

УНИВЕРЗИТЕТ У БЕОГРАДУ
ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ



**СОФТВЕРСКИ СИСТЕМ ЗА УПРАВЉАЊЕ ДОГАЂАЈИМА
У РЕКРЕАТИВНОМ СПОРТУ**

КОРИШЋЕЊЕМ MEAN РАДНОГ ОКВИРА

Дипломски рад основних академских студија

Ментор:

Др Марко Мишић, доцент

Кандидат:

Филип Којић 2018/0285

Београд, август 2023.

САДРЖАЈ

САДРЖАЈ	2
1. УВОД.....	4
2. ПРЕГЛЕД ПОСТОЈЕЋИХ РЕШЕЊА	6
2.1. MEETUP	6
2.2. PLAYO.....	7
2.3. TIMSTER	8
2.4. УПОРЕДНА АНАЛИЗА КАРАКТЕРИСТИКА ПОСТОЈЕЋИХ РЕШЕЊА.....	9
2.4.1. Опсег спортова	9
2.4.2. Цена.....	9
2.4.3. Доступност.....	10
2.4.4. Тржиште.....	10
3. ПРЕГЛЕД КОРИШЋЕНИХ ТЕХНОЛОГИЈА	11
3.1. NODE.JS	11
3.1.1. Libuv	12
3.1.2. npm (node package manager).....	13
3.1.3. package.json	14
3.1.4. JSON (JavaScript Object Notation).....	14
3.2. EXPRESS.JS	15
3.3. MONGODB	17
3.4. ANGULAR	18
3.4.1. Архитектура	19
3.4.2. Модули	20
3.4.3. Компоненте.....	20
3.4.4. Сервиси	20
3.4.5. Dependency Injection.....	20
3.4.6. Рутирање.....	20
3.4.7. Изрази	20
3.4.8. Директиве.....	21
3.4.9. Observable и Promise.....	21
3.5. ПРЕГЛЕД ОСТАЛИХ ТЕХНОЛОГИЈА.....	21
3.5.1. TypeScript	22
3.5.2. Bootstrap.....	22
3.5.3. Google Cloud Platform.....	22
4. ФУНКЦИОНАЛНА СПЕЦИФИКАЦИЈА	24
4.1. ОПИС ПРОБЛЕМА	24
4.2. КАТЕГОРИЈЕ КОРИСНИКА	24
4.2.1. Учесник	25
4.2.2. Организатор.....	25
4.2.3. Администратор.....	25
4.3. ФУНКЦИОНАЛНОСТИ СИСТЕМА	25
4.3.1. Регистрација.....	26
4.3.2. Пријава на систем	26
4.3.3. Пријава помоћу Google налога.....	26
4.3.4. Одјављивање са платформе	26
4.3.5. Приказ профила.....	27
4.3.6. Ажурирање профила	27

4.3.7.	Преглед актуелних догађаја	27
4.3.8.	Филтрирање актуелних догађаја.....	27
4.3.9.	Детаљни приказ догађаја.....	27
4.3.10.	Пријава/отказивање пријаве за догађај.....	28
4.3.11.	Отказивање догађаја.....	28
4.3.12.	Коментарисање догађаја.....	28
4.3.13.	Преглед и "праћење"/"отпраћивање" организатора.....	29
4.3.14.	Додавање новог догађаја.....	29
4.3.15.	Евиденција плаћања	29
4.3.16.	Преглед свих учесника система.....	30
4.3.17.	Преглед свих организатора система.....	30
4.3.18.	Преглед профила учесника и организатора.....	30
4.3.19.	Брисање корисника из система	30
4.3.20.	Аутоматско ажурирање статуса догађаја.....	30
5.	ОПИС АРХИТЕКТУРЕ И РЕАЛИЗАЦИЈЕ СИСТЕМА.....	32
5.1.	АРХИТЕКТУРА СИСТЕМА	32
5.2.	МОДЕЛ БАЗЕ ПОДАТАКА	33
5.2.1.	users.....	33
5.2.2.	events	34
5.2.3.	sports.....	35
5.3.	ДЕТАЉИ ИМПЛЕМЕНТАЦИЈЕ И ПРИСТУП РЕШАВАЊУ ПРОБЛЕМА	36
5.3.1.	Иницијализација сервера	36
5.3.2.	Аутентификација и ауторизација корисника	38
5.3.3.	Обрада пријаве помоћу Google налога.....	43
5.3.4.	Аутоматизам система и рад са мејловима	45
5.3.5.	Рад са мапама	47
5.3.6.	Рад са сликама	50
6.	ОПИС РАДА СИСТЕМА.....	52
6.1.	ОПИС РАДА УЧЕСНИКА	53
6.2.	ОПИС РАДА ОРГАНИЗАТОРА	58
6.3.	ОПИС РАДА АДМИНИСТРАТОРА.....	62
7.	ЗАКЉУЧАК.....	64
	ЛИТЕРАТУРА.....	66
	СПИСАК СКРАЋЕНИЦА	68
	СПИСАК СЛИКА.....	69
	СПИСАК ТАБЕЛА.....	70
	СПИСАК ДЕЛОВА ПРОГРАМСКОГ КОДА	71

1. Увод

Рекреативни спорт игра важну улогу у животима многих људи, омогућавајући им да се релаксирају, одржавају кондицију и на квалитетан начин искористе своје слободно време. У добу модерних технологија, потреба за лакшим и ефикаснијим начином организације и учешћа у различитим спортским активностима постаје све израженија. Савремени алати и апликације могу значајно да унапреде и олакшају ову организацију, али треба их прилагодити конкретним потребама учесника.

Мотивацију за овај рад аутор је пронашао у жељи да се олакша и унапреди искуство свих учесника у рекреативном спорту. Данашњи начин организације догађаја често се ослања на друштвене мреже и апликације за чет као што су *Messenger*, *Viber*, *WhatsApp* и слично. Међутим, аутор овог рада приметио је да ови алати често нису довољно прилагођени специфичним потребама организације спортских догађаја. Апликација *GameTime* има за циљ да значајно олакша и побољша све аспекте организације догађаја, пружајући корисницима интуитиван интерфејс, персонализоване функционалности и ефикасно решење за потребу организације спортских догађаја.

Ова апликација решава низ проблема. Пре свега, олакшава организаторима процес креирања догађаја. Кроз интуитиван интерфејс, организатори могу брзо и једноставно унети све детаље догађаја, укључујући датум, време, место одржавања, спорт и цену термина. Осим тога, омогућава им и да евидентирају све плаћања у систему за претходно одржане догађаје. Затим, учесницима омогућава лаку пријаву за жељене догађаје. Учесници могу прегледати актуелне догађаје, изабрати оне који одговарају њиховим интересима и пријавити се за учешће у њима. На овај начин, учесници лако налазе друге ентузијасте са истим спортским афинитетима. На крају, администратори имају јасан преглед свих корисника система и претходно одржаних догађаја.

С обзиром на присуство три различита типа корисника - учесника, организатора и администратора, апликација се фокусира на пружање персонализованог искуства сваком кориснику, унапређујући комуникацију и олакшавајући сваки корак процеса планирања и управљања спортским догађајима. Овакав софтверски систем доноси већу ефикасност, тачност и побољшано искуство свим релевантним актерима.

Овај рад се састоји из седам поглавља. У наставку ће бити детаљно објашњени проблеми настали приликом реализације овог софтверског система, као и начини на које су они решени. У другом поглављу, дат је преглед постојећих решења, као и опис сваког од њих. Треће поглавље доноси преглед и опис коришћених технологија у изради апликације *GameTime*. У четвртом поглављу ће бити дата функционална спецификација апликације *GameTime* са описом категорија корисника и прегледом функционалности. Пето поглавље представља опис архитектуре и реализације система. Шесто поглавље се бави презентовањем рада система за сваку категорију корисника апликације. Седмо и финално поглавље представља закључак и смернице за потенцијални даљи развој израђене апликације, као и њено унапређивање.

2. ПРЕГЛЕД ПОСТОЈЕЋИХ РЕШЕЊА

Ово поглавље намењено је анализи и прегледу апликационих решења која се баве организацијом и координацијом спортских догађаја. Представимо следеће три апликације:

- *Meetup*
- *Playo*
- *Timster*

2.1. *Meetup*

Meetup [1] је друштвена медијска платформа осмишљена да олакша организовање и одржавање како уживо, тако и виртуелних активности, окупљања и догађаја за људе и заједнице сличних интереса, хобија и професија. Иако се *Meetup* не користи искључиво за организацију спортских догађаја, због своје популарности, лакоће коришћења и напредних функционалности које нуди, може се идеално употребити и у те сврхе.

Од свог оснивања 2002. године, *Meetup* је омогућио корисницима да прате догађаје који их занимају и упознају људе сличних афинитета, било онлајн или уживо. Платформа броји преко 60 милиона чланова и корисницима је омогућено да лако сретну људе с којима ће се слагати, креирати групе и учествовати у различитим догађајима.

За категорију организатора коришћење апликације се наплаћује и постоје *Standard* и *Pro* пакети који се наплаћују на месечном, шестомесечном или годишњем нивоу. *Standard* пакет нуди опције као што су организација до три групе с неограниченим бројем чланова, заказивање догађаја, комуникација са члановима и постављање чланарине и накнада за догађаје како би се покрили трошкови. *Pro* пакет нуди платформу за управљање мрежом група, приступ мејловима учесника, напредне аналитике мреже, прилагођене обрасце за присуство, приступ API (*Application Programming Interface*) и премијум подршку од стране стручњака за заједницу. Цене су различите у зависности од земље. За Србију, месечна цена за *Standard* пакет је 14.49 USD (*United States dollar*), док је шестомесечна цена 9.99 USD, а годишња 6.89 USD. *Pro* пакет кошта 35 USD месечно, док је шестомесечна цена 30 USD. [1]

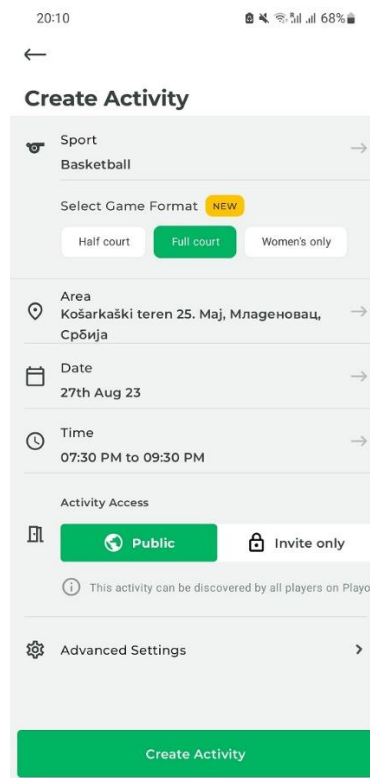


Слика 2.1.1. Почетни екран *Meetup* апликације

2.2. *Playo*

Playo [2], апликација основана 2015. године, представља платформу за организацију спортских догађаја и окупљање љубитеља разних врста спорта, омогућавајући им да пронађу играче, терене, тренере и савете унутар једне платформе. Апликација подржава организацију догађаја за преко 50 различитих спортова. Иако је глобално доступна, апликација има највећи успех на индијском тржишту.

Корисници могу пронаћи играче и спортске групе у свом окружењу на основу вештина, спортских интересовања и временских преференција. Могуће је креирати спортске догађаје, изазвати пријатеље или тимове на утакмице и пратити разне статистике. Такође, корисници могу резервисати спортске терене, сазнати више о спортској опреми и добијати обавештења о локалним спортским догађајима. Осим тога, постоји могућност плаћања унапред, као и преглед садржаја, цена и рецензија терена. Постоји и секција за блогове и чланке, као и могућност постављања питања унутар заједнице. Сваки корисник има свој профил где може пратити свој напредак, зарађивати бодове и упоређивати се с другима. Коришћење апликације је бесплатно за све типове корисника. [2]



Слика 2.2.1. Креирање новог догађаја у апликацији *Playo*

2.3. *Timster*

Timster [3] је иновативна апликација осмишљена како би олакшала организацију утакмица фудбала и малог фудбала. Ова андроид апликација је плод рада српске фирме *Solaris Development & Design*, која је за реализацију овог пројекта освојила друго место на такмичењу *Smart City Challenge Serbia* 2016. године. Кроз ову апликацију, љубитељи фудбала могу лако саставити тимове, лоцирати терене и пронаћи додатне играче из своје близине, ако се суоче с недостатком играча за предстојећу утакмицу.

Једна од главних предности *Timster* апликације је могућност филтрирања играча и утакмица унутар радијуса од 25 километара, што омогућава корисницима да лако пронађу и комуницирају са другим ентузијастима фудбала у својој близини. Ова апликација није само алат за организацију, већ је истовремено и друштвена мрежа намењена искључиво љубитељима рекреативног фудбала. Кроз функционалности као што су *news feed* и *Timster* чет, корисници могу делити своје фудбалске тренутке, упознавати се са другима и ширити своју страст према "најважнијој споредној ствари на свету". Коришћење апликације је бесплатно за све типове корисника. [3]



Слика 2.3.1. Преглед локације у апликацији *Timster*

2.4. Упоредна анализа карактеристика постојећих решења

Приликом анализе апликација за организацију спортских догађаја и окупљања, уочава се неколико кључних разлика међу апликацијама *Meetup*, *Playo*, *Timster* и апликацијом *GameTime*, урађеном у склопу израде овог дипломског рада.

2.4.1. Опсег спортова

Док *Playo* подржава организацију догађаја за преко 50 различитих спортова, *Timster* је специфичан и фокусиран искључиво на фудбал и мали фудбал. *GameTime* нуди избор десетак најпопуларнијих спортова, док *Meetup*, иако се може користити за разне активности укључујући и спорт, није специјализован за то.

2.4.2. Цена

Playo и *Timster* су бесплатни за кориснике, док *Meetup* има тарифне моделе за организаторе. *GameTime* тренутно пружа бесплатну употребу за све типове корисника, а постоји могућност да се то у наредним верзијама промени.

2.4.3. Доступност

Timster је доступан само као андроид апликација, док су *Meetup* и *Playo* доступни за андроид и iOS (*iPhone Operating System*), као и у виду веб апликације. *GameTime* је тренутно доступна само као веб апликација.

2.4.4. Тржиште

Playo је посебно популаран на индијском тржишту, *Timster* је производ српске фирме и има локални фокус. *Meetup* је глобално препозната и широко коришћена платформа која повезује људе са сличним интересима и омогућава организацију различитих догађаја у многим земљама широм света. У својој тренутној верзији, *GameTime* је посебно осмишљен да одговара потребама и жељама љубитеља спорта у Србији.

У табели 3.3.1. дат је преглед постојећих решења проблема организације спортских догађаја заједно са њиховим карактеристикама.

Табела 3.3.1. Упоредна анализа карактеристика постојећих решења

Карактеристика/Апликација	<i>Meetup</i>	<i>Playo</i>	<i>Timster</i>	<i>GameTime</i>
Опсег спортова	Није намењена посебно спорту	Преко 50 различитих спортова	Фудбал/мали фудбал	Најпопуларнији спортови
Цена	Наплаћује се	Бесплатно	Бесплатно	Бесплатно
Доступност	Андроид iOS Веб апликација	Андроид iOS Веб апликација	Андроид	Веб апликација
Тржиште	Глобално	Индија	Србија	Србија

3. ПРЕГЛЕД КОРИШЋЕНИХ ТЕХНОЛОГИЈА

У савременом дигиталном окружењу, избор правих технологија често представља одлучујући фактор у успешности и ефикасности развоја софтверских решења. Поглавље које је пред нама пружа детаљан увид у технолошки оквир пројекта, разматрајући сваку компоненту која је коришћена за његову реализацију. Овај преглед не само да ће послужити као техничка основа за разумевање архитектуре и функционалности система, већ ће такође истаћи разлоге због којих су одређене технологије изабране и које предности оне доносе пројекту. Прво ће бити описане компоненте MEAN [4] оквира (*MongoDB, Express, Angular, Node*), а потом додатне технологије, сервиси и библиотеке које су коришћене приликом израде апликације *GameTime*.

3.1. Node.js

Node.js [5] је софтверска платформа отвореног кода која омогућава извршавање *JavaScript* кода ван окружења веб прегледача. Замишљен је да пружи једноставан начин креирања "лакших" и скалабилних веб апликација са многим улазно/излазним операцијама. *Node.js* представља парадигму "*JavaScript* свуда", која обједињује развој веб апликација око једног програмског језика, уместо различитих језика за скрипте на страни сервера и клијента. *Node.js* користи једну нит за обраду, али се ослања на механизме који ће бити описани у наставку како би обрадио захтеве ефикасно. Захваљујући неблокирајућој, догађајима вођеној архитектури, *Node.js* ефикасно управља великим бројем симултаних захтева, што га чини идеалним за развој *real-time* веб апликација. *Node.js* користи *V8 JavaScript engine* за извршавање скрипти, конвертујући их у машински код који рачунар може да разуме и изврши.

Node.js ради на основу два концепта [6]:

- 1) Асинхроност
- 2) Неблокирајуће улазно/излазне операције

Асинхроност подразумева извршавање функције повратног позива (*callback* функције). Чим добијемо одговор од другог сервера или базе података, извршава се *callback* функција. Главна нит не ради са захтевом директно. Уместо тога, она користи библиотеку *Libuv*, написану у језику *C*, која омогућава креирање "базена" нити (*Thread pool*) и коришћење помоћних радних нити за обраду улазно/излазних операција као што су читање датотеке,

упити над базом података, приступ удаљеном сервису и слично, те тако остаје доступна за друге радње.

Неблокирајуће операције се односе на рад с вишеструким захтевима без блокирања главне нити за појединачни захтев. *Node.js* се не препоручује за задатке који захтевају интензивне процесорске операције, попут сложених математичких израчунавања или обраде видео материјала. Разлог за то је што је основна архитектура *Node.js* базирана на једној главној нити, која би могла постати загушена таквим операцијама, чиме би се смањила ефикасност у обради улазно/излазних захтева.

3.1.1. *Libuv*

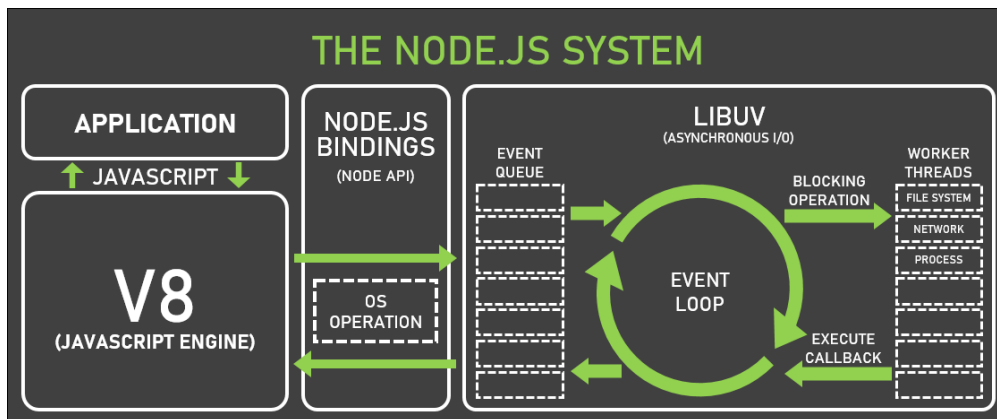
Libuv је библиотека написана у језику C која имплементира изузетно важне компоненте *Node.js* [7]:

- Петља догађаја (*Event loop*)
- Ред догађаја (*Event queue*)
- "Базен" нити (*Thread pool*)

Event loop је основни механизам који омогућава *Node.js* да извршава асинхрони, неблокирајући код упркос томе што користи само једну нит. Његова примарна улога је да непрекидно проверава да ли има функција или задатака које треба извршити. Када апликација изда захтев, на пример за читање фајла, *Event loop* наставља да обавља друге задатке док се захтев не заврши. Чим се захтев заврши и постане доступан повратни позив (*callback*), *Event loop* га извршава. Ова архитектура омогућава *Node.js* да ефикасно рукује са вишеструким захтевима.

Event queue је механизам који користи *Node.js* како би пратио све догађаје који чекају да буду обрађени. Када се неки задатак заврши, као што је читање фајла или обрада HTTP (*Hypertext Transfer Protocol*) захтева, повратни позив се ставља у ред догађаја. *Event loop* непрекидно проверава овај ред и, чим се ослободи, преузима и извршава *callback* из њега.

Thread pool је колекција радних нити које чекају да обаве задатке. У контексту *Node.js*, *Thread pool* се користи за обављање блокирајућих операција, тако да главна нит може остати неблокирајућа и брза. Када је потребна блокирајућа операција, радна нит из "базена" преузима тај задатак. Једном када је задатак завршен, радна нит враћа резултат главној нити, која потом може извршити одговарајући *callback*.



Слика 3.1.1.1 *Node.js* систем [7]

3.1.2. *npm* (*node package manager*)

npm [8] представља основни систем за управљање пакетима у *JavaScript* окружењу. С обзиром на експанзивни раст и развој *JavaScript* екосистема, потреба за ефикасним управљањем библиотекама и модулима постала је неопходност. Управо ту на сцену ступа *npm*, алат за командну линију који инсталира, ажурира или деинсталира пакете за *Node.js* апликације, омогућавајући програмерима да лако управљају зависностима (*dependencies*) својих пројеката. Сви регистровани пакети налазе се у *Node Package Registry*. [9]

Коришћење *npm* је једноставно и интуитивно. Пре свега, потребно је инсталирати *Node.js*, јер *npm* долази у пакету с њим. Команде помоћу којих се управља *Node.js* модулима углавном имају облик:

npm operacija -opcije imeModula

Неке од операција које се често користе су [10]:

- *search* - Проналази пакете модула у репозиторијуму
- *install* - Инсталира пакете помоћу датотеке *package.json* из репозиторијума или из локала
- *remove* - Уклања пакет
- *view* - Приказује детаље о модулу

Сви модули који се инсталирају помоћу *npm* се инсталирају у директоријуму *node_modules*.

3.1.3. *package.json*

package.json је кључна датотека у сваком *Node.js* пројекту. Дефинише све зависности пројекта, што значи све библиотеке и модуле од којих пројекат зависи како би правилно функционисао. Када се користи команда `npm install`, *Node.js* претражује *package.json* како би сазнао које зависности треба преузети и инсталирати. Такође, у овој датотеци се могу дефинисати скрипте, конфигурације за различите алате и многе друге поставке специфичне за пројекат. Неке од директива које садржи су: [10]

- *name* - Означава јединствени назив пакета
- *version* - Означава верзију модула
- *author* - Име аутора пројекта
- *scripts* - Одређује параметре помоћу којих се извршавају апликације конзоле
- *dependencies* - Означава модуле и верзије од којих зависи модул
- *engines* - Означава верзију модула *node* помоћу којег пакет функционише

3.1.4. *JSON (JavaScript Object Notation)*

JSON [10] је формат података који је постао стандард за размену података на интернету. Иако је његово име повезано с језиком *JavaScript*, JSON је језички независан стандард, што значи да се може користити с много различитих програмских језика, не само са *JavaScript*.

Основна синтакса JSON формата је слична литералима објеката у језику *JavaScript*, с тим што има неколико кључних разлика. У JSON формату, кључеви морају бити између двоструких наводника, а валидне вредности могу бити низови, бројеви, објекти, стрингови, *true*, *false* и *null*. Сложени типови података, попут објеката и низова, могу бити угнеждени да би се креирале сложеније структуре података.

JSON је постао изузетно популаран због своје једноставности и читљивости, што га чини идеалним за размену података између клијента и сервера у веб апликацијама. Већина савремених API користи JSON као свој примарни формат за пренос података. JSON се такође често користи за конфигурационе фајлове и похрану података у апликацијама које не захтевају комплексне системе за управљање базама података.

```
{
  "ime": "Marko",
  "godine": 20,
  "interesovanja": ["matematika", "programiranje"],
  "adresa": {
    "ulica": "Bulevar Kralja Aleksandra 73",
    "grad": "Beograd"
  }
}
```

Део кода 3.1.4.1. Пример JSON формата

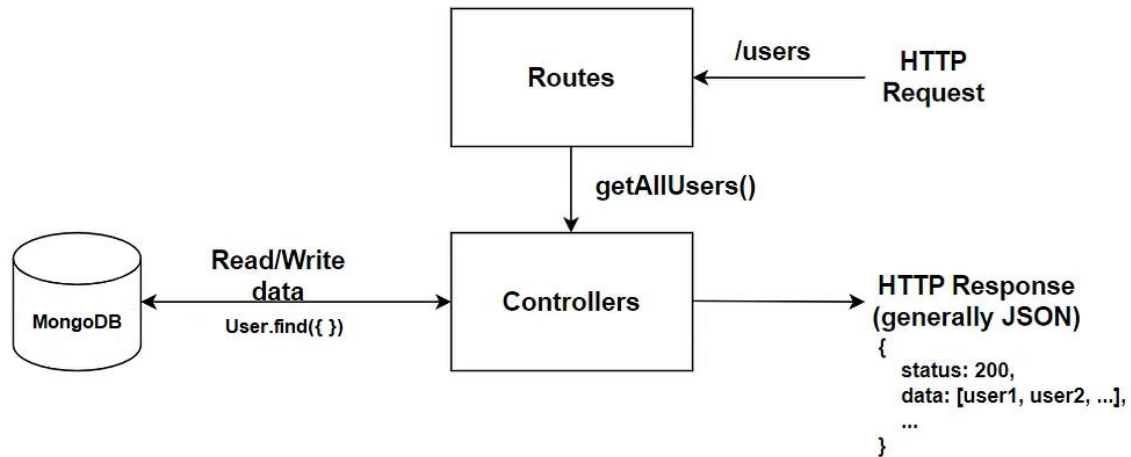
Међу предностима коришћења JSON формата су његова једноставност, лакоћа читања и широка подршка међу програмским језицима. Такође, пренос података овог формата преко мреже је брз и ефикасан. С друге стране, JSON нема подршку за коментаре и може представљати само податке, а не логику. То значи да, док је одличан за размену и похрану података, JSON није погодан за све примене где су потребне напредније функционалности, као што су операције над подацима или програмски код.

3.2. *Express.js*

Express.js [11], или краће *Express*, је минималистички и флексибилни радни оквир за *Node.js*, дизајниран за изградњу веб апликација и API. Као део екосистема *Node.js*, *Express* олакшава развој веб сервера и омогућава бржу и организовану изградњу апликација. Док *Node.js* пружа основну инфраструктуру за рад са мрежом и улазно/излазним операцијама, *Express* додаје слој који поједностављује многе аспекте израде веб апликација, као што су рутирање, обрада захтева и руковање грешкама. У комбинацији с другим алатима из екосистема *Node.js*, *Express* пружа моћну платформу за модерни веб развој.

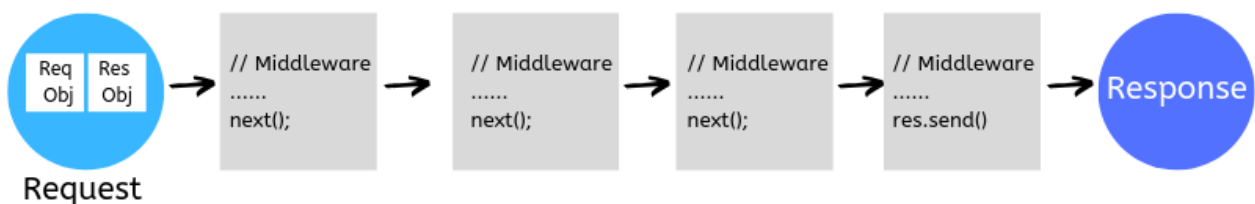
Рутирање [12] у *Express* радном оквиру представља начин на који се веб апликација или API одазива на клијентске захтеве на одређеним URL (*Uniform Resource Locator*) путањама користећи специфичне HTTP методе. Омогућавајући програмерима да лако дефинишу како сервер треба да реагује на GET, POST, PUT, DELETE и друге HTTP захтеве, *Express* пружа флексибилно и интуитивно рјешење за мапирање путања на одређене функције обраде, које су обично написане у посебним контролерима. *Express* подржава динамичко рутирање путем параметара руте, омогућавајући програмерима да "хватају" варијабле из URL, као што је ID (*идентификатор*) ресурса или корисничко име. Све ово је упаковано у елегантан и модуларан систем, где се руте могу груписати и организовати коришћењем *Express Router* објекта чиме се постиже "чист" и одржив код чак и у сложеним веб апликацијама.

`router.get("/users", getAllUsers)`



Слика 3.2.1. Процес рутирања и обраде HTTP захтева [12]

Middleware [13] у *Express* оквиру представља кључну компоненту у обради захтева и одговора унутар веб апликације. Ове функције служе као посредници између улазних захтева клијената и крајњих одговора сервера, омогућавајући програмерима да на адекватан начин дефинишу процес обраде. Уз способност да се активирају у специфичном редоследу, *middleware* функције омогућавају детаљну контролу над током апликације, било да је у питању логовање, аутентификација корисника, обрада података или руковање грешкама. Свака *middleware* функција има приступ објектима HTTP захтева (*req*) и одговора (*res*), као и *next* функцији, која сигнализира да се апликација треба померити на следећи *middleware* у низу. Оваква архитектура омогућава изузетну модуларност, где се различити аспекти обраде могу раздвојити у независне јединице функционалности.



Слика 3.2.2. Ланчана обрада захтева помоћу *middleware* функција [13]

3.3. *MongoDB*

MongoDB [14] је документно оријентисана база података која припада групи *NoSQL* система за управљање базама података. За разлику од традиционалних база података које користе табеле са предефинисаним шемама за складиштење података, *MongoDB* користи документе са флексибилном структуром, који су груписани у колекције. Иако ови документи изгледају као JSON на први поглед, они су заправо сачувани као BSON [15] (*Binary JSON*), бинарни формат који омогућава ефикаснију серијализацију и десеријализацију података него стандардни JSON. BSON такође подржава додатне типове података који нису присутни у стандардном JSON формату.

Документи у *MongoDB* могу имати један или више парова кључ-вредност. Документи не морају имати исту структуру, што значи да различити документи унутар исте колекције могу имати различите кључ-вредност парове. Оваква флексибилност даје додатну слободу програмерима приликом дефинисања или промене структуре података.

Једна од најмоћнијих карактеристика *MongoDB* јесте његова скалабилност. Скалабилност се односи на способност система да се прилагоди повећању оптерећења, било повећањем перформанси појединачне инстанце (вертикална скалабилност) или додавањем више инстанци (хоризонтална скалабилност). *MongoDB* је дизајниран да буде посебно скалабилан и то на хоризонтални начин, што га чини атрактивним за велике апликације које могу имати јако велике количине података и/или висок промет. Неколико кључних аспеката скалабилности у *MongoDB* су [16]:

- 1) **Хоризонтална скалабилност (*sharding*):** *MongoDB* подржава *sharding*, што значи да је могуће дистрибуирати податке преко више сервера. Како расте количина података или се повећава број захтева, може се додавати више чворова у кластер. Сваки чвор ће чувати део података, чиме се омогућава расподела оптерећења и ефикасност у извршавању упита.
- 2) **Репликациони сетови:** Поред хоризонталне скалабилности, *MongoDB* подржава репликацију, што значи да је могуће имати више копија података. Ово побољшава доступност и отпорност на грешке. Ако један сервер падне, други сервер (или сервери) који има копију података може преузети његову улогу.
- 3) **Аутоматско балансирање оптерећења:** Како количина података и захтева расте, *MongoDB* аутоматски премешта податке између различитих чворова у кластеру како

би се осигурало балансирано оптерећење. То омогућава систему да одржава високе перформансе чак и под условима великог оптерећења.

- 4) **Оптимизација за велике сетове података:** *MongoDB* је оптимизован за рад са великим сетовима података, користећи технике као што су индексирање за брзе упите и компресија података за ефикасно складиштење.

У суштини, скалабилност *MongoDB* омогућава организацијама да расту и прилагођавају се променљивим условима, било да се ради о повећању количине података или броја корисника. Поред скалабилности, *MongoDB* нуди различите функционалности као што су: индексирање, агрегација података, подршка за трансакције на нивоу документа, и многе друге напредне могућности. Интеграција с различитим програмским језицима је олакшана кроз разноврсне драјвере, укључујући и оне за *JavaScript*, што га чини природним избором за развој веб апликација, нарочито када се користи у комбинацији са *Node.js* и *Express*. Посебно вредна помена је *Mongoose*, популарна библиотека за *Node.js* која омогућава елегантан начин за повезивање и рад са *MongoDB* базом. [9]

3.4. *Angular*

Angular [17] је један од водећих радних оквира за израду динамичних и скалабилних једностраничних апликација (*Single Page Application*) које се извршавају унутар веб прегледача. Иако се често меша с претходном верзијом, *Angular.js*, савремени *Angular* је значајно еволуирао и прешао на снажнији и флексибилнији језик *TypeScript*. *TypeScript*, надоградња на класични *JavaScript*, пружа статичко типизирање, интерфејсе и декораторе, што га чини идеалним за развој сложених апликација.

При раду са *Angular* оквиrom, кључно је разумети његову модуларну архитектуру. Модули групишу логичке делове апликације, као што су компоненте, сервиси, директиве и други модули, омогућавајући организован и одржив развој. Овакав модуларни приступ олакшава поновно коришћење кода, тестирање и одржавање.

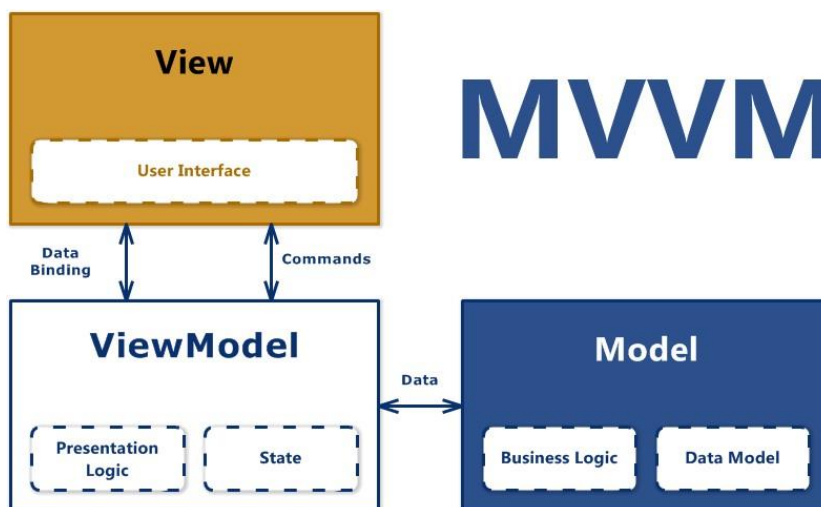
Међутим, оно што *Angular* значајно издваја од многих других радних оквира јесте његова способност двосмерне везе података (*Two-way data binding*). Ово омогућава аутоматско ажурирање корисничког интерфејса када се модел података промени, и обрнуто, без потребе за ручним интервенцијама или употребом додатних AJAX (*Asynchronous JavaScript and XML*) позива. Таква синхронизација између модела и корисничког интерфејса чини развој бржим и

интуитивнијим, док су апликације високо интерактивне и респонзивне. У наставку ће бити представљена архитектура *Angular* оквира, а затим још неколико битних карактеристика. [18]

3.4.1. Архитектура

Angular прати MVVM (*Model-View-ViewModel*) пројектни образац. Да бисмо боље разумели, размотрићемо посебно сваку компоненту овог обрасца.

- **Model:** У *Angular* оквиру, модел представља основне податке и пословну логику апликације. Ово обично укључује сервисе, који комуницирају с екстерним изворима података попут API или база података, као и класе или интерфејсе који дефинишу структуру тих података. Сервиси су често део *Angular Dependency Injection* система, што олакшава њихово тестирање и поновно коришћење.
- **View:** *Angular* користи компонентни приступ, где свака компонента има придружени HTML (*HyperText Markup Language*) шаблон који дефинише кориснички интерфејс. Ови шаблони, заједно с пратећим стиливима, чине *View* у MVVM обрасцу.
- **ViewModel:** *TypeScript* код који прати сваку *Angular* компоненту делује као *ViewModel*. Овај део компоненте управља логиком приказа, обрадом података који долазе из модела, те корисничким интеракцијама. Због двосмерне повезаности података коју *Angular* нуди, *ViewModel* аутоматски ажурира *View* када се подаци промене, и обрнуто.



Слика 3.4.1.1. MVVM архитектура [18]

3.4.2. Модули

Модули представљају основу за организацију *Angular* апликација. Сврха модула је да групише функционалности које су логички повезане. Све компоненте, сервиси и директиве који се односе на одређену функционалност могу бити груписани унутар једног модула. Осим тога, модули омогућавају "лењо" учитавање, што значи да се делови апликације учитавају само када су потребни, побољшавајући перформансе. *AppModule* је главни и стартни модул сваке Ангулар апликације, који иницијализује и покреће апликацију.

3.4.3. Компоненте

Компоненте су основне градивне јединице *Angular* апликација. Свака компонента се састоји од HTML шаблона, *TypeScript* класе која дефинише логику, и *CSS(Cascading Style Sheets)* стила.

3.4.4. Сервиси

Док компоненте управљају корисничким интерфејсом, сервиси се користе за пружање одређених функционалности које нису директно везане за приказ. То укључује комуникацију са сервером, управљање локалним подацима, или било коју другу логику која треба да буде доступна широм апликације.

3.4.5. Dependency Injection

Dependency Injection је пројектни узорак који *Angular* користи за "убризгавање" зависности (као што су сервиси) директно у компоненте, чиме се олакшава тестирање и повећава модуларност.

3.4.6. Рутирање

Рутирање омогућава дефинисање путања за различите делове апликације. Ово је есенцијално за креирање једностраничних апликација где се различите компоненте динамички учитавају на истој страници без освежавања целе странице.

3.4.7. Изрази

Изрази омогућавају повезивање података са HTML шаблоном. Они се користе унутар двоструких витичастих заграда за приказивање променљивих или резултата функција директно унутар шаблона.

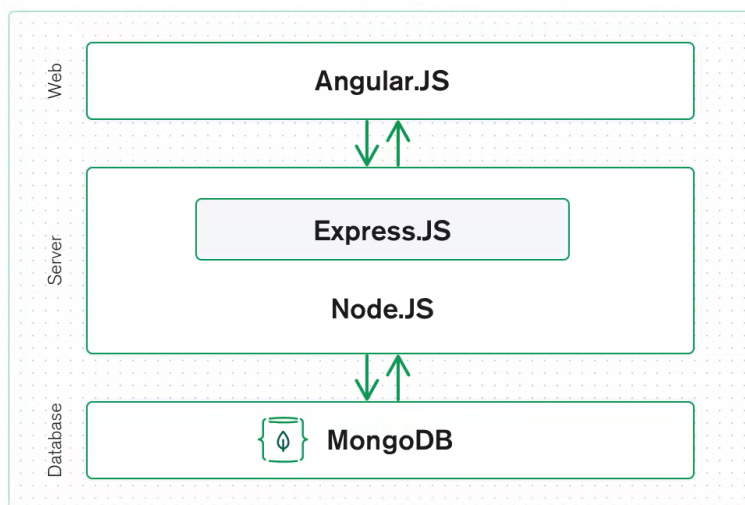
3.4.8. Директиве

Директиве су механизми који проширују функционалности HTML. Док су компоненте директиве са шаблонима, постоје и друге врсте директива које манипулишу структуром DOM(*Document Object Model*) или понашањем елемената.

3.4.9. *Observable* и *Promise*

Observable су централни део *Angular* програмирања који омогућава рад са асинхроним операцијама и догађајима на елегантан начин. *Observable* емитује ток података или догађаја које претплатници (позиваоци *subscribe* методе) могу "хватати" и реаговати на њих. За разлику од *Promise*, који се извршавају једном и враћају једну вредност, *Observable* може емитовати више вредности током времена.

Promise (обећања) представљају механизам за управљање асинхроним операцијама у *JavaScript* језику. Овај механизам се често користи за поједностављивање "*callback hell*" проблема (много угнеждених повратних позива у коду) и пружају једноставнији начин за рад са асинхроним операцијама. У комбинацији са *async/await* синтаксом, обећања омогућавају писање асинхроног кода који изгледа готово синхроно. [11]



Слика 3.4.9.1. Шема MEAN радног оквира [4]

3.5. Преглед осталих технологија

Следи кратак преглед преосталих технологија које су коришћене приликом развоја апликације *GameTime*.

3.5.1. *TypeScript*

Поред клијентске стране написане у *Angular* оквиру, *TypeScript* је коришћен и у писању кода на серверској страни. Међутим, с обзиром да већина серверских окружења (као што је и *Node.js*) извршава чисти *JavaScript*, *TypeScript* код мора бити прво преведен у *JavaScript* пре него што се изврши. То се постиже помоћу *TypeScript* компајлера(*tsc*). Овај корак је неопходан како би се осигурало да код може бити правилно извршен на серверској страни. Одлука о коришћењу *TypeScript* језика на оба краја омогућила је конзистентност у развоју и бржи развојни циклус захваљујући предностима које овај језик нуди.

3.5.2. *Bootstrap*

Bootstrap је оквир за развој клијентске стране апликације који помаже програмерима и дизајнерима у креирању визуелно привлачних и респонзивних веб апликација. У пројекту је коришћен *Bootstrap* како би се постигао модеран и професионалан изглед апликације без потребе за писањем сложених CSS правила. Осим што нуди широк спектар предефинисаних стилова, компоненти и додатака, *Bootstrap* такође обезбеђује систем који олакшава прилагођавање дизајна различитим величинама екрана и уређајима.

3.5.3. *Google Cloud Platform*

Google Cloud Platform нуди низ моћних алата и сервиса који омогућавају развој апликација са обogaћеним функционалностима. У овом пројекту су присутне четири кључне услуге које платформа пружа везано за мапе и локацијске сервисе.

- 1) ***Maps JavaScript API***: Овај API нам је омогућио да интегришемо интерактивне мапе унутар апликације, пружајући корисницима визуелни приказ локација догађаја.
- 2) ***Places API***: Коришћење овог API омогућило нам је приступ огромној бази података о местима широм света, као и функционалност *autocomplete* при претрази локација.
- 3) ***Geocoder API***: Овај сервис је био кључан када је било потребно конвертовати адресе у географске координате. На пример, када корисник унесе одређену адресу, *Geocoder API* је коришћен да би се та адреса претворила у тачне географске координате, које су потом приказане на мапи.
- 4) ***Geolocation API***: Кроз овај API, апликација је била способна да одреди тачну локацију корисника у реалном времену. Ово је омогућило филтрирање на основу близине догађаја у односу на корисникову тренутну локацију.

Током развоја апликације *GameTime*, коришћене су још неке библиотеке инсталиране уз помоћ прт. Оне ће бити укључене у текст приликом прегледа функционалности апликације.

4. ФУНКЦИОНАЛНА СПЕЦИФИКАЦИЈА

Свака софтверска апликација настаје као одговор на одређени проблем или потребу. Функционална спецификација је кључна компонента у процесу развоја софтвера која пружа детаљан приказ проблема који апликација треба да реши, као и захтеве и очекивања корисника у вези са функционалностима апликације. У овом поглављу, прво ће бити представљен опис проблема који је мотивисао развој апликације, истражујући главне изазове и потребе које апликација треба да задовољи. Након тога, прећи ћемо на категорисање корисника, дефинишући различите врсте корисника и њихове јединствене потребе. На крају, фокусираћемо се на функционалности које апликација треба да пружи свакој категорији корисника. Овим ће се осигурати да су сви кључни захтеви корисника покривени и да апликација пружа свеобухватно решење за постављени проблем.

4.1. Опис проблема

Савремени свет рекреације и спорта све више се ослања на технологију како би омогућио што бољу комуникацију и организацију догађаја. Међутим, тренутне платформе, као што су популарне друштвене мреже и апликације за комуникацију, као што су *Messenger*, *Viber*, *WhatsApp* и друге, често не успевају да задовоље специфичне потребе оних који организују спортске догађаје. Основни проблем је у томе што ове платформе нису прилагођене за брзо и прецизно планирање, евиденцију и управљање оваквим догађајима.

GameTime апликација је настала као одговор на ове изазове. Ова апликација представља иновативно решење које се директно фокусира на поједностављење процеса креирања и управљања спортским догађајима. Организатори, путем интуитивног интерфејса, могу лако унети све неопходне информације о догађају, док учесници имају могућност да прегледају, изаберу и пријаве се на жељене догађаје. Ова дигитална платформа такође олакшава администрацији да ефикасно управља свим детаљима, омогућавајући свеобухватни преглед организованих догађаја и њихових учесника.

4.2. Категорије корисника

С обзиром да се сваки корисник разликује по својим потребама, апликација је дизајнирана тако да сваком пружи прилагођено искуство. Ово не само да побољшава ефикасност планирања и управљања, већ и омогућава корисницима да се лако повежу с

другима који деле сличне спортске интересе. У апликацији постоје три категорије корисника: учесник, организатор, администратор.

4.2.1. Учесник

Учесник у *GameTime* апликацији је категорија корисника жељна укључивања у различите спортске догађаје, истраживања нових прилика за рекреацију и повезивања са сличним ентузијастима. Кроз апликацију, учеснику је пружена прилика да се на једноставан начин информише, пријављује на догађаје, комуницира са организаторима и другим учесницима, и прилагоди своје корисничко искуство својим интересовањима. Ова централна фигура је кључна за динамику спортских догађаја, и кроз *GameTime* апликацију добија све алате потребне за потпуно уживање у свету рекреације и спорта.

4.2.2. Организатор

Организатор у оквиру *GameTime* апликације игра кључну улогу као мост између спортских ентузијаста и успешно организованих догађаја. Он је одговоран за креирање, управљање и надгледање спортских активности, пружајући учесницима прилике за рекреацију и забаву. Опремљен специфичним алатима, организатор има могућност да планира догађаје, комуницира са учесницима, евидентира плаћања и прати динамику својих активности. Кроз *GameTime* апликацију, организатор не само да поставља темеље за сваку спортску активност, већ такође има приступ свеобухватним функционалностима које му омогућавају да ефикасно управља својим догађајима.

4.2.3. Администратор

Администратор унутар *GameTime* апликације има надзорну улогу над свим аспектима апликације, од корисничких профила до финансијских трансакција. Док учесници и организатори уживају у динамичном свету спортских догађаја, администратор има задатак да прати све корисничке интеракције, правилно евидентира трансакције и осигура да сви корисници имају позитивно искуство. Кроз *GameTime*, администратор има приступ алатима који омогућавају ефикасно управљање корисничким профилима, догађајима и финансијским аспектима, чинећи га кључном фигуром у одржавању равнотеже и квалитета на платформи.

4.3. Функционалности система

Да бисмо боље разумели како *GameTime* задовољава потребе својих корисника, размотрићемо опште функционалности које апликација нуди. Након тога, уронићемо дубље у

сваку категорију корисника како бисмо истражили специфичне функционалности прилагођене њиховим јединственим улогама и потребама унутар система.

4.3.1. Регистрација

Уколико корисник нема свој налог, може га креирати уношењем личних података. Ти подаци ће бити уписани у базу података на основу чега ће касније корисник моћи да приступа систему. Подаци потребни за регистрацију су:

- име
- презиме
- корисничко име(јединствено у систему)
- лозинка
- телефон
- мејл(јединствено у систему)
- тип корисника(учесник или организатор)
- опис(само за организатора)
- профилна слика(опционо)

4.3.2. Пријава на систем

Уколико је корисник претходно регистрован може се пријавити на платформу уносом корисничког имена и лозинке. Унесени подаци се морају поклопити са онима у бази.

4.3.3. Пријава помоћу Google налога

Корисник може да се пријави на систем помоћу свог налога на *Google* платформи. Уколико корисник није претходно регистрован, систем ће генерисати све податке за њега осим типа за који се региструје, броја телефона и описа(за организатора). Након пријаве корисник је преусмерен на нову страницу где довршава креирање налога допуном ових података.

4.3.4. Одјављивање са платформе

Одјава омогућава корисницима да безбедно затворе своју сесију, штитећи њихов налог од неовлашћеног приступа. Након што се одјави, корисник је преусмерен на почетну страницу(страницу за пријаву).

4.3.5. Приказ профила

Сваки корисник система, независно од типа, има могућност прегледа свог профила. При прегледу свог профила, корисник може да види све своје личне информације унесене при регистрацији, изузев лозинке, у табеларном облику. Поред основних информација, организатор у свом профилу има и табеларни приказ догађаја које је претходно организовао. Сваки од тих догађаја има линк који води до детаљног приказа, где могу бити прегледане све спецификације и информације везане за тај догађај. Слично организатору, и учесник у свом профилу има табеларни приказ догађаја на којима је учествовао. Такође, сваки од тих догађаја има линк који води до детаљног приказа тог догађаја.

4.3.6. Ажурирање профила

Корисници имају могућност да ажурирају одређене информације у свом профилу: мејл, телефон, опис(за организатора) и профилну слику.

4.3.7. Преглед актуелних догађаја

Учесници имају могућност да прегледају све актуелне догађаје који су доступни у систему. За сваки догађај ће бити приказане основне информације: врста спорта, место одржавања и датум и време догађаја. Организатори могу прегледати само догађаје које су они сами организовали.

4.3.8. Филтрирање актуелних догађаја

Учесници могу користити функционалност филтрирања да би лакше и брже пронашли догађаје који су им географски блиски. Систем омогућава филтрирање актуелних догађаја на основу удаљености од тренутне локације корисника. Корисник може да изабере једну од понуђених опција: у кругу од 5 km, 10 km, 15 km или 20 km од своје тренутне локације.

4.3.9. Детаљни приказ догађаја

Када корисник кликне на одређени догађај са листе актуелних или претходно одржаних догађаја, отвара се страна која детаљно представља информације о том догађају. Овде се кориснику приказују детаљи догађаја:

- Врста спорта
- Место одржавања
- Датум и време одржавања

- Цена термина
- Цена по учеснику(за претходно одржане догађаје)
- Потребан број учесника за догађај(за актуелне догађаје)
- Број пријављених учесника
- Рок за пријаву на догађај(за актуелне догађаје)

Испод основних информација, приказана је локација догађаја на мапи, како би учесници могли лакше да се оријентишу и планирају свој долазак. Додатно, испод мапе је доступна листа коментара које су оставили учесници или организатор о том догађају. За организаторе и администраторе, постоји додатна функционалност на овој страни. Они могу да виде табеларни приказ свих пријављених учесника за догађај. Уз сваког учесника, постоји линк који води на детаљни приказ профила тог учесника, чиме се омогућава боље упознавање са учесницима који ће учествовати(или су учествовали) на догађају.

4.3.10. Пријава/отказивање пријаве за догађај

Учесници система имају могућност пријаве на било који актуелни догађај, под условом да максимални број потребних учесника није већ попуњен. У случају да учесник реши да не жели више да учествује у неком догађају на који је претходно био пријављен, има опцију да откаже своје учешће.

4.3.11. Отказивање догађаја

Организатор има могућност да откаже било који актуелни догађај који је организовао. У случају отказивања догађаја, систем аутоматски шаље обавештење свим пријављеним учесницима на догађају. Обавештење се шаље путем електронске поште и садржи основне информације о догађају који је отказан: име и презиме организатора, спорт, локација, датум и време одржавања.

4.3.12. Коментарисање догађаја

На страници са детаљним приказом догађаја, постоји део намењен за интеракцију корисника у виду коментара. Учесници и организатори имају могућност да оставе свој коментар у вези са конкретним догађајем. Сви коментари који су постављени биће приказани хронолошким редоследом, са информацијама о аутору коментара и временском ознаком када је коментар постављен.

4.3.13. Преглед и "праћење"/"отпраћивање" организатора

Учесник има могућност прегледа свих организатора у систему. Преглед укључује име и презиме, профилну слику и кратак опис профила организатора. Такође, може да "запрати" организатора. Ако учини тако, аутоматски ће примати обавештења на свој мејл у вези са будућим догађајима које организатор креира. Ако учесник већ "прати" организатора, имаће опцију да га "отпрати". "Отпраћивањем" организатора, учесник више неће добијати обавештења о догађајима које организатор прави.

4.3.14. Додавање новог догађаја

Организатор има опцију креирања новог догађаја у систему. Процес додавања новог догађаја обухвата следеће кораке:

- **Избор врсте спорта:** Организатор бира која врста спорта ће се играти или практиковати на догађају. Доступни су следећи спортови: фудбал, кошарка, одбојка, рукомет, стони тенис, тенис, ватерполо, амерички фудбал.
- **Унос минималног и максималног броја учесника:** Организатор дефинише колико је најмање и највише учесника потребно да би догађај био успешно организован.
- **Одређивање локације:** Организатор уноси тачну локацију где ће догађај бити одржан.
- **Унос цене термина:** Организатор даје информацију о цени термина за нови догађај. Цена ће касније бити подељена са бројем пријављених учесника како би сваки учесник знао колико ће га коштати учешће.
- **Одређивање датума и времена одржавања:** Организатор уноси датум и време када ће догађај бити одржан.

По успешном уносу свих потребних информација и креирању догађаја, систем аутоматски шаље обавештење свим "пратиоцима" организатора путем електронске поште. У обавештењу се корисници информишу о новом догађају са основним информацијама о њему: име и презиме организатора, спорт, локација, датум и време одржавања, као и рок за пријаву.

4.3.15. Евиденција плаћања

Организатор и администратор имају могућност да воде евиденцију о плаћању учесника за догађаје који су већ одржани. Ова функционалност је организована у облику табеле где су наведени сви пријављени учесници догађаја. Поред колона са корисничким именом, именом,

презименом и линком који води на приказ профила учесника, у табели постоји и колона "Платио" са *checkbox* пољима које организатор/администратор може да штиклира/одштиклира и да на тај начин означи да ли је учесник платио потребан износ за учешће у догађају. Евиденцију је потребно сачувати кликом на дугме "Сачувај" испод табеле са учесницима.

4.3.16. Преглед свих учесника система

Администратор система има приступ интерфејсу који омогућава преглед свих регистрованих учесника. Овај интерфејс је представљен у форми табеле која садржи следеће колоне: корисничко име, име, презиме, статус(активан/неактиван), као и линк који води на приказ профила учесника.

4.3.17. Преглед свих организатора система

Слично као за учеснике, администратор има могућност да прегледа информације о свим регистрованим организаторима у систему у виду табеле са истим колонама као код учесника.

4.3.18. Преглед профила учесника и организатора

Поред тога што сваки регистровани корисник може да види свој лични профил, организатор и администратор имају додатна права приступа. Организатор може да види профиле учесника који су се пријавили или учествовали на његовом догађају, односно све информације које су унели приликом регистрације, осим лозинке. Администратор има детаљан преглед како учесника, тако и организатора, заједно са свим њиковим претходним учешћима/организацијама догађаја у табеларној форми, поред основних информација унетих при регистрацији.

4.3.19. Брисање корисника из система

Администратор има специјалну привилегију да брише кориснике из система. Ово брисање, уместо да потпуно уклони корисника из базе података (што може бити проблематично из разних разлога, укључујући подршку и историјске податке), поставља корисников статус на "неактиван". Када је кориснику постављен овај статус, он више не може да приступи систему и користи његове функционалности.

4.3.20. Аутоматско ажурирање статуса догађаја

Ово је пример чисто серверске функционалности, без клијентске интеракције са интерфејсом апликације. Систем на дневном нивоу ажурира статус сваког актуелног догађаја базирано на датуму и времену за пријаву на тај догађај(у односу на тренутно време) и броју

пријављених учесника. Ако је пријављено мање од потребног броја учесника, систем аутоматски мења статус догађаја на "отказан" и шаље мејл свим пријављеним учесницима са обавештењем о отказивању са основним информацијама догађаја. Ако је број пријављених учесника у оквиру потребног броја, систем аутоматски поставља статус догађаја на "завршен" и шаље подсетник свим учесницима о предстојећем догађају тог дана.

Систем је дизајниран да буде изразито интерактиван, одговарајући на сваку акцију и захтев корисника обавештењима путем прозора из *ngx-toastr* пакета инсталираног са npm. У наредном поглављу, бавићемо се детаљима архитектуре система, модела базе података, као и изазова с којима смо се суочавали приликом имплементације одређених функционалности.

5. ОПИС АРХИТЕКТУРЕ И РЕАЛИЗАЦИЈЕ СИСТЕМА

У овом поглављу, бавићемо се архитектуром система апликације *GameTime*, разоткривајући кључне компоненте које чине њену основу. Размотрићемо модел базе података који чини основ свих функционалности система, омогућавајући сигурно чување, ефикасно претраживање и манипулацију подацима. Такође, фокусираћемо се на најбитније проблеме настале приликом израде апликације, као и на то како су они препознати, анализирани, и на крају успешно решени.

5.1. Архитектура система

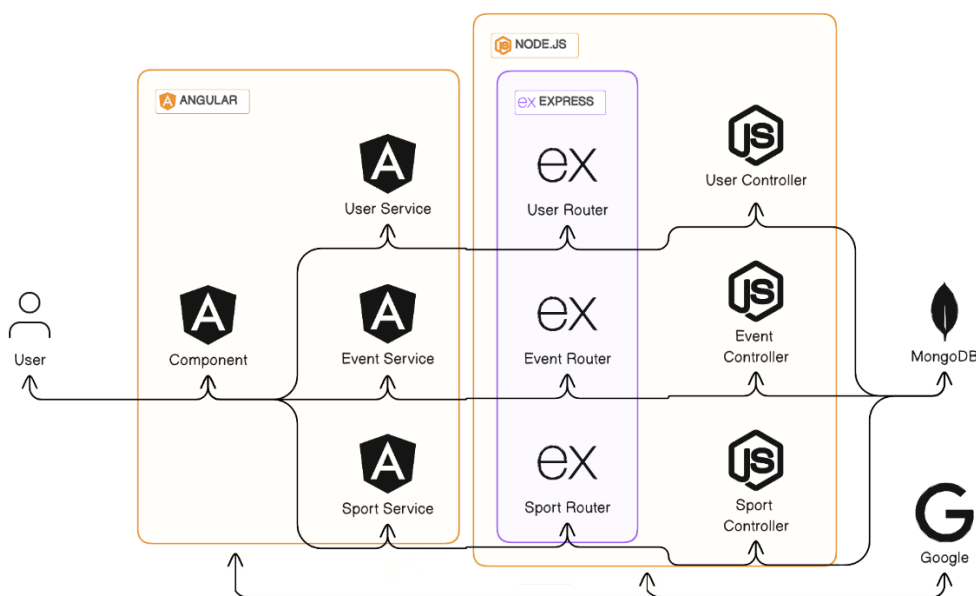
Апликација коју описујемо базира се на MEAN (*MongoDB*, *Express.js*, *Angular*, *Node.js*) архитектури. На врху ове структуре је корисник, који интерагује са апликацијом преко *Angular* компонената. Унутар ове *Angular* апликације, компоненте користе три специфична сервиса: *User Service*, *Event Service* и *Sport Service*. Захваљујући *Dependency Injection* принципу, овим сервисима је убризган *HttpClient* сервис, што им омогућава да комуницирају директно са серверским делом апликације.

Када је реч о серверском делу, користи се *Express* радни оквир, који се ослања на *Node.js* платформу. Три централна рутера - *User Router*, *Event Router* и *Sport Router* задужена су за усмеравање долазних захтева. Они филтрирају и преусмеравају све захтеве који имају URL путање које почињу са *"/users"*, *"/events"* и *"/sports"*.

Функционалност ових рутера проширена је кроз *middleware* функције које су смештене у посебним контролерима - *User Controller*, *Event Controller* и *Sport Controller*. Ови контролери управљају комуникацијом са *MongoDB* базом података, извршавајући упите и враћајући одговоре према клијентском делу апликације.

Осим комуникације унутар саме апликације, постоји и интеракција са спољним сервисима. Са клијентске стране, апликација комуницира са *Google* сервисима како би пружила функционалности везане за мапе и локације. Са серверске стране, приликом пријаве корисника путем *Google* налога, користи се верификација *Google* токена како би се осигурала аутентичност корисника.

Ова архитектура пружа флексибилност и скалабилност потребну за савремене веб апликације, док у исто време омогућава брзу и ефикасну комуникацију између различитих делова система.



Слика 5.1.1. Архитектура апликације *GameTime*

5.2. Модел базе података

Модел базе података наше апликације је дизајниран тако да подржи главне функционалности система и да обезбеди једноставну и ефикасну организацију и приступ подацима. Основна структура модела се састоји из три централне колекције: *users*, *events* и *sports*. Колекција *users* садржи информације о корисницима, укључујући личне податке, информације за пријаву, статус налога и друге релевантне информације. Колекција *events* ће, како њено име указује, чувати податке о различитим догађајима који се дешавају или су планирани у оквиру апликације. Колекција *sports* је замишљена да обезбеди информације о различитим спортским активностима или дисциплинама које су доступне у оквиру платформе. У наставку ћемо детаљније размотрити сваку од ових колекција, пружајући опис за свако од поља која чине структуру документа унутар колекција.

5.2.1. *users*

Опис поља у колекцији *users*:

- *_id* - Аутоматски генерисани идентификатор документа који *MongoDB* креира за сваки нови документ. Овај идентификатор је специфичан за *MongoDB* и користи се за ефикасну и брзу претрагу докумената унутар базе.
- *id* - Јединствени идентификатор корисника који је додат ради поједностављења у коришћењу. Будући да је представљен као цели број, често је лакши за коришћење у апликационом коду у односу на *_id*.
- *name* - Име корисника.
- *lastname* - Презиме корисника.
- *username* - Корисничко име корисника.
- *password* - Хеширана лозинка корисника.
- *email* - Електронска адреса корисника.
- *type* - Тип корисничког налога. Корисник може бити учесник, организатор, администратор.
- *status* - Тренутни статус корисничког налога. Може имати вредности "активан" или "неактиван".
- *phone* - Телефонски број корисника.
- *picture* - Име датотеке која се користи као профилна слика.
- *subscriptions* - За учесника, низ корисничких имена организатора које прати. За организатора, низ корисничких имена учесника који га прате. За администратора, овај низ је увек празан.
- *description* - За организатора, опис профила. За учесника и администратора, ово поље је увек празно.

5.2.2. *events*

Опис поља у колекцији *events*:

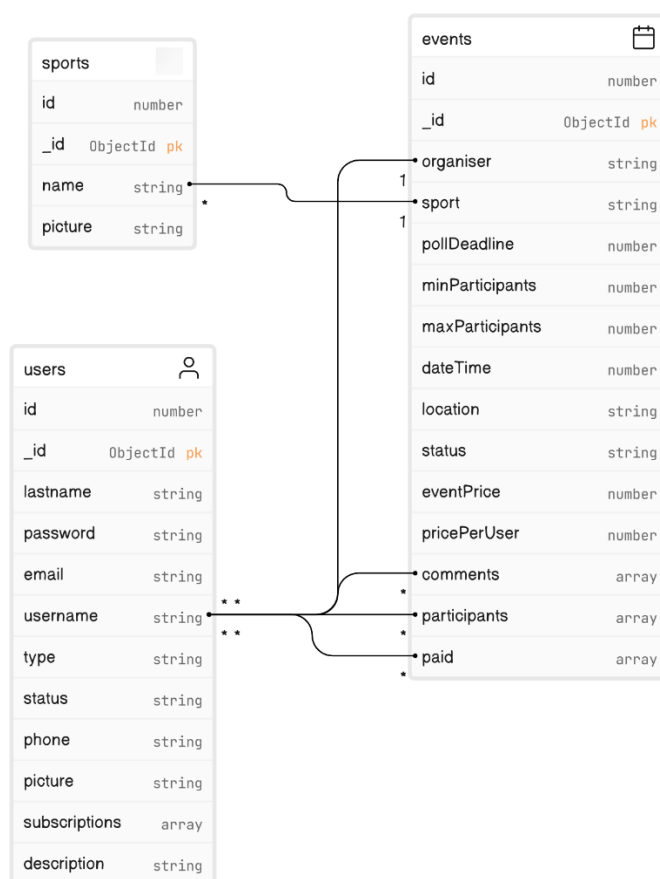
- *_id* - Аутоматски генерисани идентификатор документа који *MongoDB* креира за сваки нови документ.
- *id* - Јединствени идентификатор догађаја који је додат ради поједностављења у коришћењу.
- *organiser* - Корисничко име особе која је организатор догађаја.

- *sport* - Врста спорта који је тема догађаја.
- *pollDeadline* - Датум и време крајњег рока за пријаву на догађај.
- *minParticipants* - Минимални број учесника потребан за организацију догађаја.
- *maxParticipants* - Максимални број учесника који могу учествовати у догађају.
- *dateTime* - Датум и време одржавања догађаја.
- *location* - Локација на којој ће се догађај одржати.
- *status* – Статус догађаја, који може бити активан, завршен, или отказан.
- *eventPrice* - Укупна цена догађаја.
- *pricePerUser* - Цена по учеснику.
- *comments* - Низ коментара повезаних са догађајем. Сваки коментар има своја поља: *id*(целобројни идентификатор коментара), *username*(корисничко име онога ко је коментарисао), *name*(име онога који је коментарисао), *lastname*(презиме онога који је коментарисао), *datetime*(датум и време остављања коментара) и *text*(текст коментара).
- *participants* - Листа корисничких имена особа које учествују у догађају.
- *paid* - Листа корисничких имена особа које су платиле учешће у догађају.

5.2.3. *sports*

Опис поља у колекцији *sports*:

- *_id* - Аутоматски генерисани идентификатор документа који *MongoDB* креира за сваки нови документ.
- *id* - Јединствени идентификатор спорта који је додат ради поједностављења у коришћењу.
- *name* - Име спорта.
- *picture* - Име датотеке која се користи као слика за представљање спорта.



Слика 5.2.1. Модел базе података

5.3. Детаљи имплементације и приступ решавању проблема

У овој секцији, усмеравамо пажњу на дубљу анализу техничке реализације нашег система. Сваки софтверски пројекат доноси са собом низ јединствених изазова који захтевају детаљна и промишљена решења. Како бисмо осигурали транспарентност и дали увид у различите аспекте развоја, описаћемо кључне тачке и методе које смо применили да бисмо превазишли неке од техничких изазова приликом израде апликације *GameTime*.

5.3.1. Иницијализација сервера

Приликом иницијализације серверске апликације, многи кораци су важни да би се осигурао сигуран, ефикасан и функционалан сервер. Неколико кључних корака који се примењују у нашој апликацији обухватају:

- 1) **Подизање Express апликације:** Креирање нове *Express* апликације која нам омогућава да дефинишемо руте, користимо *middleware* и обрађујемо *HTTP* захтеве.

- 2) **Коришћење CORS(*Cross-Origin Resource Sharing*):** Омогућавање комуникације између различитих домена, конкретно клијентске стране која је покренута на порту 4200, и серверске стране која је покренута на порту 4000.
- 3) **Обрада JSON објекта из тела захтева:** Коришћење *body-parser* пакета за читање JSON објекта из тела захтева.
- 4) **Повезивање са *MongoDB*:** Успостављање конекције са базом података коришћењем *Mongoose*.
- 5) **Коришћење рутера за обраду захтева:** Дефинисање различитих рута за апликацију коришћењем *Express Router* објекта.
- 6) **Покретање сервера на порту 4000:** Ослушкивање долазних захтева на дефинисаном порту.
- 7) **Коришћење варијабли окружења за повећану сигурност:** Учитавање осетљивих информација попут стрингова за повезивање са базом података из *.env* датотеке путем *dotenv* пакета. Ово омогућава да се осетљиве информације чувају сигурно и изоловано од основног кода, штитећи их од неовлашћеног приступа или случајног дељења.

Узимајући у обзир разне техничке и сигурносне аспекте током процеса иницијализације серверског дела, осигурали смо да наша апликација удовољава модерним техничким стандардима. Кроз све ове кораке, циљ је био јасан: креирати апликацију која је брза, стабилна, сигурна и прилагодљива. Док се технологија и захтеви тржишта непрестано мењају, верујемо да је пажљивом реализацијом ове иницијализације постављен чврст темељ за будуће унапређење и адаптацију наше апликације.

```

// uvoz paketa i rutera
import express from 'express';
import cors from 'cors'
import bodyParser from 'body-parser'
import mongoose from 'mongoose'
import userRouter from './routers/user.router';
import eventRouter from './routers/event.router';
import sportRouter from './routers/sport.router';

// učitavanje varijabli iz .env datoteke
require("dotenv").config();

// podizanje Express aplikacije
const app = express();
// za omogućavanje komunikacije izmedju različitih domena
app.use(cors());
// za obradjivanje JSON objekta iz tela zahteva
app.use(bodyParser.json());

// povezivanje sa MongoDB bazom
mongoose.connect(process.env.DATABASE_URL);
const connection = mongoose.connection;
connection.once('open', () => {
  console.log('db connection ok')
})

// obrada zahteva korišćenjem Express Router
const router = express.Router();
app.use('/', router);
router.use('/users', userRouter);
router.use('/events', eventRouter);
router.use('/sports', sportRouter);

// pokretanje servera na portu 4000
app.listen(4000, () => console.log(`Express server running on port 4000`));

```

Део кода 5.3.1.1. Иницијализација сервера

5.3.2. Аутентификација и ауторизација корисника

У савременим веб апликацијама, безбедност корисничких података и заштита ресурса су од суштинског значаја. Две главне компоненте овог система безбедности су аутентификација (утврђивање идентитета корисника) и ауторизација (давање дозвола аутентификованом кориснику за приступ ресурсима). У наставку ће по корацима бити објашњено како се постижу ове две компоненте.

- 1) **Хеширање лозинке:** Када корисник креира нови налог, његова лозинка се никада не сме чувати у изворном облику у бази података због сигурносних разлога. Уместо тога, користи се процес звани хеширање да се трансформише лозинка у јединствену ниску која је тешко реверзибилна. Када се лозинка хешира, додаје се случајно генерисана вредност, односно *salt*. Ова вредност обезбеђује да чак и ако два

корисника имају исту лозинку, њихови резултујући хешеви ће бити различити. Механизам хеширања се у апликацији постиже помоћу *bcrypt* пакета [19] који генерише и користи вредност *salt* интерно, кад год се позове метода за хеширање. Алгоритам хеширања се ради сигурности обично врши у више итерација. Вредност од 10 итерација је често препоручена.

```
register = async (req: express.Request, res: express.Response) => {
  // ovde se dohvataju podaci iz tela zahteva
  // ...
  try {
    // provere jedinstvenosti korisničkog imena i mejla
    // ...

    // dohvatanje lozinke koju je korisnik uneo
    const password: string = req.body.password;

    // broj iteracija algoritma
    const saltRounds = 10;

    // heširanje izворne lozinke
    const hashedPassword = await bcrypt.hash(password, saltRounds);

    // kreira se novi korisnik sa heširanom lozinkom
    const newUser = new User({
      // ...
      password: hashedPassword,
      // ...
    });

    // ...

    // vraćanje odgovora korisniku sa informacijom o uspehu
    return res.status(200).json({
      "message": "Uspešno ste se registrovali kao " + type + "!"
    });
  } catch (error) {
    // ...
  }
}
```

Део кода 5.3.2.1. Хеширање лозинке

- 2) **Провера креденцијала и издавање JWT(*JSON Web Token*) токена:** Када корисник покуша да се пријави на систем, неопходно је проверити да ли унета лозинка одговара оној која је сачувана у бази података. Будући да се лозинке чувају у хешираном облику (из сигурносних разлога), процес провере не укључује директно упоређивање две лозинке. Уместо тога, користи се алгоритам за проверу хеша. По успешној провери креденцијала, креирамо JWT токен који садржи основне

информације о кориснику и потписујемо га тајним кључем (у овом случају, кључем сачуваним у окружењу као JWT_SECRET_KEY). Овај токен ће кориснику омогућити да приступи заштићеним ресурсима система док је токен важећи (у овом систему, токен истиче за 1 сат).

```
login = async (req: express.Request, res: express.Response) => {
  // dohvatanje podataka iz tela zahteva
  const username: string = req.body.username;
  const password: string = req.body.password;

  try {
    // provera da li postoji aktivan korisnik sa datim korisničkim imenom
    const user = await User.findOne({ "username": username, "status":"aktivan" });

    // vraćanje odgovora sa informacijom o grešci
    if (!user) {
      return res.status(400).json({ "message": "Neispravni kredencijali!" });
    }

    // provera da li je lozinka ispravna
    const isPasswordMatch = await bcrypt.compare(password, user.password);

    // vraćanje odgovora sa informacijom o grešci
    if (!isPasswordMatch) {
      return res.status(400).json({ "message": "Neispravni kredencijali!" });
    }

    // podaci za čuvanje u tokenu
    const jwtData = {
      username: user.username,
      name: user.name,
      lastname: user.lastname,
      role: user.type
    };

    // kreiranje i potpis tokena sa trajanjem 1 sat
    const token = jwt.sign(jwtData, process.env.JWT_SECRET_KEY, { expiresIn: '1h' });

    // vraćanje tokena kao odgovor
    return res.status(200).json({ token });
  } catch (error) {
    // ...
  }
}
```

Део кода 5.3.2.2. Провера креденцијала и издавање JWT токена

- 3) **Слање захтева са JWT токеном:** Приликом слања захтева ка серверу, корисник заједно са подацима потребним за извршавање обраде шаље и токен који му је издат приликом пријаве на систем. На овај начин, систем може да провери идентитет корисника и изврши захтевану обраду. Следи пример администраторске

функционалности брисања корисника, како бисмо боље разумели процес ауторизације.

Прво, на клијентској страни, администратор прикупља неопходне информације о кориснику ког жели да обрише:

```
async deleteUser(): Promise<void> {
  // podaci za slanje na server
  const data = {
    // korisničko ime korisnika koji se briše iz sistema
    username: this.username
  };

  try {
    // dohvatanje tokena iz sessionStorage
    const token: string = sessionStorage.getItem("token");
    // poziv metode iz userService koja će zahtev poslati serveru
    const response = await lastValueFrom(this.userService.deleteUser(data,
token));

    // ...
  } catch (error) {
    // ...
  }
}
```

Део кода 5.3.2.3. Позив методе сервиса *userService*

Токен и подаци се прослеђују *userService* сервису, који ће преко *HttpClient* модула да пошаље захтев серверу.

```
deleteUser(data: Object, token: string): Observable<Object> {
  // postavljanje tokena u zaglavlje zahteva
  const headers = new HttpHeaders().set('Authorization', 'Bearer ' + token);
  // slanje zahteva ka serveru na putanju '/users/deleteUser'
  return this.http.post(`${this.uri}/deleteUser`, data, {headers});
}
```

Део кода 5.3.2.4. Слање HTTP захтева са заглављем

На серверу, када се прими захтев, *middleware* функција *verifyTokenMiddleware* прво проверава да ли токен постоји у заглављу. Ако токен постоји, онда се помоћу пакета *jsonwebtoken* верификује валидност токена и проверава се улога корисника. Уколико је токен валидан и улога корисника се поклапа са дозвољеним улогама (у овом случају, само администратор може да обрише корисника), захтев се прослеђује даље на обраду.

```

userRouter.route("/deleteUser").post(
  // prvo se verifikuje token, a zatim kontroler vrši obradu
  verifyTokenMiddleware(["administrator"]),
  (req, res) => new UserController().deleteUser(req, res)
);

```

Део кода 5.3.2.5. Рутирање захтева

```

export const verifyTokenMiddleware = (allowedUserTypes: string[]) => {
  return (req: Request, res: Response, next: NextFunction) => {
    // dohvatanje tokena iz zaglavlja zahteva
    const token: string = req.headers.authorization?.split(' ')[1] || '';

    if (!token) {
      return res.status(400).json({ "message": "Nema tokena u zaglavlju!" });
    }

    try {
      // verifikacija tokena
      const decodedToken = jwt.verify(token, process.env.JWT_SECRET_KEY);
      const role = decodedToken.role;

      // proverava da li je odgovarajuća uloga korisnika
      if (!allowedUserTypes.includes(role)) {
        return res.status(401).json({ "message": "Nemate pristup ovoj usluzi!" });
      }

      // ako je sve u redu, obraditi zahtev
      next();
    } catch (error) {
      return res.status(403).json({ "message": "Vaša sesija je istekla!" });
    }
  };
};

```

Део кода 5.3.2.6. Верификација токена

```

deleteUser = (req: express.Request, res: express.Response) => {
  // korisničko ime korisnika koji se briše
  const username:string = req.body.username;
  // postavljanje statusa korisnika na neaktivan
  User.collection.updateOne({ "username": username }, { $set: { status:
"neaktivan" } }, (error, success) => {
    // obaveštavanje korisnika o neuspehu ili uspehu operacije
    if (error) {
      return res.status(400).json({ "message": "Greška pri brisanju
korisnika!", error });
    }
    else res.status(200).json({ "message": "Korisnik je obrisani!" })
  })
}

```

Део кода 5.3.2.7. Обрада захтева за брисање корисника

5.3.3. Обрада пријаве помоћу Google налога

Пријављивање кроз *Google* је све популарнији начин аутентификације корисника у веб апликацијама [20]. Омогућава корисницима брзу пријаву без потребе за памћењем крeденцијала и користи сигурносне протоколе *Google* за верификацију идентитета. Овде је приказано како се то постиже користећи *Google Sign-In JavaScript* библиотеку на клијентској страни и *google-auth-library* на серверској страни.

- 1) Први корак је укључивање библиотеке преко *index.html*:

```
<script src="https://accounts.google.com/gsi/client" async defer></script>
```

Део кода 5.3.3.1. Укључивање *Google Sign-In JavaScript* библиотеке

- 2) На страници за пријаву се врши иницијализација дугмета за *Google* пријаву и дефинише callback функција за обраду након добијања *Google* токена.

```
ngOnInit(): void {
  // ...

  // referenca na Google Accounts API.
  const gAccounts: accounts = google.accounts;

  // inicijalizacija Google Sign In komponente.
  gAccounts.id.initialize({
    // Google identifikator klijenta iz okruženja
    client_id: environment.googleClientId,

    /* ova opcija definiše da će se Google prijavljivanje otvarati u popup
    prozoru, a ne kao preusmeravanje na drugu stranicu */
    ux_mode: 'popup',

    // dodatna podešavanja za popup prozor
    //...

    // 'credential' je objekat koji sadrži Google token
    callback: ({ credential }) => {
      // delegiranje obrade drugoj metodi
      this.ngZone.run(() => {
        this.googleSignIn(credential);
      });
    },
  });

  // renderuje Google dugme za prijavu unutar elementa sa id 'gbtn'
  gAccounts.id.renderButton(document.getElementById('gbtn') as HTMLElement, {
    // podešavanje širine, oblika, boje
    // ...
  });
}
```

Део кода 5.3.3.2. Иницијализација дугмета за пријаву преко *Google*

- 3) На сличан начин као у примеру са брисањем корисника, захтев за пријаву преко *Google* налога се шаље серверу. Сервер извучи информације из токена након што га верификује [21], и уколико је корисник већ регистрован, враћа му одговор у виду JWT токена. Уколико корисник није регистрован, биће аутоматски креиран његов налог, а преостале информације(тип корисника, број телефона, опис) се накнадно уносе.

```
googleSignIn = async (req: express.Request, res: express.Response) => {
  // dohvatanje tokena iz tela zahteva
  const googleToken: string = req.body.token;
  try {
    // verifikacija tokena
    const ticket = await client.verifyIdToken({
      idToken: googleToken,
      audience: process.env.GOOGLE_CLIENT_ID // CLIENT_ID aplikacije
    });

    // dohvatanje mejla iz payload dela tokena
    const payload = ticket.getPayload();
    const email = payload.email;
    // proverava da li korisnik već ima nalog
    const user = await User.findOne({ "email": email });

    // proverava da li je korisnik aktivan
    if (user && user.status === "aktivan") {
      const jwtData = {
        // ...
      }

      // kreiranje i potpis tokena
      const token = jwt.sign(jwtData, process.env.JWT_SECRET_KEY,
{ expiresIn: '1h' });
      return res.status(200).json({ token });
    }
    //...

    // dohvatanje preostalih informacija iz payload dela
    const name: string = payload.given_name;
    const lastname: string = payload.family_name;
    const picture: string = payload.picture;
    //...

    // dodavanje novog korisnika
    const newUser = new User({
      // ...
    });
    //...
    // slanje odgovora korisniku...
  } catch (error) {
    // ...
  }
}
```

Део кода 5.3.3.3. Обрада *Google* токена на серверу

5.3.4. Аутоматизам система и рад са мејловима

Аутоматизација је саставни део савремених апликација како би се обезбедило редовно ажурирање, обавештавање корисника или одржавање апликације. У овом делу, уз помоћ *node-schedule* библиотеке, имплементиран је систематичан приступ за праћење и управљање догађајима, а уз помоћ *nodemailer* библиотеке, корисници се обавештавају путем мејла. Одржавање се врши на дневном нивоу. Систем на крају дана врши ажурирање статуса догађаја и шаље мејл обавештења кориснику.

```
const schedule = require("node-schedule");
const nodemailer = require("nodemailer");

schedule.scheduleJob("5 0 * * *", () => {
  // poziv metode svakog dana u 00:05
  new EventController().cancelEventsWithoutMinimum();
});

schedule.scheduleJob("10 0 * * *", () => {
  // poziv metode svakog dana u 00:10
  new EventController().updateEventsStatus();
});
```

Део кода 5.3.4.1. Аутоматизација система

```
sendEmail = async (emails: string, subject: string, html: string) => {
  // kreiranje transportera koristeći konfiguraciju iz okruženja.
  const transporter = nodemailer.createTransport({
    host: process.env.HOST, // smtp.gmail.com - Gmail SMTP server
    port: parseInt(process.env.PORT, 10), // port 465 - Gmail SSL port
    secure: true, // koristi se SSL/TLS enkripcija za veću sigurnost
    auth: {
      user: process.env.EMAIL, // mejl adresa koja će se koristiti za slanje
      pass: process.env.PASS /* "lozinka za aplikacije" koja je generisana na
Google nalogu */
    }
  });
  // opcije mejla
  const mailOptions = {
    from: process.env.EMAIL, // adresa pošiljaoca (administratora)
    to: emails, // adrese primalaca razdvojene sa ", "
    subject: subject, // naslov mejla
    html: html // sadržaj mejla u HTML
  };
  // Slanje mejla koristeći transportera i definisane opcije.
  transporter.sendMail(mailOptions, function (error, info) {
    if (error) {
      // ...
    } else {
      // ...
    }
  });
}
```

Део кода 5.3.4.2. Помоћна метода за слање мејлова

```

updateEventsStatus = async () => {
  const now: number = Date.now();
  try {
    // nalaženje aktuelnih događaja kojima je istekao rok za prijavu
    const events = await Event.find({ "status": "aktivan", "pollDeadline": { $lt:
now } });

    // slanje mejlova kao podsetnik na događaj za taj dan
    for (const event of events) {
      // ažuriranje statusa događaja na 'završen' i računanje cene po učesniku
      //...

      // slanje mejlova svim prijavljenim učesnicima
      this.sendEmail(emails, title, content);
    } catch (error) {
      // ...
    }
  }
};

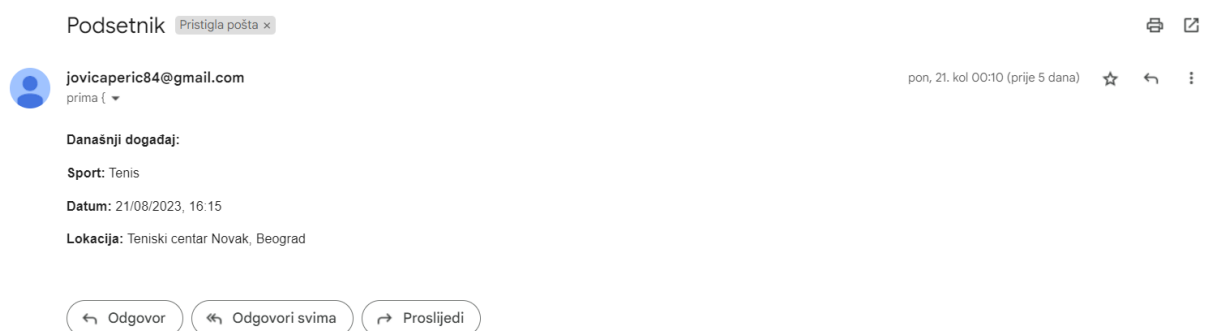
cancelEventsWithoutMinimum = async () => {
  const now: number = Date.now();
  try {
    // nalazi aktuelne događaje koji nisu ispunili minimalan broj učesnika
    const events = await Event.find({
      "status": "aktivan",
      "pollDeadline": { $lt: now },
      $expr: { $lt: [{ $size: "$participants" }, "$minParticipants"] }
    });

    // slanje mejlova o otkazivanju
    for (const event of events) {
      // ažuriranje statusa događaja na 'otkazan'
      // ...

      // slanje mejlova svim prijavljenim učesnicima
      this.sendEmail(emails, title, content);
    } catch (error) {
      // ...
    }
  }
};

```

Део кода 5.3.4.3. Методе за обраду актуелних догађаја



Слика 5.3.4.1. Пример мејла подсетника

5.3.5. *Рад са мапама*

Данас многе апликације користе мапе [22] како би побољшале корисничко искуство, омогућавајући корисницима да лако идентификују, претражују и филтрирају локације. У контексту организације догађаја, мапе доносе неколико кључних предности:

- 1) **Приказ локација на мапи:** Ова функционалност омогућава корисницима да брзо и лако разумеју где се одређени догађај одржава.
- 2) ***Autocomplete* претрага по местима:** Док корисници уносе назив места или адресу, апликација може да предложи одговарајуће локације, што убрзава процес уноса и смањује могућност грешке.
- 3) **Филтрирање локација у близини корисника:** Ово омогућава корисницима да ограниче резултате на оне локације које су им најрелевантније и налазе се у њиховој непосредној близини.

Geocoder API се користи за претварање текстуалне адресе у географске координате, које се затим користе за приказ те локације на мапи.

```
// deklaracija Google mape
map: google.maps.Map;

async initMap(address: string): Promise<void> {
  // uvoz potrebnih biblioteka za rad sa Google mapama i markerima
  const { Map } = await google.maps.importLibrary("maps") as
google.maps.MapsLibrary;
  const { AdvancedMarkerElement } = await google.maps.importLibrary("marker") as
google.maps.MarkerLibrary;

  /* kreira se novi geocoder, koji će transformisati zadatu tekstualnu adresu u
geografske coordinate */
  const geocoder = new google.maps.Geocoder();

  geocoder.geocode({ address }, (results, status) => {
    if (status === google.maps.GeocoderStatus.OK) {
      // iz rezultata geokodiranja dobijaju se geografske koordinate
      const lat = results[0].geometry.location.lat();
      const lng = results[0].geometry.location.lng();

      // definisanje pozicije na osnovu dobijenih koordinata
      const position = { lat, lng };

      // inicijalizacija mape sa zadatim centrom i veličinom
      this.map = new Map(
        document.getElementById('map') as HTMLElement,
        {
          zoom: 16,
          center: position,
        }
      );

      // dodavanje markera na mapu kako bi korisnik znao tačnu lokaciju
      const marker = new AdvancedMarkerElement({
        map: this.map,
        position: position,
        title: 'Lokacija'
      });
    } else {
      // ukoliko geokodiranje nije uspešno, ispisuje se razlog neuspeha
      console.log('Geocode was not successful for the following reason: ' +
status);
    }
  });
}
```

Део кода 5.3.5.1. Приказ локације на мапи

Наредни део кода се односи на *Google Places Autocomplete* сервис, који омогућава аутоматско попуњавање претраге за места користећи унос корисника.

```
// definisanje promenljive za Google autocomplete
autocomplete: google.maps.places.Autocomplete;

initAutocomplete(): void {
    // kreiranje novog autocomplete objekta.
    this.autocomplete = new google.maps.places.Autocomplete(
        document.getElementById("location") as HTMLInputElement, { /* HTML element
u koji korisnik unosi tekst */
        types: [], /* prazan niz znači da nema specifičnih ograničenja pri
pretrazi tipova lokacija */
        componentRestrictions: { "country": ["RS"] }, /* pretraga ograničena na
Srbiju (RS) */
        fields: ["place_id", "geometry", "name", "formatted_address"] /* polja
koja želimo da dobijemo nazad od servisa */
    });
}
```

Део кода 5.3.5.2. Омогућавање *autocomplete* функционалности

Филтрирање локација у близини корисника ће бити објашњено по корацима. Прво је потребно дефинисати променљиву која ће означавати радијус у km у којем корисник жели да прикаже догађаје.

```
selectedRadius: number = -1; // default vrednost, filtriranje nije primenjeno
```

Помоћу *Geolocation* API добија се тренутна локација корисника, на основу које ће се даље вршити филтрирање.

```
getCurrentLocation(): void {
    // proverа podrške za geolokaciju
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition((position: GeolocationPosition) =>
        {
            // dobijanje koordinata trenutne lokacije
            const pos = {
                lat: position.coords.latitude,
                lng: position.coords.longitude,
            };
        });
        //...

        // poziv funkcije za filtriranje
        this.filterEvents(pos);
    } else {
        console.error("Geolocation not supported by this browser.");
    }
}
```

Део кода 5.3.5.3. Дохватање тренутне локације корисника

Затим се на раније приказани начин помоћу *Geocoder* сервиса добијају координате локација догађаја, а затим се применом *Haversine* формуле [23] рачуна удаљеност између локације корисника и локације догађаја. Локације се филтрирају по удаљености према раније дефинисаном радијусу.

```
async filterEvents(pos: { lat: number; lng: number }) {
  // ...

  // geokodiranje lokacija svih događaja
  const eventLocations = await Promise.all(geocodePromises);

  // filtriranje događaja
  const filteredEvents = this.allActiveEvents.filter((event, index) => {
    const distance = this.getDistance(pos, eventLocations[index]);
    return distance <= this.selectedRadius * 1000; // pretvaramo radijus iz km u m.
  });

  // ažuriramo listu aktuelnih događaja
  this.activeEvents = filteredEvents;
}
```

Део кода 5.3.5.4. Филтрирање догађаја у дефинисаном радијусу

Удаљеност између две локације се рачуна помоћу *Haversine* формуле.

```
getDistance( location1: { lat: number; lng: number },
  location2: { lat: number; lng: number }): number {
  // помоћна funkcija za konvertovanje stepeni u radijane.
  const rad = (x: number): number => (x * Math.PI) / 180;

  const R = 6378137; // Zemljin poluprečnik u metrima.
  const dLat = rad(location2.lat - location1.lat);
  const dLng = rad(location2.lng - location1.lng);

  // Haversine formula za računanje udaljenosti između dve tačke na sferi.
  const a = Math.sin(dLat / 2) * Math.sin(dLat / 2) + Math.cos(rad(location1.lat))
    * Math.cos(rad(location2.lat)) * Math.sin(dLng / 2) * Math.sin(dLng / 2);
  const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
  const distance = R * c;

  return distance;
}
```

Део кода 5.3.5.5. Рачунање удаљености између две локације

5.3.6. Рад са сликама

Приликом развоја веб апликације, управљање сликама, посебно профилним сликама корисника, може представљати изазов. С обзиром на различите изворе слика, битно је разумети разлику у обради и дохватању слика које долазе из локалне похране (у овом случају, путем *multer* библиотеке) у односу на оне које долазе директно са *Google* сервера (за кориснике

који се пријављују путем *Google* налога). Ова два приступа се разликују у погледу складиштења, приступа и преноса.

```
getUserPicture = async (req, res) => {
  const picture = req.query.image; // dohvatanje imena slike iz zahteva

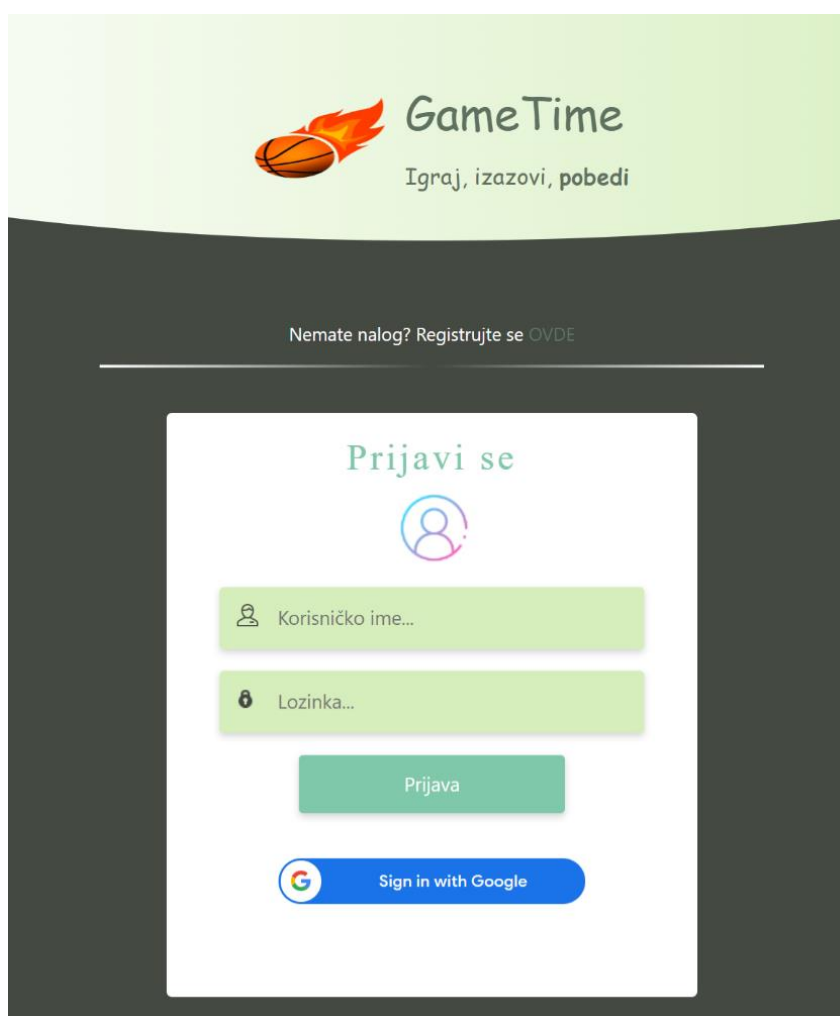
  try {
    if (!picture.includes("googleusercontent")) { /* provera da li je slika
upload-ovana lokalno */
      // ako jeste, šalje se slika sa lokalnog skladišta
      return res.sendFile(path.join(__dirname,
`../../uploads/users/${picture}`));
    }
    else { // ako slika dolazi sa Google servera
      const imageUrl = picture;
      // asinhroni zahtev ka Google serveru da se dohvati slika
      const response = await axios.get(imageUrl, { responseType: 'arraybuffer'
});
      // konverzija slike u format za prenos
      const imageBuffer = Buffer.from(response.data, 'base64');
      // postavljanje odgovarajućeg tipa za sliku
      const contentType = response.headers['content-type'];
      res.set('Content-Type', contentType);
      // šalje se slika klijentu
      res.send(imageBuffer);
    }
  } catch (error) {
    console.log(error);
    // u slučaju greške, vraća se odgovarajuća poruka klijentu
    return res.status(400).json({ "message": "Greška pri dohvatanju slike
korisnika!", error });
  }
};
```

Део кода 5.3.6.1. Враћање профилне слике корисника клијентској страни

6. ОПИС РАДА СИСТЕМА

У овом поглављу, с намером да пружимо детаљан и свеобухватан увид у функционисање нашег система, презентоваћемо апликацију *GameTime* кроз серију илустрација које демонстрирају рад апликације за сваку специфичну категорију корисника.

Опште функционалности које су доступне свакој категорији корисника су креирање налога и пријава на систем. Попуњавањем форми корисник креира нови налог или се пријављује уз постојеће креденцијале.



Слика 6.1. Почетни екран апликације(форма за пријаву)

Imate nalog? Prijavite se [OVDE](#)

Registruj se

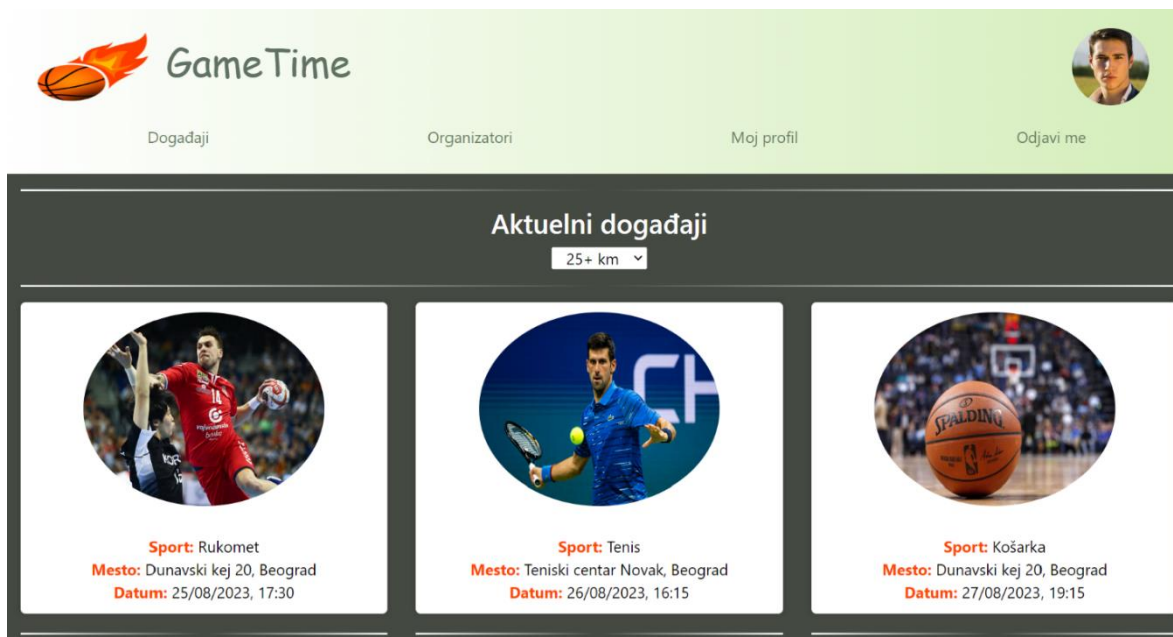
No file chosen

Sign in with Google

Слика 6.2. Форма за регистрацију

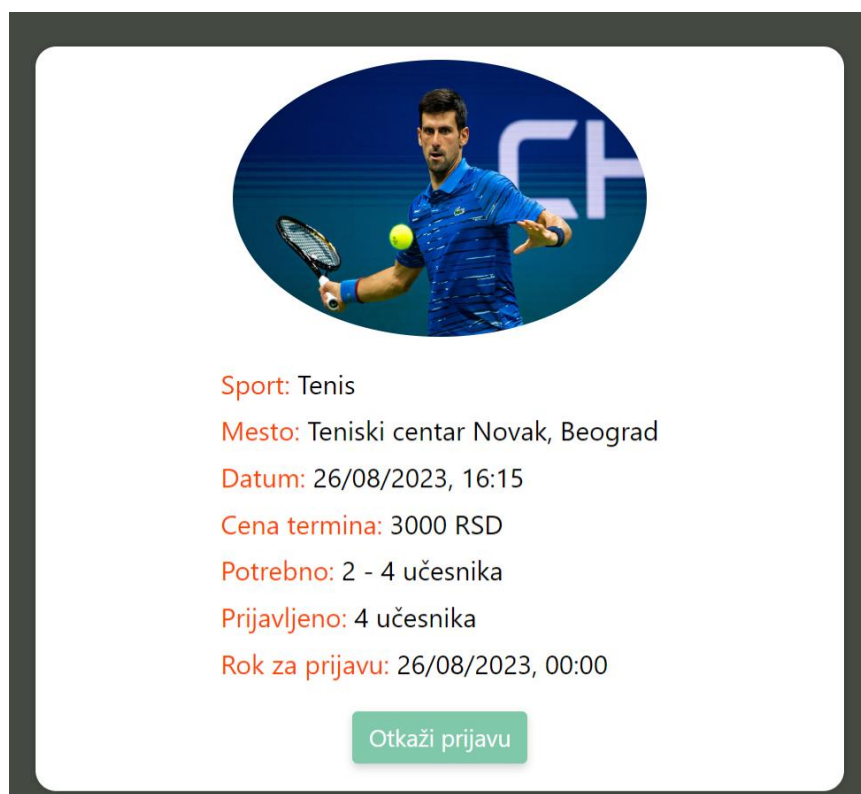
6.1. Опис рада учесника

У овој секцији ћемо приказати изглед апликације за најважније функционалности учесника. Учесник на почетној страни има јасан преглед свих актуелних догађаја у систему са основним информацијама, као и могућност филтрирања тих догађаја на основу удаљености од своје тренутне локације.

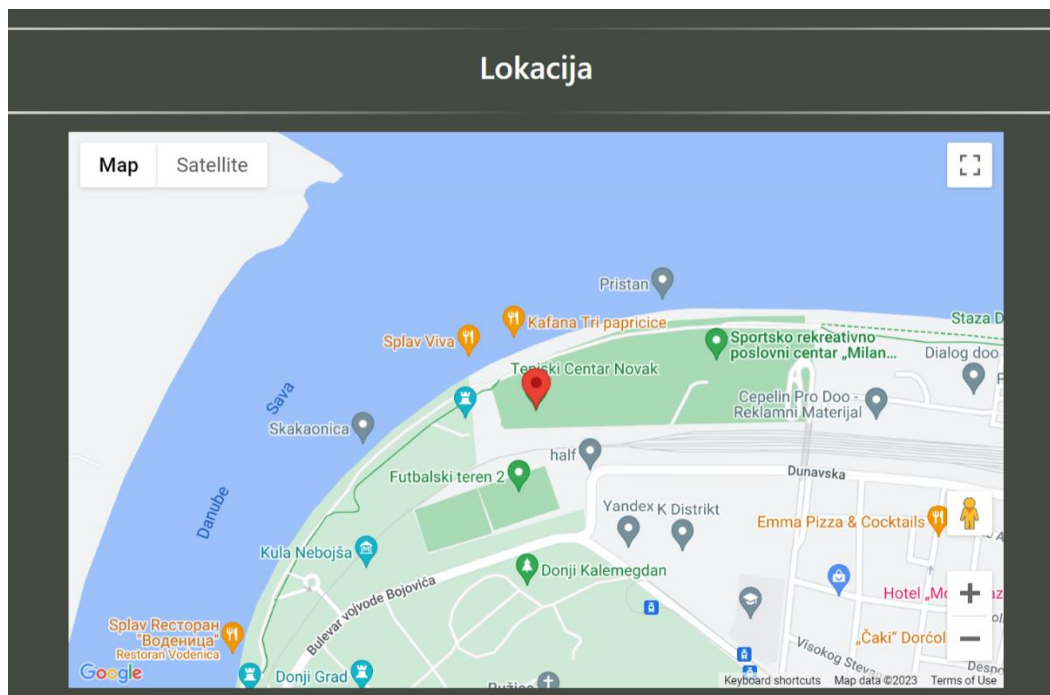


Слика 6.1.1. Почетни екран учесника

Кликом на неки од понуђених догађаја, отвара се нова страница са свим информацијама о догађају, као и приказом локације на мапи и секцијом за остављање коментара везаног за догађај. Такође, постоји опција пријаве/отказивања пријаве за актуелни догађај.

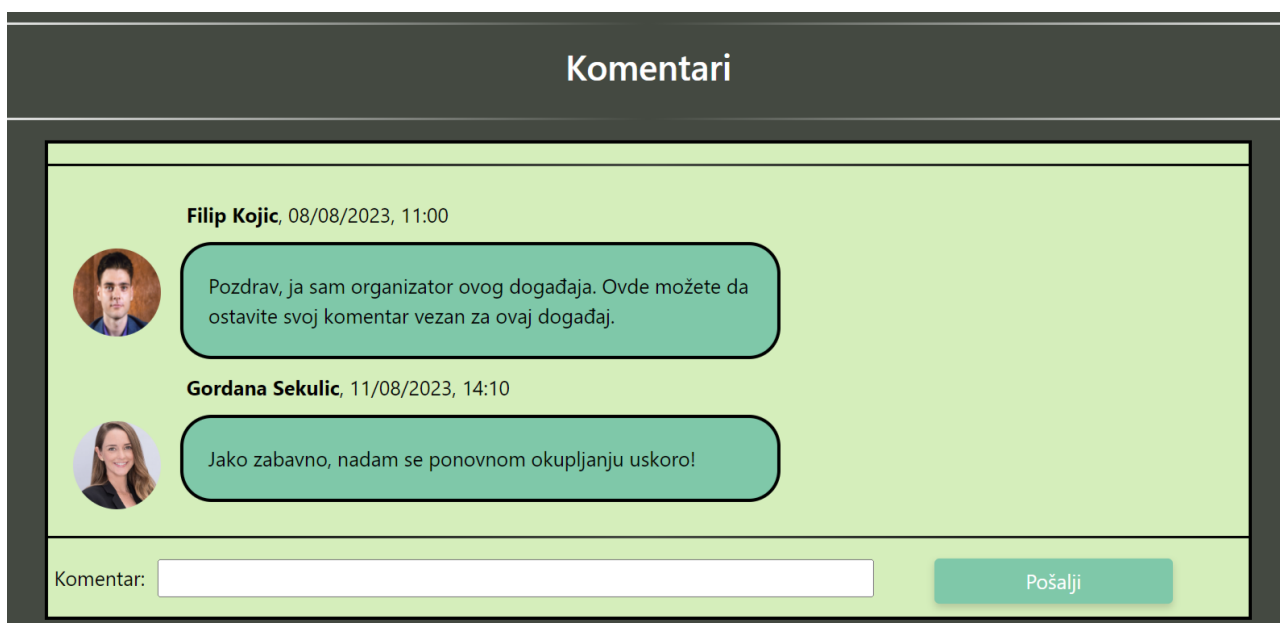


Слика 6.1.2. Приказ детаља о актуелном догађају



Слика 6.1.3. Приказ локације догађаја


Коментари се приказују хронолошки, са именом и презименом аутора, као и датумом и временом остављања коментара. Учесник може да остави свој коментар попуњавањем поља предвиђеног за то и кликом на дугме за слање.



Слика 6.1.4. Преглед коментара и форма за коментарисање

Учесник може да прегледа профиле организатора, односно њихову профилну слику и опис профила. Може да "запрати"/ "отпрати" организатора, чиме се пријављује/одјављује за добијање обавештења о догађајима тог организатора.


Organizatori



Lazar Mikic

Kao organizator događaja, moj cilj je da svaka prilika postane nezaboravno iskustvo. Moje bogato iskustvo omogućava mi da sa preciznošću i kreativnošću ...


Zaprati



Ana Jovic

Kao organizator, imam privilegiju da stvaram nezaboravne događaje. Moje iskustvo i kreativnost omogućavaju mi da svaki detalj pažljivo osmislim, stvar ...

Zaprati



Filip Kojic

Filip Kojic je stručan i posvećen sportski organizator, koji se ističe u organizaciji fudbalskih turnira, košarkaških liga i atletskih takmičenja. Nje ...

Zaprati

Слика 6.1.5. Преглед профила организатора

Одласком на страницу за преглед свог профила, учесник може да види све податке унесене при регистрацији, осим лозинке. Такође, види табеларни приказ свих догађаја на којима је претходно учествовао, заједно са линковима који воде на детаљан приказ догађаја. Може да види цену за учешће у свим претходним догађајима, као и колико је укупно дужан.

Moji podaci

Ime	Petar
Prezime	Simic
Korisničko ime	petar
Tip	ucesnik
Telefon	+381 66 4010257
Mejl	simicp393@gmail.com

Ukoliko želite da izmenite informacije o sebi, kliknite [OVDE](#)

Слика 6.1.6. Преглед свог профила


Prethodna učešća					
Sport	Mesto	Datum	Cena po učesniku	Plaćeno	Detalji
Odbojka	Odbojkaški tereni, Ada, Beograd	14/08/2023, 13:30	625 RSD	✓	LINK
Stoni tenis	Tržni centar Rajičeva, Beograd	13/08/2023, 16:30	250 RSD	✗	LINK
Fudbal	FK Dorćol, Beograd	12/08/2023, 14:15	1000 RSD	✓	LINK
Košarka	Dunavski kej 20, Beograd	11/08/2023, 19:00	500 RSD	✗	LINK
Tenis	Teniski centar Novak, Beograd	10/08/2023, 18:00	750 RSD	✗	LINK
Ukupan dug: 1500 RSD					


Слика 6.1.7. Преглед претходних учешћа


Додали смо *hover* ефекат тако да приликом преласка миша преко редова табеле, они мењају боју, чиме се постиже боља прегледност и лакша оријентација корисника. Исто је урађено са линковима кроз читаву апликацију.


Последња битна ставка везана за учесника је ажурирање профила. При одласку на страницу за ажурирање профила учесника, поља су попуњена његовим тренутним подацима. Уколико жели, може их изменити и сачувати. Подаци које учесник може да промени су мејл, телефон и профилна слика.

Ažuriraj profil



 simicp393@gmail.com

 +381 66 4010257

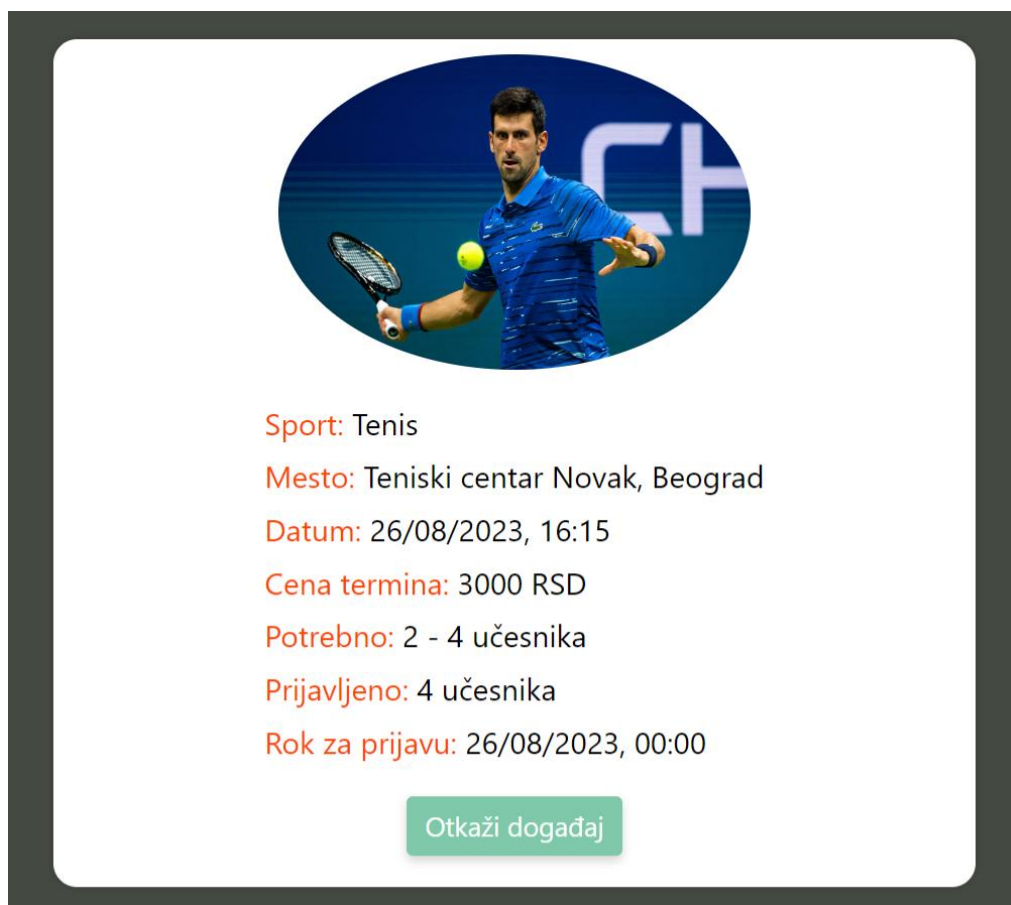

 No file chosen

Слика 6.1.8. Форма за ажурирање профила учесника

6.2. Опис рада организатора

Након детаљног приказа функционалности учесника, сада ћемо прећи на организатора, фокусирајући се притом на кључне аспекте специфичне за његову улогу, без осврта на сличне елементе који су већ обрађени у контексту учесника, а које такође поседује и категорија организатора.

Када се организатор пријави на систем, види актуелне догађаје које је он организовао са основним информацијама. Када кликне на неки од актуелних догађаја, добија детаљан приказ са свим релевантим информацијама. За разлику од учесника, организатор има опцију да откаже догађај, као и да види све пријављене учеснике.




Слика 6.2.1. Приказ актуелног догађаја за организатора

Učesnici			
Korisničko ime	Ime	Prezime	Detalji
gordana	Gordana	Sekulic	LINK
milan	Milan	Jokic	LINK
petar	Petar	Simic	LINK
stefan	Stefan	Rakic	LINK

Слика 6.2.2. Приказ пријављених учесника за догађај

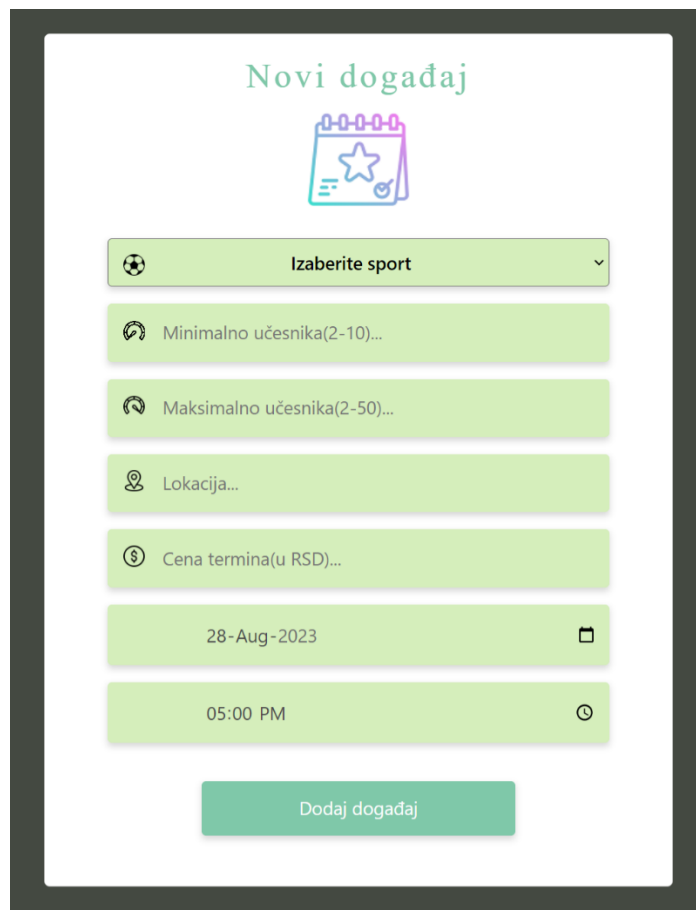
Кликом на линк, организатору се приказује профил учесника са профилном сликом и свим релевантним информацијама.



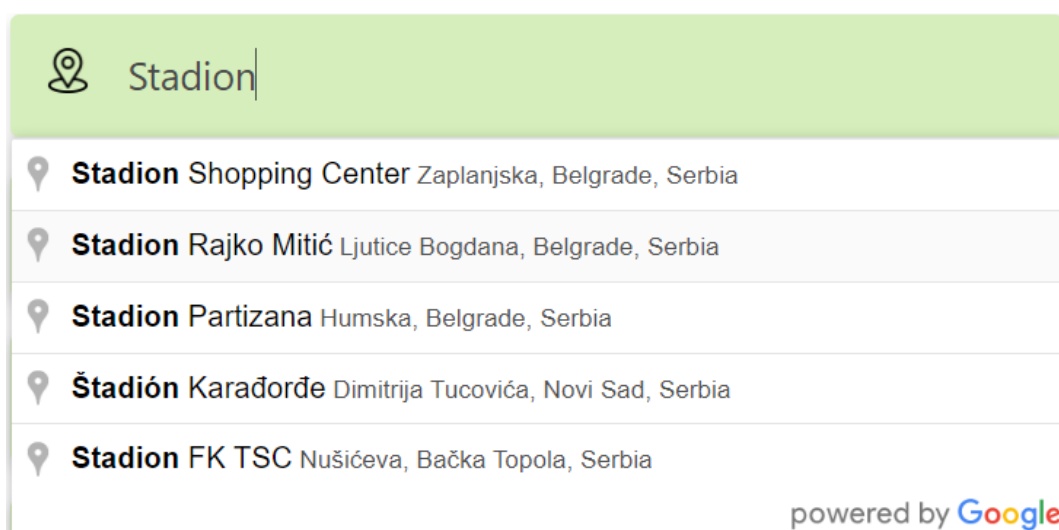
Ime	Petar
Prezime	Simic
Korisničko ime	petar
Tip	ucesnik
Telefon	+381 66 4010257
Mejl	simicp393@gmail.com
Status	aktivan

Слика 6.2.3. Приказ профила учесника организатору

Организатор има опцију додавања новог догађаја у систем, попуњавањем форме са свим неопходним подацима везаним за догађај. Путем *autocomplete* функционалности, организатор може да брзо и ефикасно лоцира жељено место или адресу за одржавање догађаја.



Слика 6.2.4. Форма за додавање новог догађаја



Слика 6.2.5. *Autocomplete* функционалност

Организатор приликом прегледа свог профила има могућност да прегледа своје претходно одржане догађаје. Кликом на линк за тај догађај, одлази на страницу са детаљним приказом информација о догађају. Додатна функционалност у односу на учесника јесте вођење евиденције о плаћању. Штиклирањем и чувањем избора, организатор има јасан преглед ко од учесника је платио, а ко му дугује новац за организовање догађаја.

Učesnici				
Korisničko ime	Ime	Prezime	Detalji	Platio
gordana	Gordana	Sekulic	LINK	<input checked="" type="checkbox"/>
milan	Milan	Jokic	LINK	<input checked="" type="checkbox"/>
petar	Petar	Simic	LINK	<input type="checkbox"/>
stefan	Stefan	Rakic	LINK	<input checked="" type="checkbox"/>
Ukupan dug: 500 RSD				
Sačuvaj				

Слика 6.2.6. Евидентирање плаћања

Приликом креирања или отказивања догађаја, учесницима који су пријавом или "праћењем" организатора везани за догађај, шаље се мејл обавештења са свим релевантним информацијама. Такође, на дан одржавања догађаја, пријављеним учесницима се шаље мејл подсетник.



Слика 6.2.7. Мејл обавештења о новом догађају

6.3. Опис рада администратора

Сада прелазимо на врло битну улогу у нашем систему - администратора. Администратор има посебна овлашћења и одговорности које га издвајају од осталих корисника. У наредном сегменту, детаљно ћемо се упознати са специфичностима и алатима који су на располагању особи са овом улогом у систему.

Када се администратор пријави на систем, види табеларни приказ свих учесника у систему. Кликом на линк у табели, отвара му се страница са детаљним приказом профила учесника. У менију има опцију да на идентичан начин прегледа све организаторе, заједно са линковима који воде на детаљан приказ њихових профила.



Učesnici				
Korisničko ime	Ime	Prezime	Status	Detalji
gordana	Gordana	Sekulic	aktivan	LINK
milan	Milan	Jokic	aktivan	LINK
milica1	Milica	Petrovic	aktivan	LINK
petar	Petar	Simic	aktivan	LINK
stefan	Stefan	Rakic	aktivan	LINK

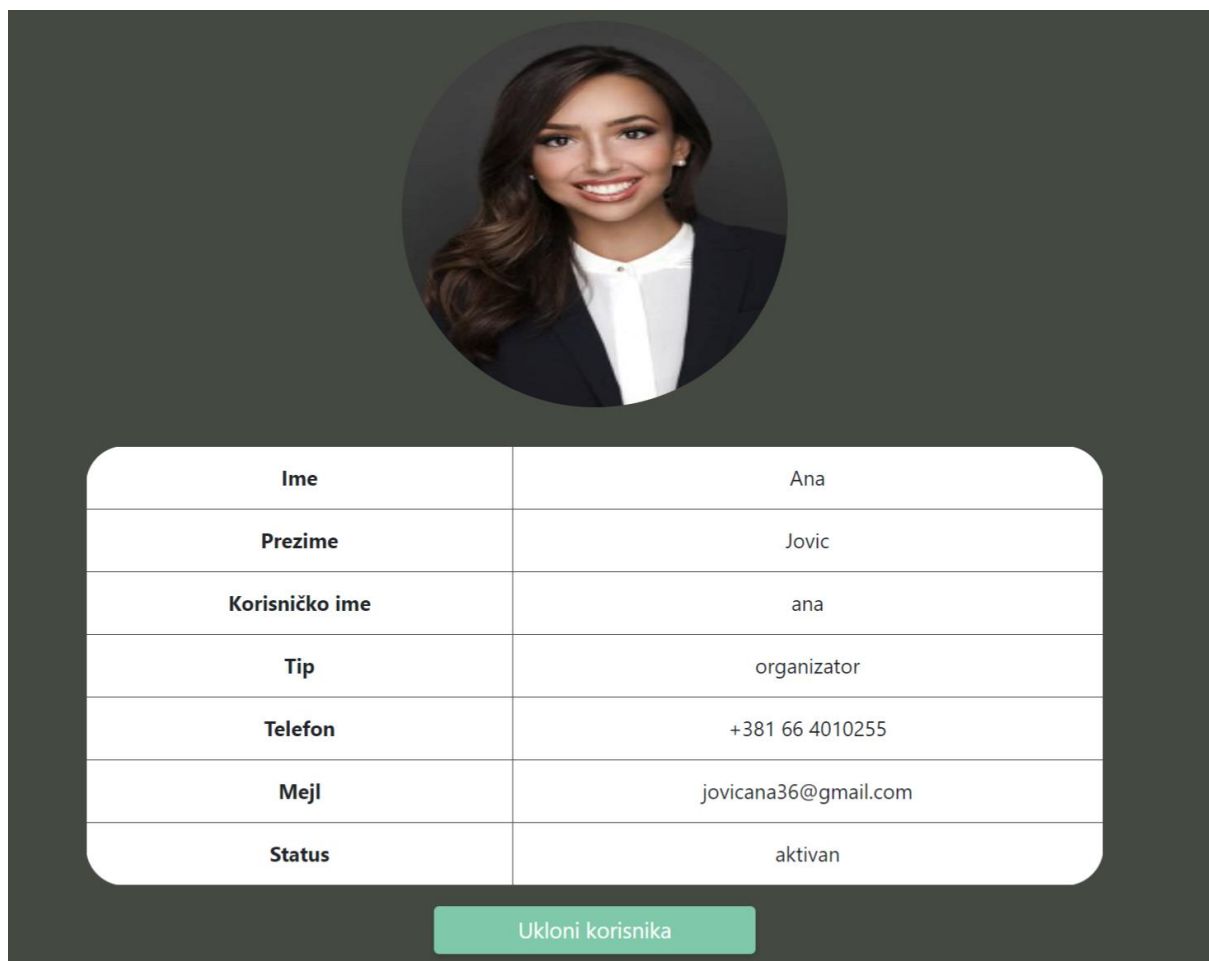
Слика 6.3.1. Преглед свих учесника система



Organizatori				
Korisničko ime	Ime	Prezime	Status	Detalji
ana	Ana	Jovic	aktivan	LINK
filip	Filip	Kojic	aktivan	LINK
lazar	Lazar	Milic	aktivan	LINK

Слика 6.3.2. Преглед свих организатора система

Када администратор прегледа профил учесника или организатора, види све што и они, односно основне информације, као и претходне догађаје у којима су учествовали. Додатна привилегија коју администратор има јесте брисање корисника из система, односно постављање његовог статуса на "неактиван". Када администратор то учини, корисник више неће моћи да приступа платформи, али се његов налог не уклања потпуно из система, ради сигурности и потенцијалних постојећих дуговања.



Слика 6.3.3. Преглед корисничког профила са опцијом брисања

Администратор, као и претходно описане категорије корисника, има функционалности попут прегледа и ажурирања свог профила, прегледа детаља о претходно одржаним догађајима, као и могућност евиденције плаћања у систему. Пошто су све наведене функционалности раније описане и приказане, нећемо их овде додатно разматрати.

Тиме је завршен преглед свих функционалности система. У наредном поглављу ће бити дат кратак осврт на цели систем апликације *GameTime*, као и виђење аутора како би систем могао даље да се побољша и надогради.

7. ЗАКЉУЧАК

Рекреативни спорт заслужује специјализоване алате који ће унапредити искуства учесника, организатора и администратора. У добу информационих технологија, постоје средства која могу да реше проблеме са којима се суочавају они који желе да учествују у спортским активностима. Апликација *GameTime* је резултат разумевања те чињенице. Она је израђена са циљем да обезбеди интуитивно и персонализовано искуство за сваког корисника, засновано према његовим индивидуалним потребама и интересовањима.

Суочен са изазовом да креира апликацију која је ефикасна и прилагођена потребама сваке категорије корисника, аутор је усвојио најновије технолошке стандарде и принципе дизајна. Коришћењем технологија као што је *MEAN stack*, уз интеграцију са другим моћним сервисима, направљен је робустан, скалабилан и једноставан систем за употребу.

С обзиром на актуелне трендове у свету мобилних апликација, еволуција *GameTime* могла би убрзо да укључи и адаптацију платформе за *Android* и *iOS* системе. Ово би значајно проширило доступност апликације и омогућило корисницима да уживају у њеној функционалности где год да се налазе.

Што се тиче платних трансакција, интеграција са савременим платним порталима или системима за онлајн плаћање могла би да претвори *GameTime* у комплетну платформу, на којој корисници не само да се пријављују за различите спортске активности, већ могу и да обаве плаћање учешћа без изласка из оквира апликације. Аутоматско вођење евиденције уз помоћ интеграције би значајно олакшало учешће и управљање приходима.

У будућности, могла би се разматрати интеграција с различитим платформама за друштвене мреже или чак увођење модула за вештачку интелигенцију, који би помогао у побољшању предлагања догађаја на основу интереса и претходних активности корисника.

Са аспекта тестирања, док је ручно тестирање кроз интеракцију са интерфејсом било неопходно током развојне фазе, постоји потреба да се у будућности размотри увођење аутоматских тестова. Овакво тестирање би обезбедило да свака нова функционалност или измена не утиче негативно на постојеће компоненте апликације.

Иако је коришћење *MongoDB* у локалном окружењу представљало солидан избор, могао би се размотрити и прелазак на *MongoDB Atlas* у будућности, ради боље скалабилности и повећане безбедности.

У крајњем случају, *GameTime* је више од само софтверске апликације. Он представља нови начин на који људи могу да учествују и уживају у рекреативним спортским активностима, обезбеђујући платформу која подстиче физичку активност, здрав живот и друштвено повезивање. Ова апликација има потенцијал да унапреди и промени начин на који се рекреативни спорт разуме и практикује.

ЛИТЕРАТУРА

- [1] *Meetup*, <https://www.meetup.com/> (посећено: 17.08.2023.)
- [2] *Playo*, <https://playo.co/> (посећено: 17.08.2023.)
- [3] *Timster*, <https://apkpure.com/timster-na%C4%91i-igra%C4%8De-za-fudbal/com.solaris.timster> (посећено: 17.08.2023.)
- [4] Преглед MEAN *stack* технологије, <https://www.mongodb.com/mean-stack> (посећено: 19.08.2023.)
- [5] Преглед *Node.js*, <https://www.tutorialandexample.com/node-js-tutorial> (посећено: 19.08.2023.)
- [6] Основни концепти *Node.js*, <https://www.geeksforgeeks.org/explain-the-working-of-node-js/> (посећено: 19.08.2023.)
- [7] Механизам рада *Node.js*, <https://www.tutorialandexample.com/node-js-event-loop> (посећено: 19.08.2023.)
- [8] Преглед npm, <https://docs.npmjs.com/about-npm> (посећено: 19.08.2023.)
- [9] Регистар пакета *Node.js*, <https://www.npmjs.com/> (посећено: 19.08.2023.)
- [10] Предавање о *Node.js* из предмета Програмирање интернет апликација, https://rti.etf.bg.ac.rs/rti/ir4pia/materijali/predavanja/si4pia_NodeJS.pdf (посећено: 19.08.2023.)
- [11] Преглед *Express.js*, <https://www.besanttechnologies.com/what-is-expressjs> (посећено: 20.08.2023.)
- [12] *Express* рутирање, <https://javascript.plainenglish.io/lets-build-apis-with-node-express-andpostgresql-3be160d50519> (посећено: 20.08.2023.)
- [13] *Express middleware*, <https://iq.opengenus.org/middlewares-in-express/> (посећено: 20.08.2023.)
- [14] *MongoDB* сајт, <https://www.mongodb.com/> (посећено: 21.08.2023.)
- [15] Преглед JSON и BSON, <https://www.mongodb.com/json-and-bson> (посећено: 21.08.2023.)

- [16] Скалабилност у *MongoDB*, <https://www.mongodb.com/basics/scaling> (посећено: 21.08.2023.)
- [17] *Angular.io* сајт, <https://angular.io/> (посећено: 21.08.2023.)
- [18] Предавање о *Angular* из предмета Програмирање интернет апликација, https://rti.etf.bg.ac.rs/rti/ir4pia/materijali/predavanja/si4pia_Angular7-P1-P3.pdf (посећено: 23.08.2023.)
- [19] *bcrypt* пакет, <https://www.npmjs.com/package/bcrypt> (посећено: 25.08.2023.)
- [20] Документација за пријаву са *Google*, <https://developers.google.com/identity/gsi/web/guides/overview> (посећено: 26.08.2023.)
- [21] Документација за верификовање *Google* токена, <https://developers.google.com/identity/gsi/web/guides/verify-google-id-token> (посећено: 26.08.2023.)
- [22] Документација за рад са *Google* мапама, <https://developers.google.com/maps> (посећено: 27.08.2023.)
- [23] *Haversine* формула, <https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128> (посећено: 27.08.2023.)

СПИСАК СКРАЋЕНИЦА

API - *Application Programming Interface*

AJAX - *Asynchronous JavaScript and XML*

BSON - *Binary JSON*

CORS - *Cross-Origin Resource Sharing*

CSS - *Cascading Style Sheets*

DOM - *Document Object Model*

HTML - *HyperText Markup Language*

HTTP - *Hypertext Transfer Protocol*

ID - *идентификатор*

iOS - *iPhone Operating System*

JSON - *JavaScript Object Notation*

JWT - *JSON Web Token*

MEAN - *MongoDB, Express, Angular, Node*

MVVM - *Model-View-ViewModel*

npm - *node package manager*

tsc - *TypeScript compiler*

URL - *Uniform Resource Locator*

USD - *United States dollar*

СПИСАК СЛИКА

Слика 2.1.1. Почетни екран <i>Meetup</i> апликације.....	7
Слика 2.2.1. Креирање новог догађаја у апликацији <i>Playo</i>	8
Слика 2.3.1. Преглед локације у апликацији <i>Timster</i>	9
Слика 3.1.1.1 <i>Node.js</i> систем [7].....	13
Слика 3.2.1. Процес рутирања и обраде HTTP захтева [12]	16
Слика 3.2.2. Ланчана обрада захтева помоћу <i>middleware</i> функција [13].....	16
Слика 3.4.1.1. MVVM архитектура [18].....	19
Слика 3.4.9.1. Шема MEAN радног оквира [4]	21
Слика 5.1.1. Архитектура апликације <i>GameTime</i>	33
Слика 5.2.1. Модел базе података	36
Слика 5.3.4.1. Пример мејла подсетника	46
Слика 6.1. Почетни екран апликације(форма за пријаву).....	52
Слика 6.2. Форма за регистрацију	53
Слика 6.1.1. Почетни екран учесника	54
Слика 6.1.2. Приказ детаља о актуелном догађају.....	54
Слика 6.1.3. Приказ локације догађаја.....	55
Слика 6.1.4. Преглед коментара и форма за коментарисање.....	55
Слика 6.1.5. Преглед профила организатора.....	56
Слика 6.1.6. Преглед свог профила	56
Слика 6.1.7. Преглед претходних учешћа	57
Слика 6.1.8. Форма за ажурирање профила учесника	57
Слика 6.2.1. Приказ актуелног догађаја за организатора.....	58
Слика 6.2.2. Приказ пријављених учесника за догађај.....	59
Слика 6.2.3. Приказ профила учесника организатору	59
Слика 6.2.4. Форма за додавање новог догађаја	60
Слика 6.2.5. <i>Autocomplete</i> функционалност.....	60
Слика 6.2.6. Евидентирање плаћања	61
Слика 6.2.7. Мејл обавештења о новом догађају	61
Слика 6.3.1. Преглед свих учесника система	62
Слика 6.3.2. Преглед свих организатора система	62
Слика 6.3.3. Преглед корисничког профила са опцијом брисања.....	63

СПИСАК ТАБЕЛА

Табела 3.3.1. Упоредна анализа карактеристика постојећих решења	10
---	----

СПИСАК ДЕЛОВА ПРОГРАМСКОГ КОДА

Део кода 3.1.4.1. Пример JSON формата	15
Део кода 5.3.1.1. Иницијализација сервера	38
Део кода 5.3.2.1. Хеширање лозинке	39
Део кода 5.3.2.2. Провера креденцијала и издавање JWT токена	40
Део кода 5.3.2.3. Позив методе сервиса <i>userService</i>	41
Део кода 5.3.2.4. Слање HTTP захтева са заглављем	41
Део кода 5.3.2.5. Рутирање захтева	42
Део кода 5.3.2.6. Верификација токена	42
Део кода 5.3.2.7. Обрада захтева за брисање корисника	42
Део кода 5.3.3.1. Укључивање <i>Google Sign-In JavaScript</i> библиотеке	43
Део кода 5.3.3.2. Иницијализација дугмета за пријаву преко <i>Google</i>	43
Део кода 5.3.3.3. Обрада <i>Google</i> токена на серверу	44
Део кода 5.3.4.1. Аутоматизација система	45
Део кода 5.3.4.2. Помоћна метода за слање мејлова	45
Део кода 5.3.4.3. Методе за обраду актуелних догађаја	46
Део кода 5.3.5.1. Приказ локације на мапи	48
Део кода 5.3.5.2. Омогућавање <i>autocomplete</i> функционалности	49
Део кода 5.3.5.3. Дохватање тренутне локације корисника	49
Део кода 5.3.5.4. Филтрирање догађаја у дефинисаном радијусу	50
Део кода 5.3.5.5. Рачунање удаљености између две локације	50
Део кода 5.3.6.1. Враћање профилне слике корисника клијентској страни	51