резултата који се добијају на улазу, враћа секвенцу од 60 низова величине 4 који представљају план тренинга.

Поступак формирања скупа података је био извоз базе података у *CSV* фајл помоћу имплементиране *Django* команде `export_to_csv`, који се даље учитавао са библиотеком *Pandas* како би се оформиле секвенце. За сваку трку

[1] *DDoS* напад онемогућава приступ интернет услузи преплављивањем многобројним захтевима, трошећи њен капацитет и онемогућавајући одговор на легитимне захтеве

из базе података која је дистанце 5, 7, 7.7, 10, 21 или 42 километара се узимао скуп тренинга из претходних 60 дана. Тренинзи су представљени секвенцом са типом тренинга, дужином, дистанцом и просечном зоном срчаног пулса, док за дане без тренинга све елементе секвенце садрже нулу.

Овакав модел би могао да се дотренирава са подацима корисника који се пријављују на апликацију и ауторизују је на апликацији *Strava*. Међутим, због тренутне недовољне количине података да би модел давао релевантне резултате, овај модел није убачен у *API* серверске апликације. Уместо тога, користио се приступ са предлагањем тренинга корисника који је остварио најближи резултат циљаном.

Глава 7

Закључак

Овај рад је на примеру апликације за вежбаче обрадио концепте развоја апликација за оперативни систем *Android* и апликација *Python* програмског језика које користе оквире *Django* и *Django REST Framework*. У раду је након теоријског увода у поменуте технологије дат детаљан опис имплементације и употребе креиране апликације. Такође, детаљно су представљене остале технологије и алати коришћени при развоју, попут протокола *OAuth 2.0*, оквира *Scrapy* за екстракцију података, *Cron* послова, концепта мрежних кука итд. Рад садржи сумиране препоруке за наредне фазе унапређења апликације.

Креирана апликација доприноси пољу спортских апликација за трчање кроз детаљну статистику о тренинзима, предлагање различитих планова тренинга припреме за одређену тркачку дистанцу и предикцијом временског резултата на наредној трци. Коришћењем поменутих технологија, омогућен је рад са корисничким подацима, њихова анализа, обрада и интерпретација у складу са потребама корисника. Апликација представља пример начина за рад са корисничким подацима и приказује како они могу допринети бољем раду програма и корисничком искуству, разноврсност корисничких података доприноси бољем раду алгоритама апликације. У контексту рада са корисничким подацима, даља побољшања се огледају како у унапређењу постојећих функционалности новим алгоритмима, тако и у додавању нових приступа при раду са постојећим подацима.

Апликација овог типа може служити као темељ за изградњу апликација које раде са корисничким подацима, апликација које користе ауторизацију спољних налога са других сервиса, као и за развој других типова апликација које су базиране на оперативном систему *Android* или користе оквире *Django*

и *Django REST Framework*.

Библиографија

[1] Meet Android Studio. https://developer.android.com/studio/intro. Приступљено: 10. јул 2024.

[2] Android. https://www.britannica.com/technology/Android-operating-system. Приступљено: 10. јул 2024.

[3] Dawn Griffiths, David Griffiths. *Android programiranje bez oklevanja*. CET Computer Equipment and Trade, Računarski fakultet, 2017.

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

[5] William S. Vincent. *Django for APIs*. Leanpub, 2023.

[6] Django Documentation - Migrations. https://docs.djangoproject.com/en/5.0/topics/migrations/. Приступљено: 2. јул 2024.

[7] Django Documentation - Writing views. https://docs.djangoproject.com/en/5.0/topics/http/views/. Приступљено: 3. јул 2024.

[8] Django Documentation - The Django template language. https://docs.djangoproject.com/en/5.0/ref/templates/language/. Приступљено: 3. јул 2024.

[9] Leonard Richardson, Mike Amundsen. *RESTful Web APIs*. O'Reilly, 2013.

[10] Ивона Милутиновић. Апликација Train Wiser - имплементација. https://github.com/ivonamilutinovic/MasterThesis/train_wiser. Приступљено: 20. август 2024.

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

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

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

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

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

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

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

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

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

[20] Django OAuth Toolkit Documentation. `https://django-oauth-toolkit.readthedocs.io/en/latest/index.html`. Приступљено: 31. јул 2024.

[21] Exponentially Weighted Moving Average (EWMA). `https://corporatefinanceinstitute.com/resources/career-map/sell-side/capital-markets/exponentially-weighted-moving-average-ewma/`. Приступљено: 9. август 2024.

[22] Retrofit. `https://square.github.io/retrofit/`. Приступљено: 1. август 2024.

[23] Save simple data with SharedPreferences. `https://developer.android.com/training/data-storage/shared-preferences`. Приступљено: 2. август 2024.

Биографија аутора

Ивона Милутиновић је рођена 1995. године у Нишу. Прве две године специјализованог одељења за ученике обдарене за математику Гимназије Светозар Марковић завршила је у Нишу, а последње две године Математичке гимназије завршава у Београду 2014. године. Након тога уписује основне студије на Математичком факултету, смер Рачунарство и Информатика, и завршава их 2019. године са просеком 9.68. Мастер студије на Математичком факултету уписује исте године и остварује просек 9.4 на основу положених испита. Током студија је учествовала на такмичењу *Mathackaton*, похађала две летње школе у компанији *RT-RK* и била добитница неколико стипендија међу којима и стипендије Доситеја Фонда за младе таленте Републике Србије. Од 2019. године ради у *IT* индустрији као софтверски инжењер у области аутономне вожње.