

Съдържание:

Увод.....	3
I глава. Литературен преглед.....	4
1.1 Платформа за резервации „Tereni.bg“.....	4
1.2 Платформа за резервации „easybook.bg“.....	5
1.3 Платформа за резервации „Sport4All.bg“.....	6
1.4 Онлайн система за резервация на автобусни билети.....	6
1.5 Онлайн система за болнични резервации „Hospital Reservation System – HRS“.....	9
1.6 Мобилно приложение за резервация на автобусни билети.....	12
1.7 Приложение за онлайн резервации на незасти билети за влакове.....	14
1.8 Изводи за разгледаните до момента платформи.....	16
1.9 Функционалности и предимства на разработвания дипломен проект.....	17
II глава. Анализ и описание на използваните технологии.....	19
2.1 Програмен език „Java“. Комуникация „Модел – Изглед – Управление“ („Model-View-Controller“). Архитектурен стил „Репрезентационен трансфер на състояния“ - „REST“.....	19
2.1.1 Комуникация „Модел – Изглед – Управление“ - „Model-View-Controller“/“MVC“.....	19
2.1.2 Архитектурен стил „Репрезентационен трансфер на състояния“ – „REST“.....	20
2.2 Технологична рамка с отворен код „Spring Framework“.....	21
2.3 Базы данни. Релационни бази данни.....	24
2.3.1 Релационни бази данни.....	24
2.4 Система за управление на релационни бази данни „MySQL“.....	25
2.5 Технологична рамка „Hibernate ORM“.....	25
2.5.1 Архитектура на технологичната рамка „Hibernate“	25
2.6 Среда за разработка на приложения „Eclipse Integrated Development Environment“.....	27

2.6 Платформа с отворен код „Postman“.....	28
III глава. Архитектура на програмната реализация.....	29
3.1 Основна архитектура на приложението.....	29
3.2 Бизнес логика на приложението.....	31
3.2.1 Обекти (Entities).....	31
3.2.2 Хранилища (Repositories).....	32
3.2.3 Услуги (Services).....	33
3.2.4 Управление (Controllers).....	43
3.2.5 Защита на приложението и нейната конфигурация.....	48
3.2.6 Файл с настройки на приложението „application.properties“	53
3.2.7 Диаграма на връзките между обектите (ER – Entity Relationship diagram).....	54
IV глава. Експериментални данни и изводи.....	55
4.1 Експериментални данни.....	55
4.2 Изводи.....	60
4.3 Варианти за бъдеща оптимизация.....	61
Използвани литературни източници.....	62
Списък на използваните съкращения.....	64

Увод

Целта на тази дипломна работа е разработката на уеб приложение за резервация на спортни съоръжения. Платформите за резервации от този тип са доста популярни днес, тъй като улесняват и спестяват време на своите потребители.

Дипломната реализация ще представлява бек-енд частта на приложението, като за целта ще бъдат използвани Java, Spring Boot, Hibernate и MySQL.

В първа глава предстои да бъдат разгледани различни платформи за резервации, принципът на работа на които е подобен на този, по който трябва да функционира приложението. Ще бъде направен анализ на техните недостатъци и ще бъдат описани основните функции, които разработваната система трябва да притежава, както и кои от описаните недостатъци ще подобри тя.

Във втора глава ще бъдат подробно разгледани избраните за разработка на дипломния проект технологии и софтуерни продукти, техните характеристики и функции.

Подробно ще бъде разгледана архитектурата на програмната реализация, основните класове, методи, функционалности и връзките между тях – в глава три.

Накрая, в четвърта глава, ще бъдат направени изводи за създаденото приложение, до каква степен е изпълнена зададената функционалност и какви бъдещи оптимизации биха могли да се направят, които да подобрят и надградят неговите възможности.

I глава. Литературен преглед

От години в множество сфери се използват програми и автоматизации за резервиране на различни услуги – билети за транспорт, различни развлечения и спортове, резервации в хотели, запазване на час за разнообразни услуги и много други. Днес приложенията от подобен тип са изключително разпространени и неразделна част от нашия живот. Подобни платформи значително улесняват както своите клиенти, така и служителите, спестявайки време, иначе прекарвано в чакане на опашки и телефонни разговори.

Приложенията за резервации за спорни съоръжения улесняват своите потребители както в запазването на час, така и в търсенето на подходящ обект. Те дават подробна информация за наличните опции в различни квартали, градове, данни за техните размери, предлагани услуги, сравнение на цените и други. В момента на българския пазар има разработени няколко платформи за резервация от подобен тип – това са tereni.bg – посветена изцяло на резервация на футболни игрища в София, easybook.bg – за резервация на разнообразни услуги в страната, sport4all.bg – за резервация на спорни съоръжения за град Варна. Основните им функции и възможности ще бъдат разгледани в следващите параграфи.

Освен гореспоменатите приложения ще бъдат разгледани и още няколко системи – три за резервация на различен тип билети, и една за запазване на часове за преглед в болница. Техният принцип на работа и функционалности силно се припокриват с начинът, по-който трябва да функционира приложението за резервации на спортни съоръжения.

Накрая на тази глава ще бъде описано и с какво текущата дипломна работа цели да подобри възможностите на вече разгледаните приложения и платформи.

1.1 Платформа за резервации „Tereni.bg“

„Tereni.bg“ е платформа за резервация на футболни игрища, намиращи се в град София. Предоставя основна информация за това дали обектът е закрит или открит, каква е настилката, какви са неговите размери, каква е цената за резервация. Разполага със снимки на обектите, визуализация на тяхната локация чрез [google maps](https://www.google.com/maps) и график на свободните и заетите часове за текущата седмица. Предлага и допълнителна информация като наличие на паркинг, фитнес, интернет, кафене и др. Резервациите в платформата стават

изцяло онлайн, а начинът на плащане се осъществява на място на самото игрище.

Един от основните минуси на платформата е, че за момента работи с обекти, разположени само в София. Като друг минус може да се отчете и липсата на възможност за регистрация на потребителите. [1]

По време на създаване на дипломното задание платформата беше свалена и вече не функционира. Информацията за нея е все пак е включена като пример какво трябва да съдържа едно приложение за резервации на спортни съоръжения.

1.2 Платформа за резервации „easybook.bg“

Това е платформа за резервации с доста широк обхват. Включва обекти, свързани с красота, здраве, поддръжка на автомобили, на дома, както и такива, свързани със спорт и здраве. За разлика от „tereni.bg“, които са фокусирани само върху футболни игрища тази платформа предлага резервации за най-различни видове тренировки и спортове. Неин плюс е и че работи с обекти от цялата страна.

За всеки бизнес е предоставена снимка, работно време за текущият ден, адрес и бърз достъп до неговата локация чрез „Google maps“, информация за цената на час, свободни места, капацитет и график със свободните часове. Според типа на обекта се предлага и само възможност за закупуване на абонаментна карта.

Конкретно за футболни игрища се предоставя информация, включваща каква е неговата настилка, размерите и конструкцията му, какви са условията. Дава се информация за цената за резервация за час, и свободните часове.

Платформата разполага с възможност за регистрация, а резервации могат да се извършват само от потребители, влезли в профила си. Лог-ин формата разполага с опции за запомняне на паролата и линк за възстановяване на забравена такава. Плюс на платформата е, че освен стандартна регистрация позволява и влизане чрез вече съществуващи „Facebook“ и „Apple“ профили.

Резервацията в платформата е обвързана с непосредствено онлайн плащане, като потребителят има възможност да направи транзакцията чрез „Apple Pay“, „Google Pay“, както и чрез „Revolut“ и „Visa“ карти. След

избиране на желаното игрище, дата и час за резервация клиентът разполага с 30 минути за осъществяване на плащането. [2]

1.3 Платформа за резервации „Sport4All.bg“

Тази платформа е фокусирана върху спортни обекти във Варна и е разработена по общински проект.

За всеки спортен комплекс е дадена основна информация – къде се намира, локацията му в „Google maps“, какви обекти включва – игрища, зали, какво е работното му време, цена за час. Тук плюс е, че е предоставен директен контакт с отговорното лице за комплекса, както и че е налична информация кога част от обектите могат да се използват безплатно. Минус е обаче липсата на възможност да се направи резервация за безплатните часове.

За всеки комплекс наличните игрища и зали са разделени в отделни секции, съдържащи размерите на конкретния терен, работното му време, цени или безплатен диапазон, както и друга допълнителна информация.

Регистрацията на нови потребители се извършва само чрез формата на сайта, след което всеки потребител има възможността да редактира данните, които е въвел.

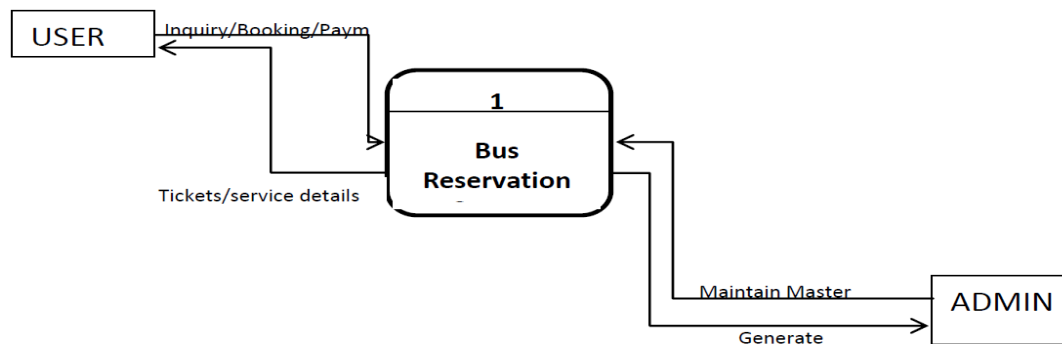
Платформата, както и предходните разполага с график на свободните и заети часове, а резервациите се извършват само от потребители, влезли в профила си. Плащането се извършва чрез кредитна или дебитна карта и няма опция да се извърши на място. След успешно извършено плащане системата генерира билет, който потребителят да представи при явяване в обекта. [3]

1.4 Онлайн система за резервация на автобусни билети

Проектът е уеб базиран, като фронт-енд частта на разглеждания софтуер е създадена с помощта на програмния език „PHP“, а за бек-енд частта е използвана системата „MySQL“. Функционалностите, с които разполага ще бъдат разгледани на няколко нива.

Основните взаимодействия в системата са представени на Фигура 1.1:

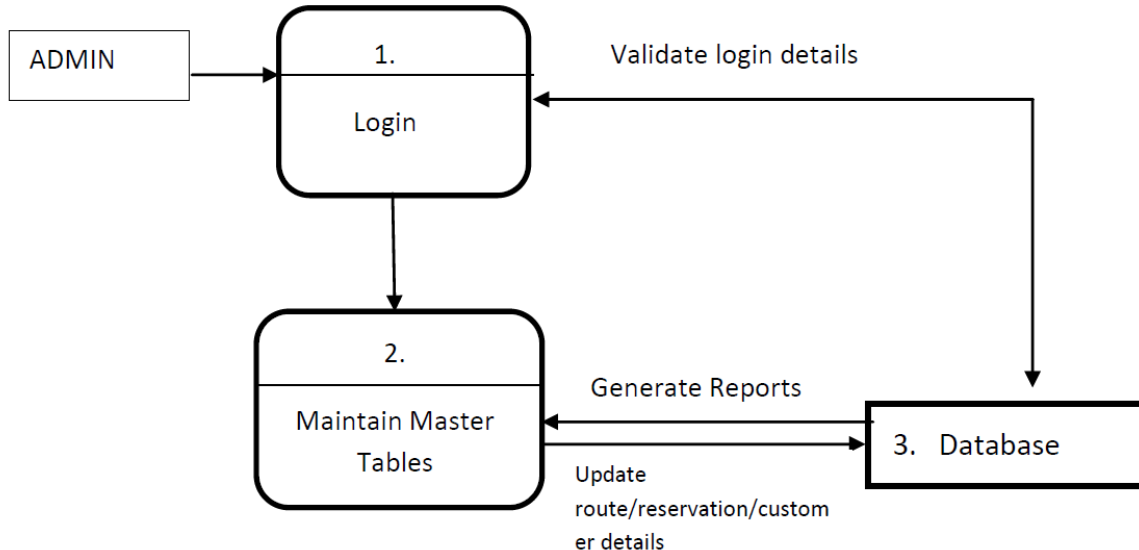
Level 0



Фигура 1.1 Основни взаимодействия и операции в платформата, [4]

Потребителят има възможност да прави запитвания, да резервира, да прави плащания. От своя страна приложението връща информация на потребителя и генерира билети. Имаме администратор, който управлява системата, показан на фигура 1.2.

LEVEL 1

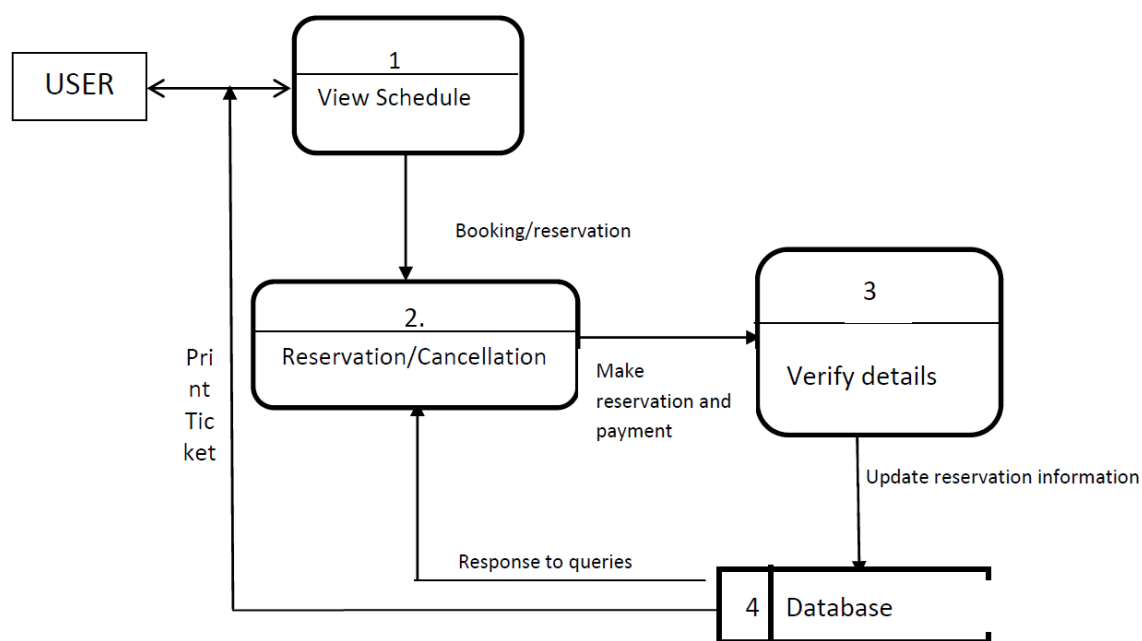


Фигура 1.2 Администраторски функции в системата, [4]

Администраторският профил има възможност за лог-ин в приложението, а базата данни валидира неговото потребителско име и парола. След успешен вход, администраторът има възможност да променя маршрути, резервации, както и информация за клиентите.

Възможностите на потребителя са показани на фигура 1.3:

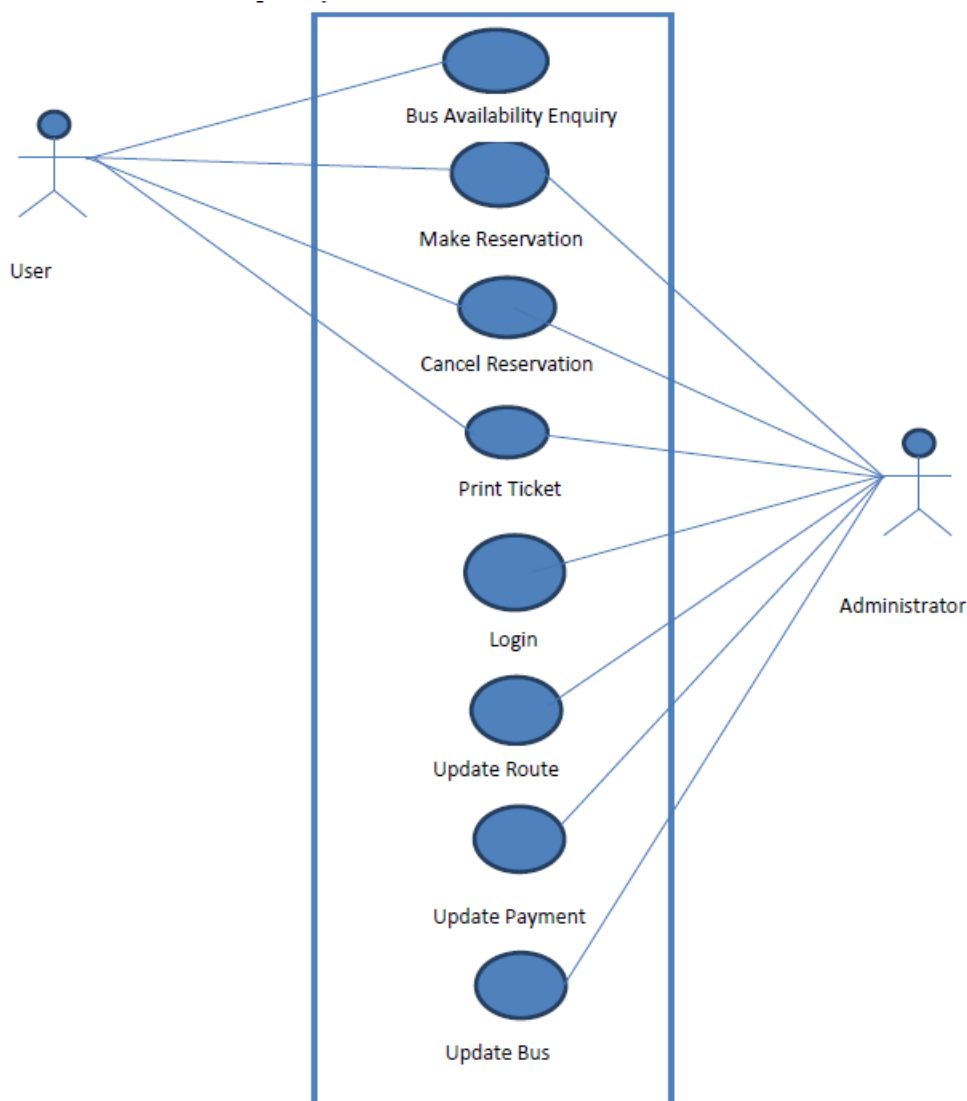
Level 2



Фигура 1.3 Потребителски функции в системата, [4]

За разлика от администратора, потребителите нямат нужда от вход в свой профил. Те директно имат достъп до разписанията на автобусите. Могат да правят резервации и да ги отменят, като по време на тези операции си взаимодействат с базата данни, която при резервация им връща билет за принтиране, а при отмяна обновява данните си. Всеки билет, генериран от базата, съдържа идентификационния номер на транзакцията, име, адрес и телефон на клиента, както и данни за курса, номера на мястото и часът на търгване на автобуса.

Следната диаграма на потребителските случаи - „use-case“ обобщава функциите, както на потребителя, така и на администратора и показва кои от тях се припокриват:



Фигура 1.4 Диаграма на потребителските случаи за системата – „Use-case“, [4]

Като минуси на тази система може да се отчете липсата на вход и профил за потребителите, наличието на които би позволило да се поддържа постоянна информация за клиента, както и история на неговите резервации. [4]

1.5 Онлайн система за болнични резервации „Hospital Reservation System – HRS“

Тази система е създадена за „Андроид“, а основната ѝ идея е да осигури възможност на пациентите онлайн да запазят час за преглед, улеснявайки както тях, така и лекарите в болницата.

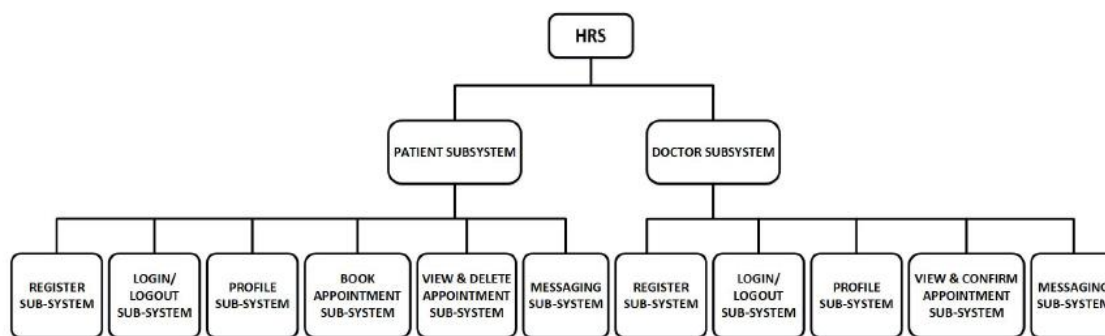
Приложението разполага с панели както за клиентите, така и за медицински лица. Всички имат възможност да се регистрират, като всеки лекар въвежда своя идентификационен номер, който се използва при влизане в системата.

Панелът за пациенти се състои от следните секции – профил, секция за резервиране на час, секция за преглед и отказ от резервация, секция за съобщения, в която пациентът получава потвърждения от лекаря за записан час, изход от профила.

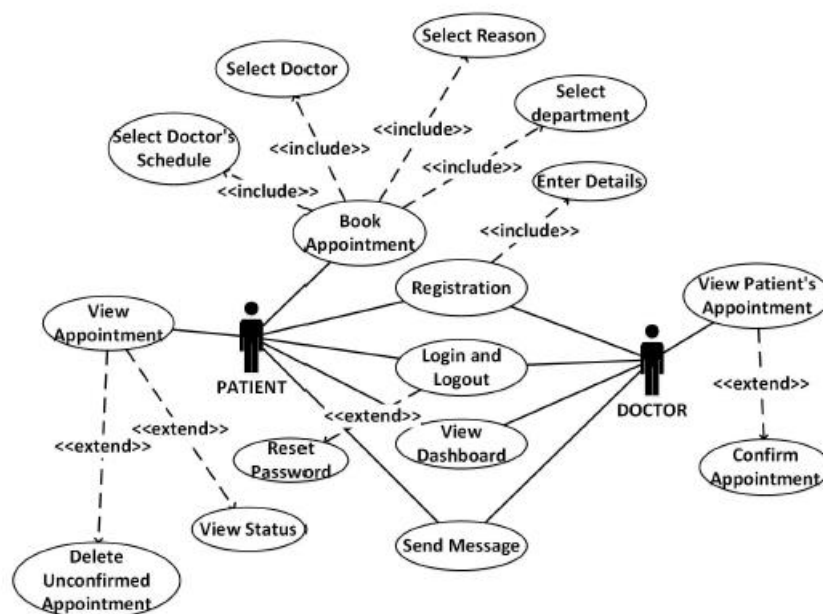
Панелът за лекари от своя страна се състои от профил, секция за преглед и потвърждаване на заявки за записване на час, платформа за съобщения и възможност за изход от профила.

Комуникацията се осъществява чрез уеб сървър и система за „Андроид“. Сървърът съхранява информацията за пациентите и лекарите в база данни. Приложението извлича данни от базата под формата на „JSON“ (JavaScript Object Notation – „JavaScript“ обектна нотация) обекти чрез „PHP“ код, съдържащ „MySQL“ заявки. „JSON“ обектите се извличат в „Java“ класове за да се презентират на потребителите.

На следващите фигури могат да се видят диаграмата на потребителските случаи – „use-case“, както и модулите, от които се състои системата:



Фигура 1.5 Модули, от които се състои системата, [5]



Фигура 1.6 Диаграма на потребителските случаи на системата – „Use-case“, [5]

И пациентите, и медицинските лица трябва да се регистрират в приложението за да получат достъп до функциите му. Въвеждат се стандартни данни като потребителско име, парола, имена, имейл, телефон, дата на раждане, както и пол. Разликата между двете регистрации е наличието на уникален идентификационен код за всеки лекар, както и отделението, в което работи. След регистрацията се генерира съобщение дали тя е била успешна или не.

За влизане в системата се изисква потребителско име и парола и отново се генерира съобщение за успешен/неуспешен вход.

След като са влезли, първото, което се зарежда за всички потребители е техният профил, съдържащ основната информация, въведена от тях при регистрацията.

При резервация на час за преглед всеки пациент въвежда причина за запис на часа, избира отделение, името на желанния лекар, ден и час за преглед от падащи менюта. Като при избор на отделение от падащото меню, автоматично се генерират лекарите, работещи в него, а при вече избрано лице се генерират свободните за него дати и часове, улеснявайки потребителя.

При успешна регистрация се генерира потвърдително съобщение за клиента, както и му се изпраща смс.

Пациентите имат и възможност за преглед и изтриване на направените резервации. Информацията за тях е под формата на таблица, съдържаща идентификационен номер на резервацията, името на лекаря, на отделениято, дата, час, причина за запис на часа, статус, стойност на прегледа и бутон за изтриване. Като дадена резервация може да се изтрие само ако е със статус „В очакване“ и все още не е потвърдена от съответния лекар. След изтриване до потребителя се генерира съобщение успешна или неуспешна е била операцията.

От своя страна лекарите разполагат с таблица съдържаща направените резервации за часове. Тя съдържа идентификационен номер на резервацията, име и телефон на пациента, причина за запазения час, статус и бутон за потвърждение. При потвърждение на заявения час се генерира съобщение, че операцията е успешна, статусът на резервацията се сменя на „Потвърдена“, а пациентът вече няма възможност да я изтрие.

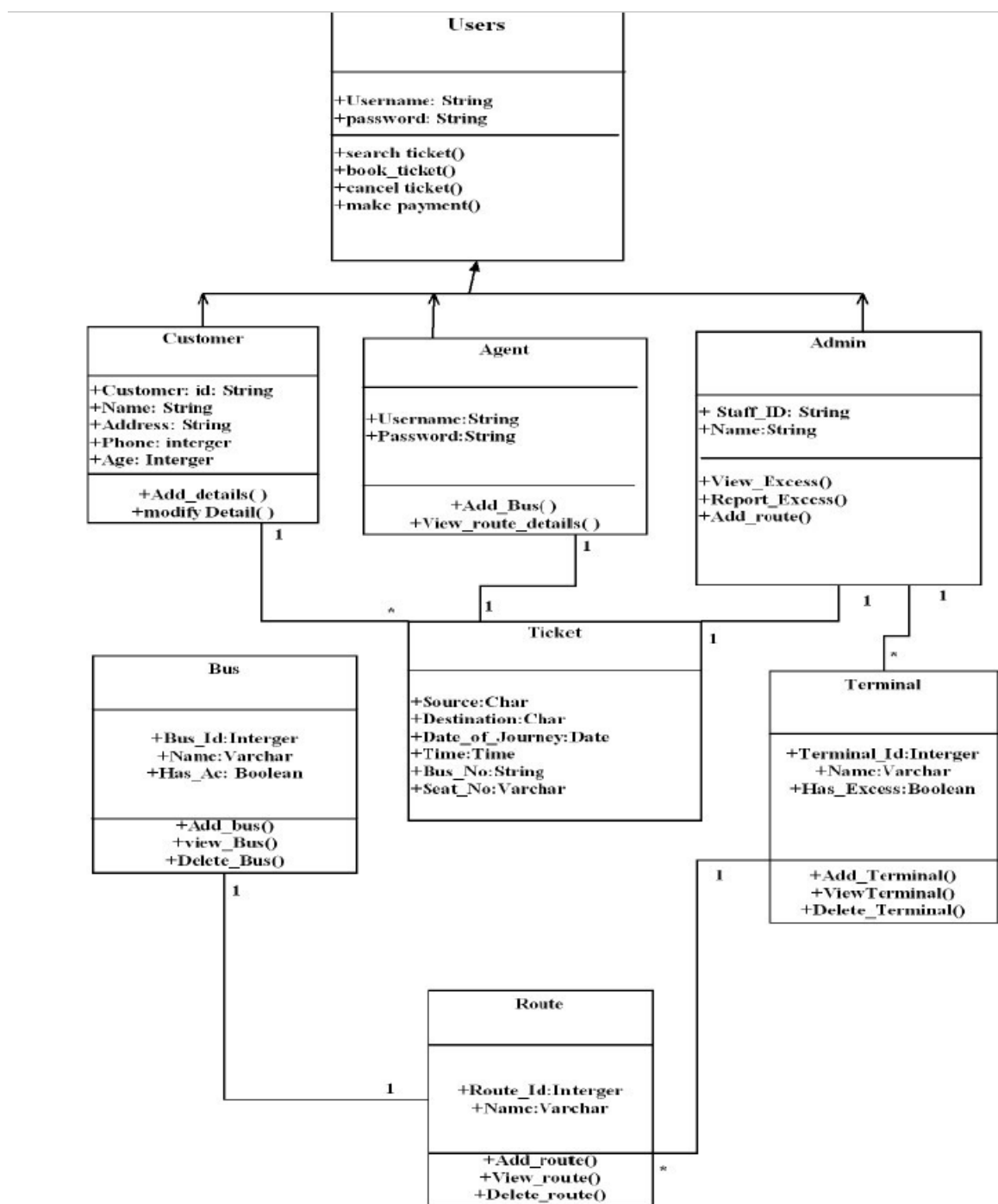
Приложението разполага и с възможност за комуникация между пациенти и лекари, като при изпращане на съобщение се получават и известия. [5]

1.6 Мобилно приложение за резервация на автобусни билети

Основната цел на тази система е възможността за резервация на автобусни билети, като освен нея се използва и алгоритъм за разпределение на автобусите, така че да се постигне максимална ефективност и за клиентите и за транспортните фирми. Неин плюс е и, че може да се използва от различни фирми, не е разработена за конкретна такава. Приложението е създадено, чрез използване на програмните езици „PHP“, „JavaScript“, „HTML“ (HyperText Markup Language - Език за маркиране на хипертекст) и системата „MySQL“.

Използва се „Round Robin“ алгоритъм, който на определен интервал проверява капацитета на автобусите, които скоро трябва да тръгнат. Ако е достигнат половината или повече от техния капацитет се насрочва да потеглят в посоченият им час. В противен случай, при малък брой пътници те се преразпределят към следващият автобус за същия маршрут и получават съобщение, генерирано от системата, че има промяна. „Round Robin“ е избран заради предимствата му спрямо други алгоритми за резервации – лесен за имплементация, решава проблема със „гладуването“, при който дадена заявка не може да ползва процесора, защото той е ангажиран от други, смятани за по-важни, дава възможност за едновременното правилно разпределение на ресурсите между автобусите и осигуряване на удовлетвореност на клиентите.

Клас диаграмата и връзките между отделните класове на приложението са показани на фигура 1.7:



Фигура 1.7 Клас диаграма на приложението, [6]

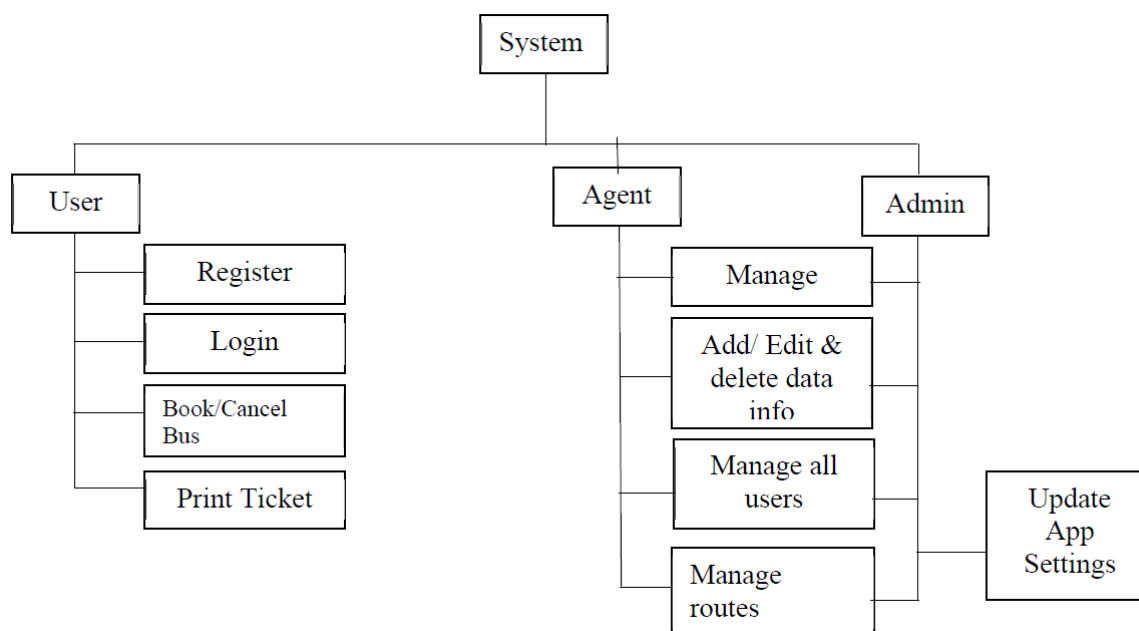
Приложението се състои от основен клас Потребители – „Users“, който съдържа методи даващи възможност за преглед на билети, резервация на билет, отказ от вече направена такава и извършване на плащане.

Този клас наследяват Клиент – „Customer“, Агент – „Agent“ и администратор - „Admin“. Клиентът има идентификационен номер, може да

добавя и променя данните, които е въвел за себе си. Основната функция на Агента е да добавя автобуси, и да има достъп до информацията за техните маршрути. Администраторът може да добавя нови маршрути и да управлява системата. Всички влизат в приложението чрез потребителско име и парола.

Автобусите, билетите, маршрутите и терминалите са обособени в свои собствени класове. [6, 7]

Архитектура на системата може да се види на следващата фигура:



Фигура 1.8 Архитектура на системата, [6]

1.7 Приложение за онлайн резервации на незаети билети за влакове

Това приложение е „Андроид“ – базирано и е създадено за „Индийски железници“, с цел улесняване резервацията на билети.

За да го използват, потребителите трябва да си направят регистрация, която се записва в базата данни. Запазва се информация за потребителското име, парола, дата на раждане, уникален идентификационен номер, пол, мобилен телефон, както и имейл.

Системата има отделна услуга – Кондуктор „Ticket Collector“, който вписвайки се в платформата, може да въведе номера на влака. След подаването на номера се генерира списък на всички пътници, които ще пътуват в този влак.

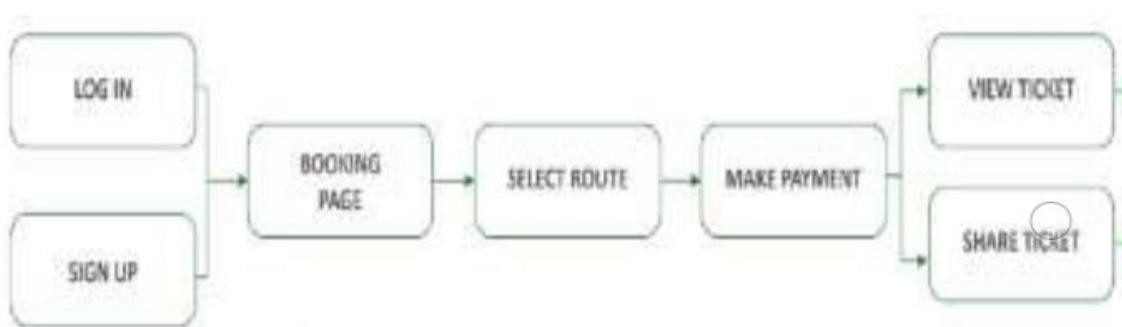
След като запазят и успешно платят своя билет, потребителите получават потвърждение за направената операция. Приложението поддържа история на покупките под формата на код за бърз отговор – „Quick Response – QR“ код, в който са кодирани потребителското име, дата на пътуването, точка на тръгване и крайна дестинация. Въпросният код може да бъде сканиран от контролните органи, проверяващи билетите, за да бъде потвърдено, че е направена покупка. Приложението разполага и с функция за пращане на имейл за потвърждение на транзакцията, съдържащ текстови файл с релевантната информация за клиента и пътуването.

Друга удобна функция на приложението е интеграцията на „Google maps“, позволяваща клиента да бъде навигиран от текущата му локация до гарата, от която трябва да замине.

Приложението използва трислойна архитектура:

- **Слой база данни „Database layer“** – в случая е използвана база данни, запазваща информацията в „JSON“ формат.
- **Слой Услуги „Application Service layer“** – тук са разпределени системните функции и бизнес логиката на приложението. Слойът отговаря също и за промени по базата данни, според заявките от най-горно ниво.
- **Слой Клиентски интерфейс „Customer Interface layer“**

Диаграмата на взаимодействия на приложението е представена на следващата графика:



Фигура 1.9 Диаграма на взаимодействия на системата, [8]

Приложението се състои от следните функции:

- **Регистрация** - При първото си посещение в приложението потребителите следва да се регистрират, предоставяйки своите данни.;
- **Вход** – Посредством потребителско име и парола. Потребителят има и възможност за възстановяване на паролата, ако я забрави. При този случай предоставя имейлът, с който се е регистрирал, а на него се изпраща линк за избор на нова парола. Промяната се отразява директно и в базата данни.;
- **Избор на маршрут** – След успешна автентификация потребителят може да избере следните данни – начална станция, крайна станция, брой пътуващи.;
- **Проверка на свободни влакове** – Визуализира се списък на влаковете с маршрут, минаващ през избраните от потребителя гари, както и в колко часа тръгва и пристига всеки от тях.;
- **Резервация на билети** – Потребителят задава имената и възрастта на всеки от пътниците, за които запазва билети.;
- **Извършване на плащане** – Използва се външна система, която да удостовери електронното плащане.;
- **Потвърждение на транзакцията** – при успешно плащане приложението предоставя страница с потвърждение, съдържаща потребителско име, час, точка на тръгване и дестинация. Тази информация е кодирана в код за бърз отговор – „QR“ код, видим в приложението, като към потребителя се изпраща и имейл с потвърждение. [8]

1.8 Изводи за разгледаните до момента платформи

В предходните параграфи разгледахме различни софтуерни проекти, чиято идея е резервацията на определен продукт или услуга. Всеки един от тях освен плюсове има и своите недостатъци, които следва да разгледаме тук.

Сред разгледаните български платформи за резервации на спортни съоразения като основен минус може да се посочи обвързването на резервацията с непосредствено плащане. Не всеки потребител ще има желание да извърши такова, а при евентуална отмяна на резервацията целият процес се усложнява поради необходимостта от връщане на платената вече сума.

Друг минус е, че две от платформите – „tereni.bg“ и „Sport4All.bg“ са ограничени за конкретен град – София и Варна. Като редовно трениращ

потребител, използващ приложението в конкретен град, би било хубаво да имаш опцията ако отидеш на почивка в друг да можеш да разгледаш наличните игрища там и да си направиш резервация. По-глобалният обхват на приложението би го направило и доста по-популярно.

Голям минус на разгледаната в точка 1.4 онлайн система за резервация на автобусни билети е, че изисква вход само за администратора. Липсата на регистрация и вход за клиентите прави невъзможна поддръжката на трайна информация за тях, както и на история на направените от тях резервации.

Платформата за болнични резервации от точка 1.5 е доста добре разработена, но като негатив може да се отчете невъзможността за отмяна на резервация след нейното потвърждение от лекар. Винаги могат да възникнат непредвидени обстоятелства и по-добрият вариант е резервацията да може да се отмени, а лекарят, на който е била причислена да получи известие за това.

Недостатък на системата за резервация на автобусни билети от точка 1.6 е липсата на история за извършени плащания.

Приложението от точка 1.7 за резервация на автобусни билети пък дава достъп до различните маршрути чак след задължителен вход/регистрация на потребителя. Въпреки, че е добре самата регистрация да се извърши от влезли в профила си потребители, за тях ще е много по-лесно да могат да изберат желанят маршрут и да видят какви влакове минават през него, без да се регистрират специално за това.

На база на изброените негативи на описаните системи, в следващите параграфи следва да разгледаме какво ще представлява приложението за резервации на спортни съоразения и кои от разгледаните минуси ще подобри.

1.9 Функционалности и предимства на разработвания дипломен проект

Позовавайки се на разгледаните до момента софтуерни продукти следва да оформим какви възможности трябва да има разработваната дипломна работа.

Част от подобренията на изложените досега недостатъци на останалите платформи ще бъдат следните – в приложението, ще могат да се извършват свободно резервации, без те да са обвързани с непосредствено плащане, както и да се отменят без ограничения. Спортните обекти, които се добавят, няма да са ограничени за конкретен град, могат да са разположени в цялата

страна. Потребителите ще имат свободен достъп до информацията за спортните обекти без нужда от регистрация, а приложението ще бъде напълно безплатно. Всеки потребител, след вход в приложението ще има достъп до резервациите, които е правил, което липсва като функция в част от разгледаните платформи.

Ще бъде разработена бек-енд частта на приложение за резервация на спортни игрища, като то ще е уеб базирано. Ще се използват технологиите „Java“, „Spring Boot“, релационна база данни.

Потребителите на приложението ще са разделени на три основни типа – клиенти, фирми и администратор. Ще се използва основен клас Потребител - „User“, с атрибути потребителско име и парола, и възможност за резервация на игрище и нейната отмяна. Типовете потребители ще се разграничават посредством роли, задавани на всеки от тях при регистрация. Всички ще имат възможност за вход в системата посредством логин формата на технологичната рамка „Spring Security“.

Клиентите свободно ще могат да разглеждат качените в приложението спортни игрища и информацията, свързана с тях. Ще разполагат и с функция да видят игрищата само за конкретен град, за да могат по-лесно да се ориентират или да филтрират игрищата по техния вид – футболни, волейболни и други . За да направят резервация обаче, ще трябва да са регистрирани и влезли в профила си. Ще имат възможност да разглеждат и отменят вече направени резервации, както и да редактират данните, които са въвели за себе си.

Фирмите, влизайки в профила си ще имат възможност да добавят свои спортни игрища, както и да редактират техните параметри и информация. Ще имат и възможност да променят собствените си данни като адрес, телефон и др.

Администраторският профил ще има възможността да добавя нови админи, да вижда пълен списък на регистрираните потребители според тяхната роля, както и да трие клиентски профили и спортни игрища.

В следващите глави предстои да бъдат описани и разгледани използваните технологии за разработка на дипломния проект, както и неговата архитектура реализация.

II глава. Анализ и описание на използваните технологии

В тази глава следва да бъдат разгледани използваните технологии и софтуерни продукти за разработка на дипломното задание. Ще бъде използван обектно-ориентираният език „Java“, както и технологичната рамка „Spring“. Ще се използва релационна база данни, а за връзка между приложението и базата, ще се използва „Hibernate“. За регистрация и вход на потребителите ще бъде използвана технологичната рамка „Spring Security“. Основните характеристики на всяка от тези технологии ще бъдат разгледани в следващите параграфи.

2.1 Програмен език „Java“. Комуникация „Модел – Изглед – Управление“ („Model-View-Controller“). Архитектурен стил „Репрезентационен трансфер на състояния“ - „REST“.

Java е широко използван обектно-ориентиран език от високо ниво, създаден с идеята кодът да се напише веднъж и да може да бъде изпълняван навсякъде. Компилираният код може да се изпълнява на всяка машина, поддържаща Java, без да има нужда да се рекомпилира. За целта програмите се компилират до байткод, подобен на машинен код, но създаден да се изпълнява от виртуална машина. Той може бъде пуснат на всяка Java Virtual Machine (JVM), независимо от конкретната компютърна архитектура, което го прави много универсален.[9-10] Тази универсалност и широка разпространеност на езика е и една от основните причини да бъде избран за разработка на приложението.

2.1.1 Комуникация „Модел – Изглед – Управление“ - „Model-View-Controller“/“MVC“

Модел – Изглед – Управление – „MVC“ е често използван модел за дизайн и разработка на приложения. Чрез него бизнес логиката се разделя от потребителския интерфейс – „UI“ („User Interface“) като се разпределят роли на модел, изглед и управление в приложението. Основната идея е чрез разделянето на логиката и интерфейса, те да могат да бъдат променяни без с тези промени да влияят един на друг.

- Модел „Model“

Моделът е отговорен за капсулиране на данните на приложението, той е базата данни, която използваме и е стабилен във времето. Не трябва да има пряка връзка между модела и изгледите, това ще означава директна връзка между потребителя и базата данни.

- Изглед „View“

View представлява интерфейсът, с който си служи потребителя. Може да бъде графичен, но не е задължително. Неговата роля е единствено да презентира данните, тук не присъства никаква бизнес логика.

- Управление „Controller“

Контролерът отговаря за получаване и обработка на заявки от потребителя и осъществява връзката между бизнес логиката и базата данни.

- Комуникация между трите компонента

Моделът не трябва да знае за съществуващите изгледи. Комуникацията между управлението, модела и изгледите се осъществява чрез събития, често идващи от управлението. Събитията осигуряват механизми за комуникация с минимални зависимости. Изгледите получават известие за събитие като клик на мишка или необходимо съдържание и регистрират тези, които трябва да обработят. [11-13]

2.1.2 Архитектурен стил „Репрезентационен трансфер на състояния“ – „REST“

Репрезентационният трансфер на състояния („Representational State Transfer“ - „REST“) е клиент – сървърен архитектурен стил, при който клиентът изпраща заявка до сървъра, той я обработва и връща отговор. Заявките и отговорите са построени около трансфера на репрезентации на ресурси. Ресурсът се дефинира чрез унифициран идентификатор на ресурси – „URI“ – „Uniform Resource Identifier“ и се представя чрез документ, в който е описано текущото или предвиденото състояние за него.

Основните принципи на „REST“ са – възможност за адресиране, липса на състояние и еднороден интерфейс. „REST“ моделите и сетовите от данни трябва да могат да оперират като ресурси, обозначени с идентификатори „URI“. Използва се стандартизиран интерфейс с фиксирана група от „HTTP“ (Hypertext Transfer Protocol – Протокол за пренос на хипертекст) методи. Всяка транзакция е независима и няма връзка с предходните, тъй като данни,

нужни за обработка на дадена заявка се съдържат само в нея. Данните за сесията на клиента не се поддържат от сървърната страна, така че отговорите от сървъра също са независими.

ПОЛУЧИ – „GET“, СЛОЖИ – „PUT“, ПУБЛИКУВАЙ – „POST“ и ИЗТРИЙ – „DELETE“ са някои от основните „HTTP“ методи, използвани при този тип приложения за извличане, създаване, промяна и изтриване ресурси. [14 - 16]

2.2 Технологична рамка с отворен код „Spring Framework“

„Spring framework“ представлява технологична рамка с отворен код, разработена за „Java“ платформата. Съставена е от приблизително 20 различни модула, осигуряващи функциите, които предлага. Те са групирани в няколко основни категории – Основен контейнер - „Core Container“, Достъп до данни/ Интеграция – „Data Access/Integration“, Уеб – „Web“, Аспектно ориентирано програмиране - „AOP“ (Aspect Oriented Programming), Набор от инструменти – „Instrumentation“, Тест – „Test“. Ще разгледаме накратко всяка от тези категории.

- **Модул Основен контейнер – „Core Container“**

В този слой се съдържат модулите Основа - „Core“, Зърна – „Beans“, Контекст – „Context“ и Експресивен език – „Expression Language“.

Модулите „Core“ и „Beans“ осигуряват фундаментални функции на „Spring“ като инверсия на контрол – „IoC“ („Inversion of Control“) и инжектиране на зависимости – „Dependency Injection“.

Модулът „Основа“ предоставя възможност да се достъпват обекти по начин, подобен на регистър, на принципа на „Java“ интерфейс за именуване и директории - „JNDI“ („Java Naming and Directory Interface“). Този модул наследява характеристики от „Beans“ модула и добавя поддръжка за интернационализация, разпространение на събития, зареждане на ресурси, и прозрачно създаване на контексти. Модулът поддържа и функционалности от корпоративното издание на „Java“ – „Java EE“ („Enterprise Edition“).

Модулът за експресивен език осигурява мощна функционалност за създаване на заявки и манипулация на обектни графи по време на изпълнение на приложението. Езикът поддържа методи „получи“ - „get“ и „задай“ – „set“ за деклариране на променливи, извикване на методи, достъп до съдържанието на масиви и колекции, логически и аритметични оператори, именувани променливи и извикване на

обекти по име от контейнера за инверсия на контрол на „Spring“. Поддържа също създаване и селектиране на списъци, както и чести обединения на списъци.

- **Модул Достъп до данни/Интеграция – „Data Access/Integration“**

Този слой е съставен от модулите „Връзка с базата данни“ – „Java Database Connectivity“ („JDBC“), Обектно-релационно съпоставяне – „Object-relational mapping“ („ORM“), Обектно съпоставяне с езикът „XML“ (Разширяем език за маркиране – „Extensible Markup Language“) - „Object XML Mapping“ – „OXM“, Услуга за съобщения – „Java Message Service“ („JMS“) и Транзакция - „Transaction“.

Модулът „Връзка с базата данни“ – „JDBC“ осигурява абстрактен слой, който премахва нуждата от кодиране и преобразуване на специфични кодове за грешки, свързани с базата данни.

Модульът за „Обектно-релационно съпоставяне“ – „ORM“ предоставя интеграционни слоеве за популярни интерфейси като „Устойчив интерфейс за програмиране“ – „JPA“ („Java Persistence Application Programming Interface“), „Хибернация“ – „Hibernate“, „Java Обекти от данни“ - „JDO“ („Java Data Objects“) и други, за да могат да се използват заедно с всички функции, предоставени от „Spring“.

„ОХМ“ модулът поддържа Обектни мапинг имплементации за „Java архитектура за XML свързване“ – „JAXB“ („Java Architecture for XML Binding“), „Castor“, „XML зърна“ - „XML Beans“ и други.

Услугите за съобщения съдържат функции за създаване и унищожаване на съобщения.

Модулът „Транзакция“ поддържа програмен и декларативен мениджмънт на транзакции за класове, които имплементират специални интерфейси и за всички прости Java обекти - „РОJO“ („Plain Old Java Objects“).

• Модул	Уеб	-	Web
---------	-----	---	-----

Модулът се състои от следните слоеве – „Уеб“ – „Web“, „Уеб-Сървлет“ – „Web-Servlet“, „Уеб-Поддръжка“ – „Web-Struts“ и „Уеб-Потрлет“ – „Web-Portlet“. Уеб модулът предоставя основни функции като качване на многокомпонентни файлове, инициализиране на контейнера за инверсия на контрол, използвайки слушатели на сървлети – „servlet listeners“ и уеб-ориентиран контекст на приложението. Модулът съдържа и свързаните с уеб компоненти на отдалечената поддръжка на Spring. Уеб-Сървлет съдържа „Модел-Изглед-Управление“ („Model – View – Controller“ – „MVC“)

```
graph LR; Request --> DS[Dispatcher Servlet]; DS -- Response --> Response; DS -- Request --> HM[Handler Mapping]; HM -- Controller --> DS; DS -- Request --> C[Controller]; C -- ModelAndView --> DS; DS -- View Name --> VR[View Resolver]; VR -- View --> DS; DS <--> |Model| V[View]; V -- Response --> DS;
```

The diagram illustrates the Spring MVC architecture. At the center is the **Dispatcher Servlet**. It receives an incoming **Request** and sends a **Response** back. The **Dispatcher Servlet** interacts with four other components: **Handler Mapping**, **Controller**, **View Resolver**, and **View**.
1. **Handler Mapping**: The **Dispatcher Servlet** sends a **Request** to the **Handler Mapping**, which returns a **Controller**.
2. **Controller**: The **Dispatcher Servlet** sends a **Request** to the **Controller**, which returns a **ModelAndView**.
3. **View Resolver**: The **Dispatcher Servlet** sends a **View Name** to the **View Resolver**, which returns a **View**.
4. **View**: The **Dispatcher Servlet** sends a **Model** to the **View**, which returns a **Response** to the **Dispatcher Servlet**.

„Уеб-Поддръжка“ – „Spring-Struts“ модулят пък се състои от поддържащи класове за интегриране на класическа поддръжка на уеб ниво в дадено „Spring“ приложение.

„Уеб-Потрлет“ – „Web-Portlet“ модулят осигурява възможност „MVC“ моделът да бъде използван в портлет среда и като функционалност е подобен на „Уеб-Сървлет“ модула.

- Този модул на Spring осигурява съюз-съвместимо аспекти-оринетирано програмиране – „АОР“ („Alliance-compliant Aspect-oriented Programming“) имплементация, която дава възможност да се отделят части на от имплементацията на кода, които логически трябва да са разделени.

Модулът „Набор от инструменти“ поддържа организация на класове и имплементации за тяхното зареждане, които да се използват при някои сървъри за приложения.

- **Модул „Тест“ – „Тест“**
Модулът поддържа тестването на „Spring“ компоненти със софтуерните продукти „JUnit“ или „TestNG“. Осигурява непрекъснато зареждане на „Контексти на приложението“ – „Spring ApplicationContexts“ и кеширането на тези контексти. Осигурява и “фиктивни” обекти, с които кодът да се тества изолирано. [18]

2.3 Бази данни. Релационни бази данни

Базите данни представляват колекция от информация, която се управлява от система за мениджмънт на бази данни – „Database Management System,, („DBMS“). Тази система трябва да поддържа следните функции:

- Да позволява на потребителите да създават нови бази данни и да дефинират техните схеми.
- Да дава възможност на потребителите да изпращат запитвания към базата и да могат да я променят, чрез подходящ език.
- Да осигурява възможност за съхранение на много големи количества данни за голям период от време и да позволява ефикасен достъп до тях.
- Да осигурява възстановяване на базата в случай на откази, повреди или неправилна експлоатация.
- Да контролира достъпа до данните от множество потребители наведнъж, без да се получават неочаквани взаимодействия между потребителите.

Базите данни основно се разделят на релационни и нерелационни. За разработка на текущия проект ще се използва релационна база данни, характеристиките на която следва да бъдат разгледани.

2.3.1 Релационни бази данни

При този тип бази данни, данните се представят под формата на двумерни таблици, наречени релации. Колоните в таблицата се наричат атрибути, а редовете представляват отделните инстанции на обекта. За всеки ред от релацията е зададен конкретен първичен тип данни като String или int. Не е позволено да се използва тип данни, чиито стойности могат да се разбият на по-малки компоненти.

Всеки атрибут също има свой първичен тип данни, наречен домейн, който може да бъде включен в описанието на схемата. Релациите представляват сетове от инстанции на даден обект. Съответно атрибутите на една релация могат да се подредят по различни начини, без да бъде променена самата релация.

Ключове

Един или няколко атрибута могат да формират първичен ключ на релацията. Тяхната стойност не може да се повтаря между различните инстанции, което гарантира, че всеки запис в релацията ще може да бъде идентифициран по уникален начин.

Съществуват и външни ключове, които се използват връзка между две отделни релации. Копие от първичният ключ от едната релация се включва към структурата на втората релация, за която той се явява външен. [19]

2.4 Система за управление на релационни бази данни „MySQL“

„MySQL“ е система за управление на релационни бази данни с отворен код, създадена и поддържана от корпорацията „Oracle“. Работи с езикът за структурирани заявки - „SQL“ („Structured Query Language“), който е един от най-разпространените езици за достъп до бази данни. Според програмната среда, „SQL“ заявките могат да бъдат пращани директно, да бъдат включени към код, написан на друг език или да се използва приложение за конкретен език, което скрива „SQL“ заявките. [20]

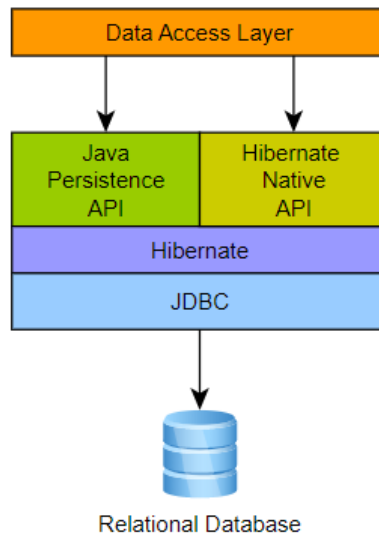
2.5 Технологична рамка „Hibernate ORM“

„Hibernate ORM“ представлява технологична рамка за свързване на обектно-ориентирано приложение към релационна база данни. „Hibernate“ заменя директният достъп до базата данни с функции от високо ниво, обработващи обектите.

Основните му функции са мапинг между „Java“ класове и таблици в базата данни, както и мапинг между различните типове данни в „Java“ и „SQL“. Освен тях има функционалност за заявки и извличане на данни. Hibernate значително скъсява времето за разработка на приложението, което иначе би отишло в ръчен мениджмънт на „SQL“ и „JDBC“.

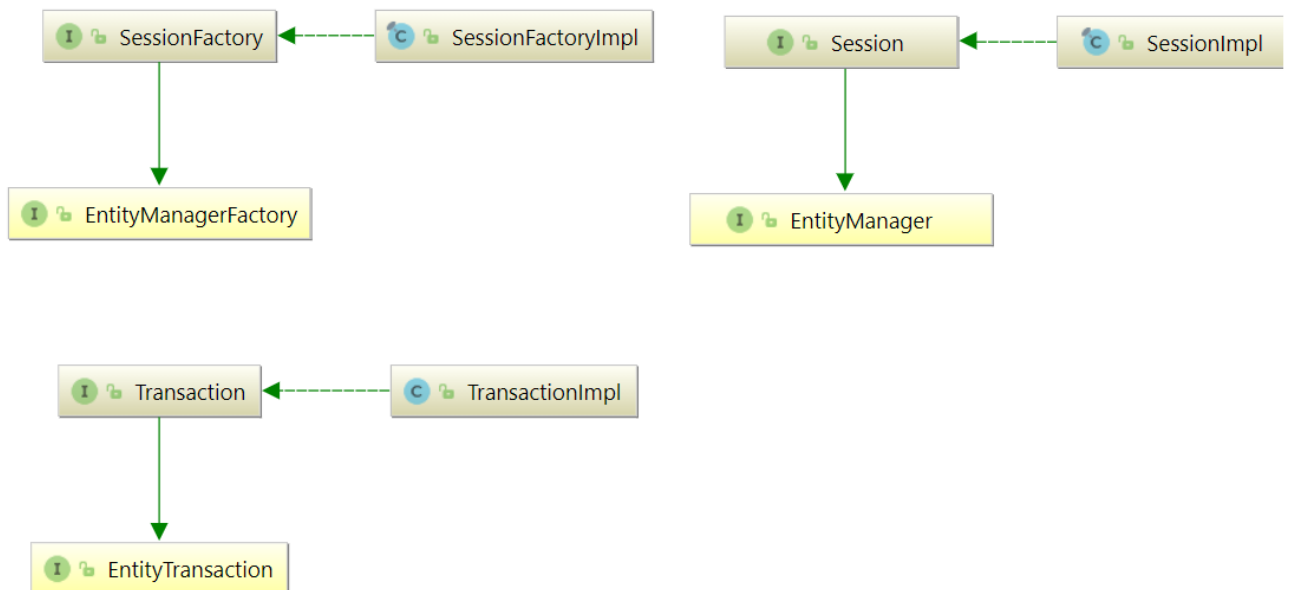
2.5.1 Архитектура на технологичната рамка „Hibernate“

„Hibernate“ като решение от тип Обектно-релационно съпоставяне – „ORM“ стои между нивото за достъп до данни на „Java“ приложението и релационната база данни, както може да се види на фигура 2.3. Софтуерната реализация използва „Hibernate“ приложения за зареждане, съхранение, заявки и др.



Фигура 2.2 – Връзка между приложението и базата чрез „Hibernate“, [21]

„Hibernate“ имплементира спецификациите на „Java Устойчив интерфейс за програмиране“ – „Java Persistence API - JPA“, а връзките между двете са представени на диаграмата на фигура 1.3.



Фигура 2.3 – Връзки между „Java Persistence API“ и „Hibernate“, [21]

Следва да бъдат разгледани основните „Hibernate“ приложения:

- **Фабрика за сесии – „SessionFactory“**

Представяне на мапинга между домейн модела на приложението и базата данни. Играе роля на фабрика за създаване на нови сесии. Поддържа услуги, които „Hibernate“ използва, като кеш памет от второ ниво, пулове за връзка, интеграция на транзакционни системи и др. Фабрика за управление на обекти – „EntityManagerFactory“ в „JPA“ е еквивалентът на Фабрика за сесии в „Hibernate“ и двете се обединяват в еднаква имплементация. Фабриката за сесии е само една за дадено приложение, тъй като е скъпа за създаване.

- **Сесия – „Session“**

Представява еднонишков обект с кратък живот и моделира „Единица работа“. Сесията обвива Връзката в „JDBC“ и играе роля на фабрика за инстанциите от тип Транзакция.

- **Транзакция – „Transaction“**

Обект, използван от приложението за разграничаване на границите на отделните физически транзакции. Еквивалентът в „JPA“ е Транзакция на обекти – „EntityTransaction“ и двете играят роля на абстрактно приложение за да изолират софтуерният продукт от основно използваната транзакционна система „JDBC“.[21]

2.6 Среда за разработка на приложения „Eclipse Integrated Development Environment“

За разработка на приложението ще бъде използвана интегрираната среда за разработка „Eclipse IDE“ („Integrated Development Environment“), тъй като поддържа както „Java“, така и „Spring“.

Приложението предоставя цялостна среда за софтуерна разработка и включва компоненти като редактор на код, инструменти за автоматизация на генерирането на код, компилатор, интерпретатор, дебъгер. Силно улеснява процеса на програмиране, а негов плюс е и фактът, че е с отворен код и е безплатно. [22]

2.7 Платформа с отворен код „Postman“

„Postman“ представлява платформа с отворен код, даваща възможност за създаване, тестване, мониторинг и разработка на документация за софтуерни продукти. [23]

За разработка на дипломното задание платформата ще бъде използвана за изпращане на заявки и тестване работата на приложението.

III глава. Архитектура на програмната реализация

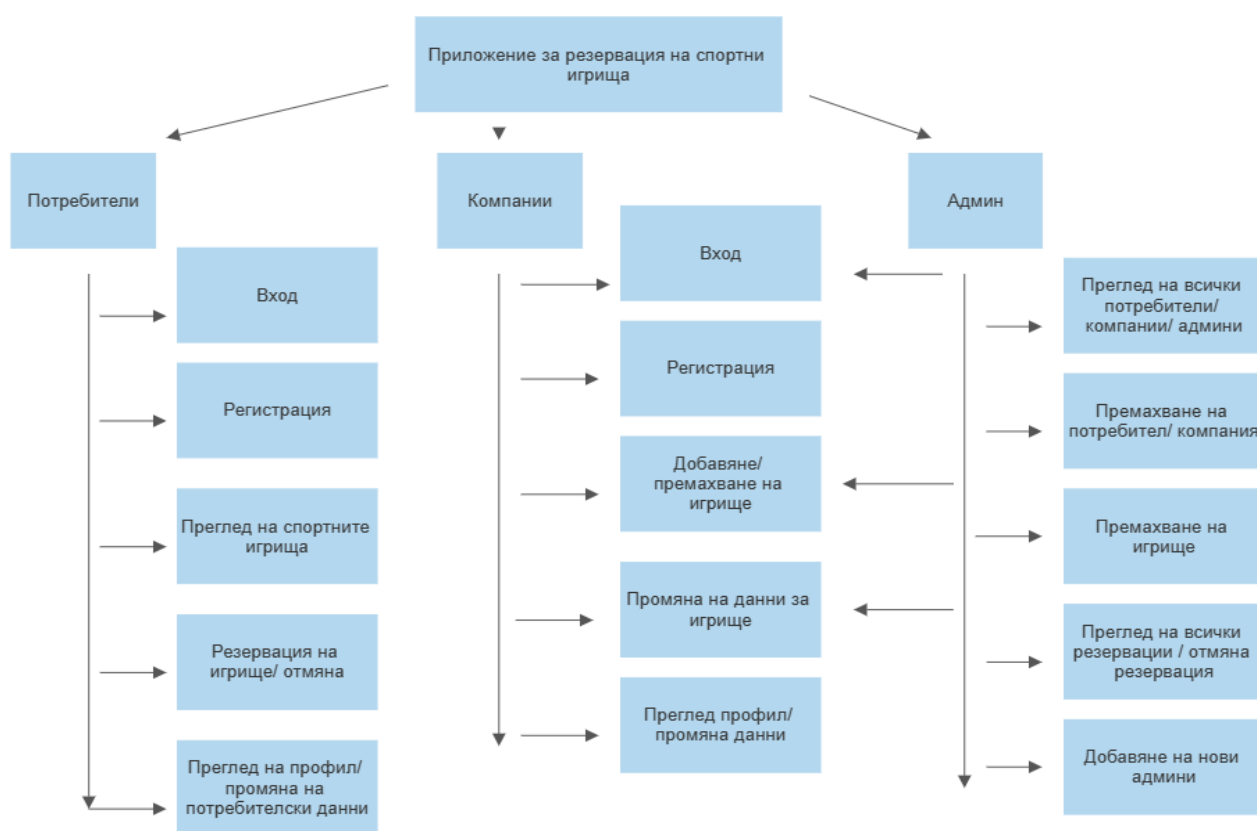
В тази глава следва да бъде разгледана програмната реализация на разработвания дипломен проект, неговите структура, компоненти и функционалности.

За направата на приложението за резервация на спортни игрища са използвани „Java“ версия 17, „Spring Boot“ версия 2.6.0, а за базата данни е използван „MySQL“. Функционалностите на всички използвани технологии са подробно разгледани във втора глава.

Ще бъде представена архитектурата на приложението с основните му функции, след което ще бъде разгледана програмната реализация и конфигурирането на сигурността чрез „Spring“ – „SpringSecurity“, посредством която се осъществява лог-ин на системата и управление достъпа до функциите ѝ, спрямо различните потребители.

3.1 Основна архитектура на приложението

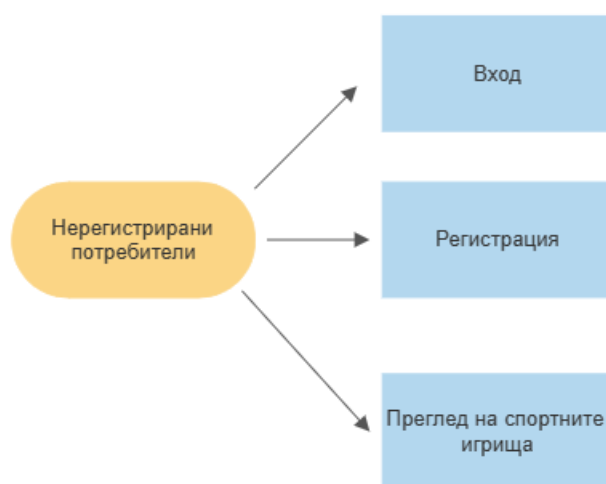
Ще започнем с преглед на основната архитектура на приложението за резервация на спортни игрища, представена на фигура 3.1:



Фигура 3.1 Архитектура на приложението, създадена чрез софтуерният продукт „SmartDraw“, [24]

Потребителите на платформата са разделени на три основни типа, според вида на операциите, които ще извършват – обикновени потребители или Клиенти, Компании и Администратори. Всички те се пазят в общо хранилище, а разграничението помежду им се прави посредством роли, добавяни автоматично при запазване на нов потребител в базата.

Потребител, който все още не е влязъл в своя профил има достъп до следните функции, показани на фигура 3.2 – преглед на спортните игрища – преглед на всички/ филтриране по град/ филтриране по тип на игрището, вход в системата и регистрация като потребител или компания. Регистрацията е разделена на две части, за да може да се добави правилната роля на клиента.



Фигура 3.2 Графика на операции, извършвани от анонимен потребител, създадена чрез софтуерният продукт „SmartDraw“, [24]

След успешен вход потребителите вече могат да направят резервация на избраното игрище, да отменят вече направена такава. Имат достъп до профилните данни, които са въвели при регистрацията и възможност да ги променят, както и преглед на направените от тях резервации.

Регистрираните компании от своя страна имат възможност да добавят спортни игрища, да променят информацията за вече въведени такива, както и за своите профилни данни, също и да премахват вече въведени игрища.

Администраторите имат достъп до допълнителни функции, освен вече изброените за клиенти и компании. Освен премахване/добавяне на игрище,

направа/ отмяна на резервация те имат право да добавят нови администратори, както и да трийт потребители. Имат и възможност за преглед на всички потребители от всеки тип, както и на всички направени резервации.

Структурата на приложението е разпределена в следните компоненти:

- **Обекти - Entities** – обектите, необходими за реализация на приложението;
- **Услуги - Services** – основната логиката на приложението, операции с базата данни;
- **Управление - Controllers** – обработка на заявките и връзка между бизнес логиката и базата данни;
- **Хранилища - Repository** – връзка с базата данни;
- **Сигурност, Конфигурация на сигурността - Security, SecurityConfig** – конфигурация на автентификацията на потребителите и достъпа им до отделните ресурси.

Освен самият код на приложението други важни файлове са „application.properties“, където се настройва връзката с базата данни и „pom.xml“, съдържащ конфигурационни данни за Мейвън.

Подробна информация за функционалното разделение на логиката, отделните класове и методи, какво съдържат, както и връзките между тях ще бъдат разгледани в следващите параграфи.

3.2 Бизнес логика на приложението

3.2.1 Обекти (Entities)

Апликацията използва следните обекти – Потребители - Users, Роли - Role, Резервации - Reservation, Спортните игрища - Field.

Обектът „Потребители“ – „Users Entity“ се използва за създаване на потребителите на приложението. Съдържа като променливи основни данни, необходими за всеки потребител – потребителско име - username, парола - password, (имейл - email, телефонен номер - phoneNumber, име - firstName, фамилия - lastName, статус на потребителя - active. Всички те се задават по време на регистрацията на клиента. Класът съдържа още идентификационен номер – id, генериран автоматично при запазване на нов потребител и сет от роли, с които той ще разполага. Статусът „active“ се задава автоматично на true при добавяне на нов потребител.

Обектът „Роля“ - Role Entity дефинира ролите, които даден клиент на приложението може да има. Всяка роля има име - name и идентификационен номер - roleId.

„Потребители“ и „Роля“ класовете са обвързани помежду си с Много към много – „@ManyToMany“ анотация, така че всеки потребител да може да има повече от една роля и всяка роля да има много потребители. В базата се генерира таблица потребители-роли - users_roles, в която всеки от двата обекта е представен със своя идентификационен номер.

Обект „Спортно игрище“ - Field Entity дефинира самите спортни игрища. Всяко игрище разполага със следните параметри – fieldId – идентификационен номер, fieldName – име на игрището, location – адрес на игрището, type – вид на игрището, state – състояние, дали игрището е свободно или ако е резервирано – в какъв диапазон, price – цена за резервация, contactInformation – контакт със съответната фирма, workingHours – работно време.

Обект „Резервация“ - Reservation Entity дефинира обекта на резервацията. Всяка резервация има идентификатор - reservationId, потребителското име на клиента, от който е направена – madeBy, името на игрището, което е резервирано – fieldName, продължителност на резервацията – reservationDuration.

Всяки обект разполага с методи за извличане/задаване на параметрите - getter/setter, свой конструктор и предефинирани toString() методи.

3.2.2 Хранилища (Repositories)

За всеки от описаните досега обекти имаме кореспондиращо хранилище, осъществяващо връзката с базата данни и съответната таблица.

В **Хранилището за потребители - UserRepository** са дефинирани следните методи:

- **findByEmail()** – намери чрез имейл – търси потребителят по даден имейл и връща резултат от тип Users;
- **findByUsername()** – намери по потребителско име - търси потребителят по потребителско име и връща резултат от тип „Потребител“;
- **findById()** – намери по идентификатор - търси потребителят по идентификационния му номер и връща резултат от тип „Потребител“;

- `deleteById()` – изтрий чрез идентификатор – трие потребителят, посредством идентификатор;
- `findByRolesIn()` – намери по роли - връща списък с всички потребителите, имащи даден тип роля.

Хранилище за роли - `RoleRepository` – има метод намери по име - `findByName()`, търсещ ролята в базата посредством нейното име.

Хранилище за спортни игрища - `FieldRepository` разполага със следните футкции:

- `deleteById()` – изтрива дадено игрище, посредством неговото `id`;
- `FindByFieldId()` – открива игрище чрез `id`;
- `FindByFieldName()` – открива игрището чрез неговото име.

Хранилище за резервации - `ReservationRepository` съдържа методите:

- `findByFieldNameAndReservationDuration()` – търси спортното игрище чрез неговото име и продължителност на резервацията;
- `findById()` – търси игрището чрез неговото `id`;
- `findAllByMadeBy()` – търси всички игрища, направени от даден потребител.

3.2.3 Услуги (Services)

Приложението разполага с четири класа от тип „Услуга“, отговарящи на четирите разгледани обекта – Услуги за потребители - `UserService`, Услуги за роли - `RoleService`, Услуги за резервации - `ReservationService`, Услуги за спортни игрища - `FieldService`. Ще разгледаме методите, съдържащи се във всеки от тях.

Услуги за потребители - `UserService`, съдържа логиката необходима за операциите, свързани с потребителите на приложението – тяхното добавяне, премахване, промяна на данни в базата и други, посредством хранилището да потребители- „`UserRepository`“. Класът разполага със следните методи:

- **`addRegisteredCustomer()`** – **добави регистриран клиент** – чрез този метод се добавят нови потребители от тип `Customer` в базата данни. Програмният код може да се види на фигура 3.3
- Приема като вход обект от тип „Потребител“ с въведени от клиента всички необходими параметри като потребителско име, парола,

имена, имейл, телефонен номер.

Извършва се проверка дали в базата вече не съществува потребител с въведения имейл и потребителско име. Ако такъв потребител съществува, методът връща съответното съобщение до клиента – “Вече съществува потребител с този имейл!” или “Потребителското име е заето!”.

Ако горната проверка мине успешно, на новосъздадения потребител се добавя роля „Клиент“ – „Customer“. Паролата се хешира с помощта на енкодер, за да се осигури нейната сигурност, а за статус на потребителя – „active“ се задава „верен“ – „true“. Клиентът се запазва в базата чрез хранилището за потребители, а методът връща съобщение, че “Потребителят е добавен успешно!”;

```
public String addRegisteredCustomer(Users user){
    if(userRepository.findByEmail(user.getEmail()) != null) {
        return "Вече съществува потребител с този имейл!";
    }
    if(userRepository.findByUsername(user.getUsername()) != null) {
        return "Потребителското име е заето!";
    }else {
        Role roleCustomer = roleRepository.findByName("Customer");
        user.addRole(roleCustomer);

        user.setPassword(passwordEncoder.encode(user.getPassword()));
        user.setActive(true);
        userRepository.save(user);
        return "Потребителят е добавен успешно!";
    }
}
```

Фигура 3.3 метод „addRegisteredCustomer“, част от услуга „UserService“

- **addCompany()** – **добави компания** - аналогичен на горния метод, чрез който се добавят нови компании. Разликата се състои в добавяната роля – в този случай тя е „Компания“ – „Company“;
- **addAdmin()** – **добави админ** – аналогичен на горния метод, чрез който се добавят нови администратори. Ролята в този случай е „Админ“ - „Admin“;
- **viewAllCompanies()** – **виж всички компании** – метод, който връща списък на всички потребители с роля Company или съобщение “Няма регистрирани компании!”, ако такива липсват. Програмният код е представен на следващата фигура:

```

public String viewAllCompanies() {
    if (userRepository.findAll().isEmpty()) {
        return "Няма регистрирани потребители!";
    } else {
        Role roleCompany = roleRepository.findByName("Company");
        Set<Role> searchRoles = new HashSet<>();
        searchRoles.add(roleCompany);
        Iterable<Users> companies = userRepository.findByRolesIn(searchRoles);
        String companiesToString = "";
        for (Users users : companies) {
            companiesToString = companiesToString + users.toString() + " ";
        }
        if (companiesToString == "") {
            companiesToString = "Няма регистрирани компании!";
        }
        return companiesToString;
    }
}

```

Фигура 3.4 метод „viewAllCompanies“, част от услугата „UserService“

Прави се първоначална проверка има ли въведени потребители в базата данни, ако поради някаква причина няма такива методът връща съответен стринг “Няма регистрирани потребители!”;

- **getAllAdmins()** – вземи всички админи – аналогичен на „viewAllCompanies()“, връща списък на всички администратори;
- **viewAllCustomers()** – виж всички клиенти - аналогичен на „viewAllCompanies()“, връща списък на всички клиенти;
- **removeUser()** – премахни потребител - метод за изтриване на потребители от базата данни. За целта се използва идентификационният номер на потребителя. Както може да се види от фигура 3.5 се създава временен обект от тип Потребител и се запазва в променливата „checkIfExist“ (провери дали съществува), а в базата данни, потребителят се търси по идентификационен номер, с цел да се направи проверка съществува ли такъв запис. Ако бъде намерен такъв потребител, неговите роли се премахват, той се изтрива от базата, а ако стойността на „checkIfExist“ е нулева методът връща съобщение “Не съществува потребител с такова id!”.

```

@Transactional
public String removeUser ( long id){
    Users checkIfExists = userRepository.findById(id);
    if (checkIfExists != null) {
        checkIfExists.getRoles().clear();
        userRepository.deleteById(id);
        return "Потребителят е успешно премахнат!";
    } else {
        return "Не съществува потребител с такова id!";
    }
}

```

Фигура 3.5 метод „removeUser“, част от услугата „UserService“

- **changePersonalInformation()** – промени личната информация – чрез тази функция се осигурява възможност за промяна на профилните данни на даден потребител като имена, имейл, телефонен номер и парола. Програмната реализация е представена на фигура 3.6

```

public String changePersonalInformation(Users newUserData, String email){

    Users customerToEdit = userRepository.findById(email);
    customerToEdit.setFirstName(newUserData.getFirstName());
    customerToEdit.setLastName(newUserData.getLastName());
    customerToEdit.setEmail(newUserData.getEmail());
    customerToEdit.setPhoneNumber(newUserData.getPhoneNumber());
    customerToEdit.setPassword(passwordEncoder.encode(newUserData.getPassword()));

    userRepository.save(customerToEdit);
    return "Профилната информация беше успешно обновена!";
}

```

Фигура 3.6 метод „changePersonalInformation“, част от услугата „UserRepository“

Входни данни за метода са обект от тип Потребител, съдържащ променените данни на нашия клиент, както и неговият имейл. Чрез хранилището за потребители той се търси в базата и се запазва в променливата customerToEdit (клиент за редакция). Използвайки get (извлечи) и set (постави) методи извличаме новите данни, който сме получили и ги заменяме в нашия потребител, след което го запазваме обратно в базата данни.

- **viewProfileInfo()** – виж профилна информация – функция, която посредством даден имейл търси потребителят в базата данни. Ако го открие връща профилните му данни чрез функцията toString(), както и направените от него резервации, ако ли не връща съобщение “Няма намерен потребител”. Реализацията е показана на следваща фигура:

```

public String viewProfileInfo(String email){
    Users currUser = userRepository.findByEmail(email);
    if(!(currUser == null)) {
        String userData = currUser.toString();
        String reservationsByUser = reservationService.getReservationHistory("(" + currUser.getUsername() + ")");

        return userData + ", " + reservationsByUser;
    }
    else {
        return "Няма намерен потребител!";
    }
}

```

Фигура 3.7 метод „viewProfileInfo“, част от услугата „userRepository“

Предстои да разгледаме следващата услуга – **Услуга за Роли - RoleService**, отговарящ за логиката, свързана с потребителските роли. Той разполага с два основни метода - за добавяне на роли – addRole() и за извличане на списък на всички роли – getAllRoles(), представени на фигура 3.8.

```

public void addRole(Role role) {
    if((roleRepository.findAll().contains(role))) {
        throw new IllegalArgumentException("Тази роля вече съществува!");
    }else {
        roleRepository.save(role);
    }
}

public Iterable<Role> getAllRoles(){
    if(roleRepository.findAll() == null){
        System.out.println("Няма добавени роли.");
        return null;
    }
    return roleRepository.findAll();
}

```

Фигура 3.8 методи „addRole“ и „getAllRoles“, част от услугата „RoleService“

При добавянето на нова роля, методът получава обект от тип Роля с въведено съответното име на ролята. Извършва се проверка дали тя вече не съществува в базата. Ако се окаже, че вече е добавена се извиква изключение от тип невалиден аргумент – „IllegalArgumentException“ със съобщение, че ролята вече съществува. Ако ли не, тя се запазва. В базата предварително са добавени трите необходими роли – Клиент - Customer, Администратор - Admin, Компания - Company, но при бъдещо разрастване на приложението и необходимост могат да се добавят нови.

GetAllRoles() – вземи всички роли - от своя страна връща списък на всички налични роли, като прави и проверка дали липсват въведени роли.

Услуга за спортни игрища - FieldService - класът разполага с методи, свързани с операции по спортните игрища. Тук посредством хранилищата за игрища и резервации – „UserRepository“, „ReservationRepository“, в базата данни се добавят и премахват игрища, извлича се списък с всички игрища/игрища по зададен град, променят се данни, извършва се самата резервация.

Нека разгледаме прилежащите на този клас методи:

- **addNewField()** – **добави ново игрище** – метод за добавяне на нови спортни игрища към базата данни. Получава като вход обект от тип Спортно игрище с дефинирани променливи като име на игрището, адрес, тип, състояние, цена, информация за контакт и работно време. Проверява дали този обект вече не съществува в базата, ако това е така връща на потребителя съобщение “Това игрище вече съществува!”, ако ли не го запазва и връща “Игрището беше добавено успешно!”.
- **deleteField()** – **изтрий игрище** – изтриване на игрище чрез неговият идентификационен номер. Хранилището за игрища търси съответното игрище, с даденото „id“. Ако не го открие връща съобщение на потребителя “Не съществува игрище с такова id.”, в противен случай игрището се изтрива.
- **getAllFields()** – **вземи всички игрища** – метод, връщащ списък на всички въведени игрища, с допълнителна проверка дали няма никакви въведени такива.
- **getAllFieldsByCity()** – **вземи всички игрища по град** – тази функция връща списък на спортните игрища, намиращи се в зададен от потребителя град. Програмната ѝ реализация е показана на следващата фигура:

```

public List<String> getAllFieldsByCity(String city){
    List<String> fieldsForCity = new ArrayList<>();

    if(fieldRepository.findAll() == null){
        fieldsForCity.add("Няма добавени игрища.");
    }else {
        List<Field> allFieldsIterable = fieldRepository.findAll();
        System.out.print(city);
        for (Field field : allFieldsIterable) {
            String address = field.getLocation();
            if(address.contains(city)) {
                fieldsForCity.add(field.toString());
                fieldsForCity.add(System.lineSeparator());
            }
        }
        if(fieldsForCity.isEmpty()) {
            fieldsForCity.add("Няма добавени игрища за този град!");
        }
    }
    return fieldsForCity;
}

```

Фигура 3.9 метод „getAllFieldsByCity“, част от услугата „FiledService“

Методът получава като вход градът, за който да търси игрища. Започва с проверка дали има въведени игрища в базата – ако няма такива връща съобщение до потребителя, ако има продължава нататък. Всички игрища се пазят в списъка allFieldsIterable (променлива всички полета). Чрез „for“ цикъл за всяко се извиква адресът му и се прави проверка съдържа ли той зададеният от потребителя град. Ако градът е част от адреса игрището се добавя в нов списък fieldsForCity (игрища за града).

Накрая се прави проверка дали списъкът „fieldsForCity“ не е празен. Ако това е така към него се добавя съобщение до потребителя “Няма добавени игрища за този град!”.

- **getFieldById()- вземи игрище чрез идентификатор** – метод, търсещ дадено игрище в базата данни. Ако го открие го връща като резултат, а ако то не съществува потребителят получава съобщение “Не съществува потребител с такова id.”
- **changeFieldState()** – **промени състоянието на игрището** – функция, променяща състоянието на дадено игрище, операция, необходима както при резервация, така и при отмяната ѝ. Програмната ѝ реализация е показана на следващата фигура.

```

public void changeFieldState(int fieldId, String state){
    Field toEdit = fieldRepository.findByFieldId(fieldId);
    String changeStateTo = "";
    if(toEdit.getState().contains(state)) {
        changeStateTo = toEdit.getState().replaceAll(state, "");
        if(changeStateTo.equals("") || changeStateTo.equals(" ")) {
            changeStateTo = "Свободно";
        }
    }
    else if(toEdit.getState().equals("Свободно")) {
        changeStateTo = state;
    }else {
        changeStateTo = toEdit.getState() + " " + state;
    }
    toEdit.setState(changeStateTo);
    fieldRepository.save(toEdit);
}

```

Фигура 3.10 метод „changeFieldState“, част от услугата „FieldService“

Функцията получава като вход идентификационният номер на игрището и стринг със състоянието, което искаме да зададем. Спортното игрище ще се извлече от базата в променливата toEdit (за редактиране). Създава се празен стринг за новото състояние.

Прави се проверка дали текущото състояние вече съдържа стрингът, с който трябва да го заменим. Това е случаят, когато се изтрива резервация и периодът ѝ трябва да се премахне от състоянието. Ако този стринг бъде открит се премахва, като се заменя с празен.

Ако го няма в текущото състояние – при резервация го добавяме към вече съществуващият статус и запазваме новите данни в базата. А ако state се равнява на празен стринг – имало е резервации, но те са изтрети, се сменя обратно на “Свободно”.

- **reserve() – резервирай** – функция за резервиране на дадено спортно игрище. Нека разгледаме програмната ѝ реализация на фигура 3.11:


```

public String reserve(String madeBy, int fieldId, String duration) {
    Field fieldToReserve = fieldRepository.findByFieldId(fieldId);
    Reservation reservation = new Reservation();

    if(fieldToReserve.getState().contains(duration)){
        return "Игрището вече е резервирано за този период. Моля изберете друг.";
    }else {
        reservation.setFieldName(fieldToReserve.getFieldName());
        reservation.setMadeBy(madeBy);
        reservation.setReservationDuration(duration);
        reservationRepository.save(reservation);
        long reservationId = reservationRepository
            .findByFieldNameAndReservationDuration(fieldToReserve.getFieldName(), duration).getId();

        String newState = String.format("Резервирано за " + duration);
        changeFieldState(fieldId, newState);

        return String.format("Игрището %s е резервирано от %s за периода %s. Вашият номер на резервацията е %d.",
            fieldToReserve.getFieldName(), madeBy, duration, reservationId);
    }
}

```

Фигура 3.11 метод „reserve“, част от услугата „FieldService“

Вход на тази функция са следните променливи – madeBy (направено от – потребителското име на клиента, правещ резервацията), fieldId – id на игрището, duration – времетраене на резервацията.

Игрището се открива в базата данни и се запазва във временна променлива. Създава се и нова инстанция на Reservation - reservation.

Първо се прави проверка дали състоянието на игрището вече не съдържа периода на резервация. Ако това е така, потребителят получава съобщение “Игрището вече е резервирано за този период. Моля изберете друг”. В противен случай игрището е свободно и се продължава със следващите стъпки по резервацията.

Към reservation се добавят името на спортното игрище, от кого се прави резервацията, времетраенето ѝ, след което новата резервация се запазва. След успешното ѝ добавяне, тя се извиква вече от базата, за да достъпим идентификационния ѝ номер.

Финалните стъпки са промяна състоянието на игрището – добавяме към него стринг “Игрището е резервирано” и за какъв диапазон.

Функцията приключва с връщане на съобщение към потребителя, че игрището е резервирано за избрания период и какъв е номерът на резервацията.

- **changeFieldInfo()** - промени информацията за игрище – метод за промяна данните на вече съществуващо игрище. Като вход получава обект от тип Спортно игрище с новите данни, както и идентификационен номер на игрището.

То се търси в базата, ако не бъде открито се връща съобщение до потребителя “Няма игрище с такова id!”. Ако бъде намерено, чрез get(вземи)/set(постави) методи старите данни се заменят с нови, базата се обновява, а потребителят получава съобщение “Игрището е успешно обновено!”.

- **getAllFieldsByType()** – вземи всички игрища от тип – функция, която чрез зададен от потребителя тип игрище, да върне всички, отговарящи на него или съобщение, че “Няма добавени игрища от тип ” плюс ключовата дума, по която потребителят търси.

Последната услуга, която ще разгледаме е **Услуга за резервации - ReservationService**, където обработваме операциите по резервация на дадено спортно игрище. Ще разгледаме четирите метода, които се съдържат в този клас – reserveField() – резервирай игрище, cancelReservation() – отмени резервация, getReservationHistory() – вземи историята на резервации, viewAllReservations() – виж всички резервации:

- **reserveField()** – функция за резервиране на игрище, представена на фигура 3.12:

```
public String reserveField(String madeBy, int id, String duration) {  
    return fieldService.reserve(madeBy, id, duration);  
}
```

Фигура 3.12 метод reserveField, част от ReservationService

Този метод, получава като вход потребителско име, идентификационен номер и период на резервация, след което извиква „fieldService“ и неговият метод „reserve“, разгледан подробно в предходните параграфи.

- **cancelReservation()** – отмени резервация – съдържа логиката, необходима за отмяна на резервация. Програмната реализация е представена на следващата фигура:

```

public String cancelReservation(long reservationId, int fieldId) {
    Reservation toCancel = reservationRepository.findById(reservationId);
    if(toCancel == null) {
        return "Няма резервация с този номер!";
    }else {
        String reservationPeriod = toCancel.getReservationDuration();
        String stateToRemove = "Резервирано за " + reservationPeriod;
        fieldService.changeFieldState(fieldId, stateToRemove);
        reservationRepository.delete(toCancel);
        return "Резервацията е успешно отменена!";
    }
}

```

Фигура 3.13 метод „cancelReservation“, част от услуга „ReservationService“

Функцията приема за вход номера на резервацията и идентификационен номер на спортното игрище. Посредством хранилището за резервации тя се търси чрез нейния номер и се запазва във временен обект. Извиква се услугата за игрища „fieldService“ с метода „changeFieldState“ за да променим състоянието на игрището и да премахнем периода на резервация, след което самата резервация се изтрива от базата.

- **getReservationHistory()** – **вземи историята на резервации** – метод, който приема за вход потребителско име и връща списък с всички резервации, направени от дадения потребител.
- **viewAllReservations()** – **виж всички резервации** – функция, връщаща списък с всички направени резервации

3.2.4 Управление (Controllers)

За реализация на приложението са използвани два класа за управление – Основно Управление - MainController и Потребителско управление - UserController, в които чрез „REST“ заявки, адреси и методите на вече разгледаните услуги се извършва комуникацията между сървъра и потребителя.

Основното управление - MainController включва методи за преглед на спортните игрища, за добавяне и преглед на роли и хоум страница, на която се визуализират страниците, до които имат достъп нерегистрираните потребители. Всеки метод има дефиниран свой път, чрез който може да бъде достъпен от потребителя. Част от тях могат да се видят на следващото изображение:

```

@GetMapping("/view_all_fields") |
public List<String> getAllFields(){
    return fieldService.getAllFields();
}

@GetMapping("/view_all_fields_for_city/{city}")
public List<String> getAllFieldsForCity(@PathVariable String city){
    return fieldService.getAllFieldsByCity(city);
}

@GetMapping("/view_all_fields_for_type/{type}")
public List<String> getAllFieldsForType(@PathVariable String type){
    return fieldService.getAllFieldsByType(type);
}

@PostMapping("/add_role")
public String addNewRole(@RequestBody Role role) {
    roleService.addRole(role);
    return "Ролята е успешно добавена!!";
}

@GetMapping("/get_all_roles")
public Iterable<Role> getAllRoles(){
    return roleService.getAllRoles();
}

```

Фигура 3.14 Клас „Основно управление“ – „MainController“

Приложението разполага с три метода за достъп до спортните игрища от потребителя – преглед на всички добавени – `getAllFields()` (вземи всички игрища), и игрища, филтрирани според града, в който се намират или техния тип. В този случай, потребителят задава към адреса допълнителна ключова дума, по която да се извърши филтрацията.

В този контролер се извършва и добавянето и прегледа на потребителските роли. В базата предварително са добавени трите роли, с които оперираме - Администратор - Admin, Компания - Company, Клиент - Customer, но е оставена възможност за бъдещо прибавяне на нови. Операциите с роли, могат да се извършват само от администратор.

Потребителското управление - UserController съдържа всички останали методи, обработващи операциите по добавяне, преглед, изтриване на потребители, промяна на данни, добавяне/премахване на игрища, резервации и тяхната отмяна. Включва пътища до хоум страници за всеки тип потребител – Клиент, Админ и Компания, в които след успешен лог-ин се визуализират страниците, до които те имат достъп. Адресите на страниците тук могат да се разделят на няколко категории:

- Страници, до които имат достъп няколко типа потребители, видни на следващата фигура:

```

@RequestMapping(value = "/view_profile_info", method = RequestMethod.GET)
@ResponseBody
public String customerProfile(Principal principal){
    String email = principal.getName();
    System.out.print(email);
    return userService.viewProfileInfo(email);
}

@PutMapping("/change_user_information/{email}")
public String changeUserInformation(@RequestBody Users newCompanyData, @PathVariable String email){
    return userService.changePersonalInformation(newCompanyData, email);
}

@DeleteMapping("/delete_field/{fieldId}")
public String removeField(@PathVariable int fieldId){
    fieldService.deleteField(fieldId);
    return String.format("Игрище с идентификационен номер %d е изтрито!", fieldId);
}

@PutMapping("/reserve_field/{madeBy}/{fieldId}")
public String reserveField(@PathVariable String madeBy, @PathVariable int fieldId, @RequestBody String duration){
    return reservationService.reserveField(madeBy, fieldId, duration);
}

@DeleteMapping("/cancel_reservation/{reservationId}/{fieldId}")
public String cancelReservation(@PathVariable long reservationId, @PathVariable int fieldId){
    return reservationService.cancelReservation(reservationId, fieldId);
}

```

Фигура 3.15 Клас „Потребителско управление“ - „UserController“, част 1

customerProfile() – клиентски профил - методът позволява на потребителя да може да достъпи профилните си данни и резервациите, които е направил. За целта чрез Principal обекта се достъпва имейлът, на вече влязъл в системата потребител, който да се подаде към „viewProfileInfo“ методът на класът „userService“.

changeUserInformation() – промени потребителската информация – към адреса на метода потребителя дописва имейл адреса си, който се подава към услугата за потребители, заедно с обект от тип Потребител, съдържащ въведените от клиента параметри, които ще се променят, заедно с новите им стойности.

removeField() – премахни игрище– приема като променлива id на потребителя, въведено към адреса на метода.

reserveField() – резервация на игрище – за да се извърши тя, методът приема два параметъра към своя адрес – madeBy (направена от) – потребителското име на клиента, правещ резервацията и идентификационния номер на полето. Потребителя допълнително въвежда стринг с периода на резервация.

cancelReservation() – отмени резервация – отмяна на вече направена резервация, за целта към адреса се добавят номер на резервация и id на игрището, за което е направена.

- Страници, до които имат достъп само администратори, програмният им код е представен на фигура 3.16:

```
@GetMapping("/admin/view_all_companies")
public String viewAllCompanies() {
    return userService.viewAllCompanies();
}

@GetMapping("/admin/view_all_reservations")
public String viewAllReservations(){
    return reservationService.viewAllReservations();
}

@GetMapping("/admin/view_all_customers")
public String viewAllCustomers(){
    return userService.viewAllCustomers();
}

@GetMapping("/admin/get_all_admins")
public String viewAllAdmins(){
    return userService.getAllAdmins();
}

@Transactional
@DeleteMapping("/admin/delete_user/{userId}")
public String deleteUser(@PathVariable long userId){
    return userService.removeUser(userId);
}

@PostMapping("/admin/register_admin")
public String addNewAdmin(@RequestBody Users user){
    String resultString = userService.addAdmin(user);
    return resultString;
}
```

Фигура 3.16 клас „Потребителско управление“ – „UserController“, част 2

Тук по специфични са само функциите - deleteUser() – изтрий потребител, която дава възможност на админа да премахва регистрирани клиенти. За целта към адреса на страницата се добавя идентификатор на потребителя, който да се подаде към „removeUser()“ метода от класа Услуга за потребители – „UserService“.

Функцията addNewAdmin() – добави нов админ регистрира нов администратор, като този процес може да се извърши само от вече влязъл в профила си админ. За целта клиента попълва всички данни за новия админ, които да се приемат като обект от тип Потребител от заявката и се предават на класа „услуга“.

- Методи, достъпни до компании:

```
@PutMapping("/company/change_field_information/{id}")
public String changeFieldInformation(@RequestBody Field newData, @PathVariable int id){
    return fieldService.changeFieldInfo(newData, id);
}

@PostMapping("/company/add_new_field")
public String addNewField(@RequestBody Field field) {
    return fieldService.addNewField(field);
}
```

Фигура 3.17 клас „Потребителско управление“ – „UserController“, част 3

Методът addNewField() – добави ново поле - приема обект от тип Спортно игрище с попълнени всички данни за него, а changeFieldInformation() – промени информацията за игрище - променя данните за вече добавено игрище. За целта чрез @RequestBody (Поискай тялото) приема новите данни на игрището, а към адреса на страницата се добавя идентификационният му номер, за да може да бъде открито в базата.

- Хоум страници за админ, клиент, компания:

```
@GetMapping("/admin/home")
public String adminHomePage(){
    String welcome = "Добре дошли във вашият администраторски профил профил!\n\n";
    String menuString = "Меню: \n\n";
    String profile =
        "<HTML><body> <a href='\"http://localhost:8080/view_profile_info\"'>Преглед на профилни данни</a></body></HTML>";
    String editProfile =
        "<HTML><body> <a href='\"http://localhost:8080/change_user_information/{email}\"'>Промяна на профилни данни</a></body></HTML>";
    String removeUser =
        "<HTML><body> <a href='\"http://localhost:8080/admin/delete_user/{companyId}\"'>Изтриване на потребител</a></body></HTML>";
    String removeFields =
        "<HTML><body> <a href='\"http://localhost:8080/delete_field/{fieldId}\"'>Изтриване на игрище</a></body></HTML>";
    String cancelReservation =
        "<HTML><body> <a href='\"http://localhost:8080/cancel_reservation/{reservationId}/{fieldId}\"'>Отмяна на резервация</a></body></HTML>";
    String viewAllReservation =
        "<HTML><body> <a href='\"http://localhost:8080/admin/view_all_reservations\"'>Преглед на всички резервации</a></body></HTML>";
    String viewAllCustomers =
        "<HTML><body> <a href='\"http://localhost:8080/admin/view_all_customers\"'>Преглед на всички клиенти</a></body></HTML>";
    String viewAllCompanies =
        "<HTML><body> <a href='\"http://localhost:8080/admin/view_all_companies\"'>Преглед на всички компании</a></body></HTML>";
    String viewAllAdmins =
        "<HTML><body> <a href='\"http://localhost:8080/admin/get_all_admins\"'>Преглед на всички админи</a></body></HTML>";
    String addNewAdmins =
        "<HTML><body> <a href='\"http://localhost:8080/admin/register_admin\"'>Преглед на всички админи</a></body></HTML>";
}
```

Фигура. 3. 17 клас „Потребителско управление“ – „UserController“, част 4

Тъй като текущият проект е само бек-енд реализация, тези страници целят да визуализират просто меню с пътищата до функциите, до които всеки тип потребител ще има достъп. Нагледното им разделение може да се види на фигура 3.1, представена в началото на трета глава.

След успешен вход на потребителя се визуализира приветстващо съобщение “Добре дошли във вашия профил!”, заедно с линкове към съответните потребителски функции.

- Страници за регистрация със свободен достъп

Разделени са на две – регистрация на клиент и на компания, поради различните роли, които се разпределят към всеки тип. Имплементацията им е представена на следващата фигура, а принципът на работа е аналогичен на метода `addNewAdmin()`:

```
@PostMapping("/company_add_company")
public String addCompany(@RequestBody Users user){
    String statusString = userService.addCompany(user);
    return statusString;
}

@PostMapping("/customer_register_customer")
public String addNewCustomer(@RequestBody Users user){
    String resultString = userService.addRegisteredCustomer(user);
    return resultString;
}
```

Фигура 3.19 | 17 клас „Потребителско управление“ – „UserController“, част 5

3.2.5 Защита на приложението и нейната конфигурация

В тази секция ще бъде разгледана реализацията на класовете Персонализирани потребителски детайли - `CustomUserDetails`, Услуга за пресонализирани потребителски детайли - `CustomUserDetailsService` и Конфигурация на сигурността - `SecurityConfig`, необходими за осигуряване на защитата на приложението.

Да започнем от **Персонализираните потребителски детайли - CustomUserDetails**, който имплементира класът Потребителски детайли - `UserDetails` от технологичната рамка „Spring Security“. Програмната му реализация е представена на следващите две фигури:


```

public class CustomUserDetails implements UserDetails {

    private Users user;

    public CustomUserDetails(Users user) {
        this.user = user;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        Set<Role> roles = user.getRoles();
        System.out.println(roles);
        List<SimpleGrantedAuthority> authorities = new ArrayList<>();

        for (Role role : roles) {
            authorities.add(new SimpleGrantedAuthority(role.getName()));
        }

        return authorities;
    }
}

```

Фигура 3.20 Имплементация на клас „Персонализирани потребителски детайли“ - „CustomerUserDetails“, част 1

```

@Override
public String getPassword() {
    return user.getPassword();
}

@Override
public String getUsername() {
    return user.getEmail();
}

@Override
public boolean isAccountNonExpired() {
    return true;
}

@Override
public boolean isAccountNonLocked() {
    return true;
}

@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return user.isActive();
}

```

Фигура 3.21 Имплементация на клас „Персонализирани потребителски детайли“ - „CustomerUserDetails“, част 2

Дефинирана е променлива user, която представлява потребителят, който трябва да бъде упълномощен при вход в приложението. Класът има конструктор, след което се заместват основни методи от клас „UserDetails“:

- getAuthorities() – вземи правомощията – тук се извличат ролите, които има потребителя;
- getPassword() – вземи паролата – извиква се паролата на потребителя;
- getUsername() – вземи потребителското име – в текущия случай се извиква имейлът на потребителя;
- isAccountNonExpired() – изтекъл ли е акаунта;
- isAccountNonLocked() – заключен ли е акаунта;
- isCredentialsNonExpired() – валидни ли са идентификационните данни;
- isEnabled – актиен ли е – проверка дали потребителят е с активен статус.

Следващият необходим клас е Услуга за пресонализирани потребителски детайли - „CustomUserDetailsService“, имплементиращ класът „Услуга за потребителски данни“ „UserDetailsService“ от „SpringSecurity“. Неговият код е представен на фиг. 3.16:

```
1
2 @Service
3 @Transactional
4 public class CustomUserDetailsService implements UserDetailsService {
5
6     @Autowired
7     private UserRepository userRepository;
8
9     @Override
10    public UserDetails loadUserByUsername(String email)
11        throws UsernameNotFoundException {
12        Users user = userRepository.findByEmail(email);
13
14        if (user == null) {
15            throw new UsernameNotFoundException("Could not find user");
16        }
17
18        return new CustomUserDetails(user);
19    }
20
21 }
```

Фигура 3.22 Имплементация на класът „Услуга за пресонализирани потребителски детайли“ - „CustomUserDetailsService“

Тук се заменя само един метод на класът „UserService“ – loadUserByUsername() – зареди потребителят чрез потребителското име, който стандартно приема като вход потребителското име, а в случая имейлът на потребителя. Извършва се проверка дали потребителят фигурира в базата данни, ако липсва методът връща изключение за ненамерено потребителско име – „UsernameNotFoundException“, в противен случай се създава нова инстанция на класа „CustomUserDetails“ с потребителят от базата данни.

Накрая остава да бъде разгледан файлът **Конфигурация на сигурността - SecurityConfig**, наследяващ класът „WebSecurityConfigurerAdapter“ – „Адаптер за конфигурация на уеб сигурност“ от „SpringSecurity“ и съдържащ настройките за сигурността на приложението. Програмната му реализация е видима на следващите фигури:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    public UserDetailsService usersDetailsService() {
        return new CustomUserDetailsService();
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();
        authenticationProvider.setUserDetailsService(usersDetailsService());
        authenticationProvider.setPasswordEncoder(passwordEncoder());
        return authenticationProvider;
    }
}
```

Фигура 3.23 Клас „Конфигурация на сигурността“ – „SecurityConfig“, част 1

```
@Override
public void configure(WebSecurity web) throws Exception {
    web.ignoring()
        .antMatchers("/home")
        .antMatchers("/view_all_fields")
        .antMatchers("/view_all_fields_for_city/{city}")
        .antMatchers("/view_all_fields_for_type/{type}")
        .antMatchers("/company_add_company")
        .antMatchers("/customer_register_customer");
}

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.authenticationProvider(authenticationProvider());
}
```

Фигура 3.24 Клас „Конфигурация на сигурността“ – „SecurityConfig“, част 2

```

@Override
protected void configure(HttpSecurity http) throws Exception {

    http.authorizeRequests()
        .antMatchers("/home").permitAll()
        .antMatchers("/view_profile_info", "/change_user_information/{email}").hasAnyAuthority("Admin", "Customer",
        .antMatchers("/reserve_field/{madeBy}/{fieldId}", "/cancel_reservation/{reservationId}/{fieldId}")
        .hasAnyAuthority("Admin", "Customer")
        .antMatchers("/delete_field/{fieldId}").hasAnyAuthority("Admin", "Company")
        .antMatchers("/admin/**", "add_role", "get_all_roles").hasAuthority("Admin")
        .antMatchers("/customer/**").hasAuthority("Customer")
        .antMatchers("/company/**").hasAuthority("Company")
        .anyRequest().authenticated()
        .and()
        .formLogin().permitAll()
        .and()
        .logout().permitAll();
}

```

Фигура 3.25 Клас „Конфигурация на сигурността“ – „SecurityConfig“, част 3

Тук за методът `userDetailsService()` – услуга потребителски детайли се извиква инстанция на услугата „`CustomUserDetailsService()`“, която беше разгледан в предходните параграфи. Целта на това е да използваме промените, направени за текущото приложение, а не класът, който идва по подразбиране със „`SpringSecurity`“.

За функцията енкодер на пароли - `passwordEncoder()` на приложението е избран „`BCryptPasswordEncoder`“, чрез който да се хешират и обработват потребителските пароли, така че да бъдат защитени от опити за злонамерен достъп.

Конфигурира се „`DaoAuthenticationProvider`“, който използвайки методите „`userDetailService()`“ и „`passwordEncoder()`“ удостоверява потребителското име и паролата, които потребителят е въвел в лог-ин формата.

В метода конфигурация - `configure(WebSecurity web)` са дефинирани пътищата в приложението, който нямат нужда от защита и ще бъдат със свободен достъп за потребителите. Това са хоум страницата, трите типа преглед на игрища и регистрация на потребител/клиент.

Методът конфигурация - `configure(HttpSecurity http)` е пренаписан, така че бъде разграничен достъпът до отделните ресурси в приложението.

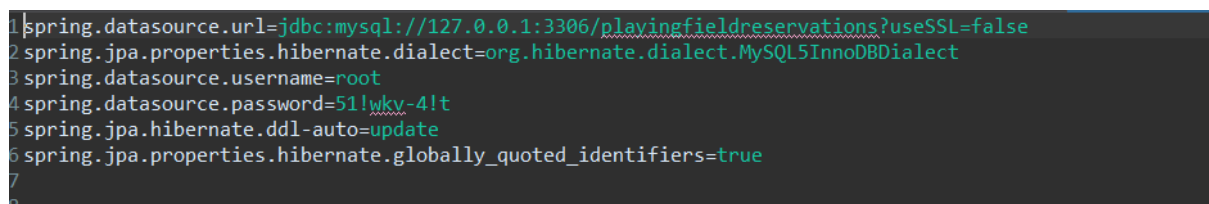
За удобство при разделянето на пътищата, тези, до които ще имат достъп само администратори започват с „`/admin/`“, клиентите с „`/customer/`“, а компаниите с „`/company/`“. Така при автентификацията на потребителя се проверява неговата роля чрез функцията `has Authority()` – има правомощие -

и ако тя не отговаря на зададената в конфигурацията, той няма да получи достъп.

Пътищата, които ще се използват от потребители с повече от една роля са отделени в отделни секции, а ролите се проверяват чрез функцията `hasAnyAuthority` – има което и да е правомощие. Тук фигурират страниците за преглед и промяна на профилната информация, до които трябва да имат достъп всички типове потребители и резервацията на игрище/ отмяната ѝ, до които трябва да достъпват Клиент и Администратор.

За лог-ин и лог-аут форма в приложението е използвана вградената в „Spring Security“ форма за лог-ин - `formLogin()`.

3.2.6 Файл с настройки на приложението „application.properties“

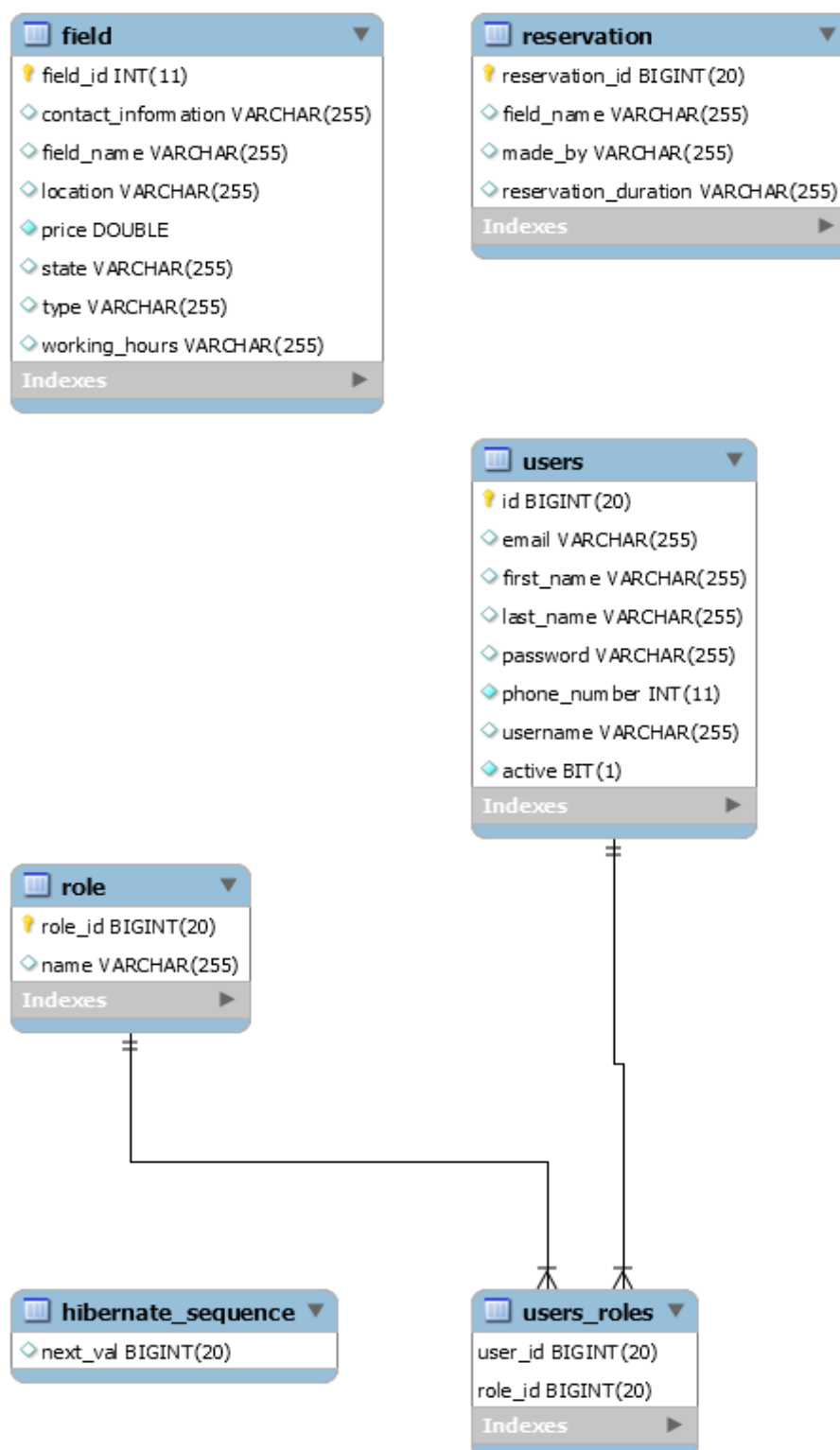
A screenshot of a text editor showing the content of the application.properties file. The text is as follows:

```
1 spring.datasource.url=jdbc:mysql://127.0.0.1:3306/playingfieldreservations?useSSL=false
2 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
3 spring.datasource.username=root
4 spring.datasource.password=51!wky-4!t
5 spring.jpa.hibernate.ddl-auto=update
6 spring.jpa.properties.hibernate.globally_quoted_identifiers=true
7
8
```

Фигура 3.26 Файл с настройки „ApplicationProperties“ на приложението

В този файл съдържа настройки на „Spring“ за свързването на базата данни към приложението, като „Hibernate“ е настроен със стойност „update“ -обновяване, за базата да не да стартира отначало при всяко пускане на приложението, а да се обновява.

3.2.7 Диаграма на връзките между обектите (ER – Entity Relationship diagram)



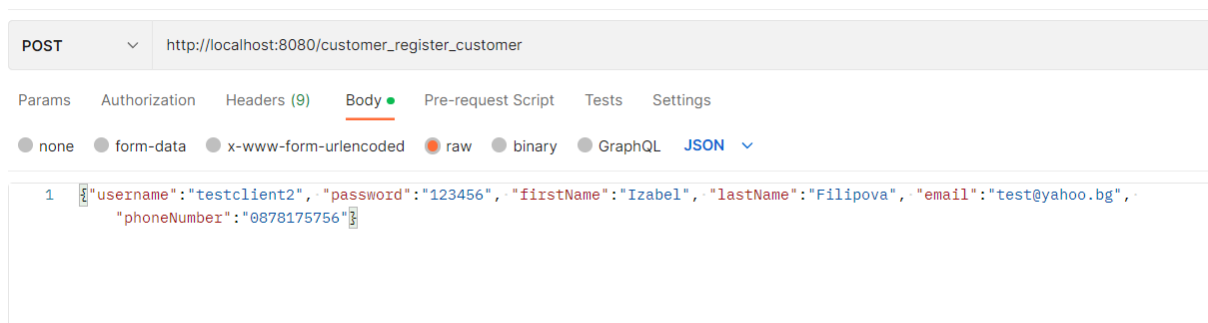
Фигура 3.27 „ER“ диаграма, генерирана чрез „MySQL Workbench“

IV глава. Експериментални данни и изводи

4.1 Експериментални данни

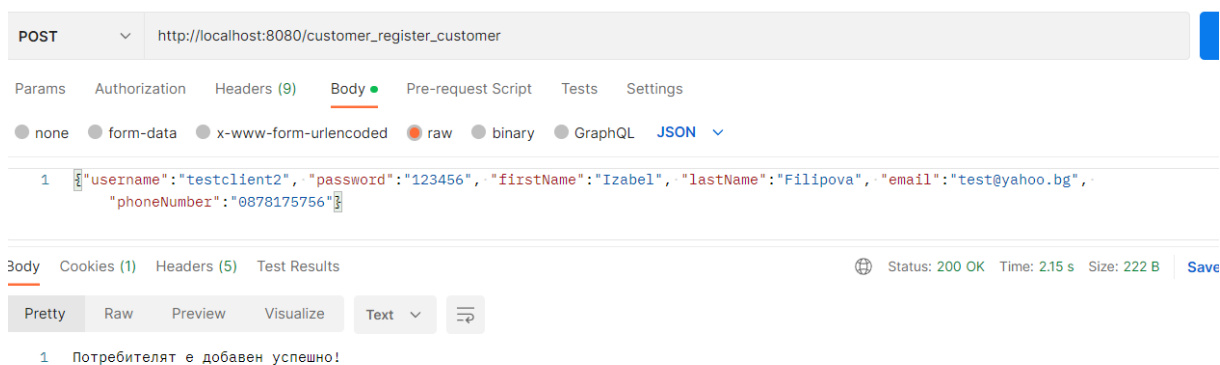
В тази секция нагледно ще бъдат показани част от функциите на приложението. Тестването се извършва с Postman за заявки различни от „GET“, тъй като проектът е на ниво бек-енд и няма как останалите заявки да се обработят директно през брауъра.

- **Регистрация на нов клиент:**



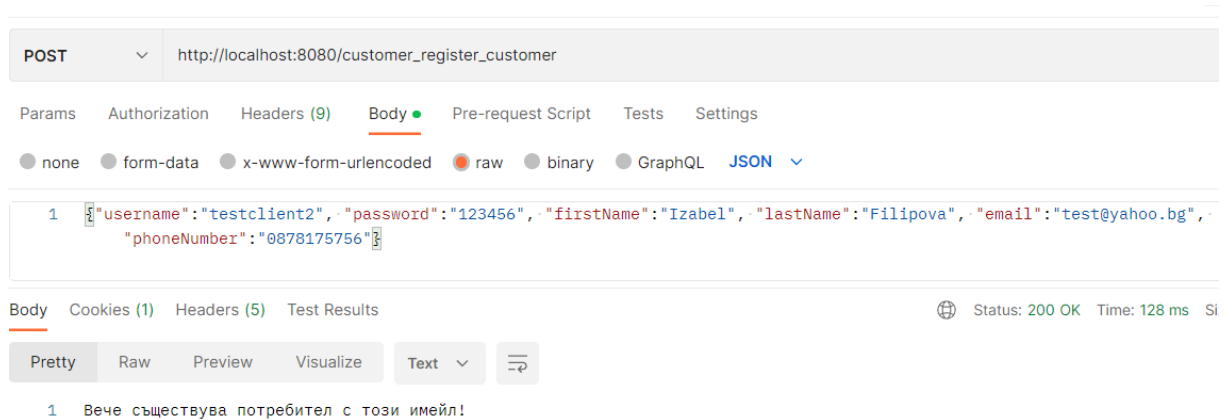
Фигура 4.1 Регистриране на клиент

Задават се всички необходими параметри без „id“ и „state“. Ако потребителят бъде запазен успешно приложението връща съобщение за това:



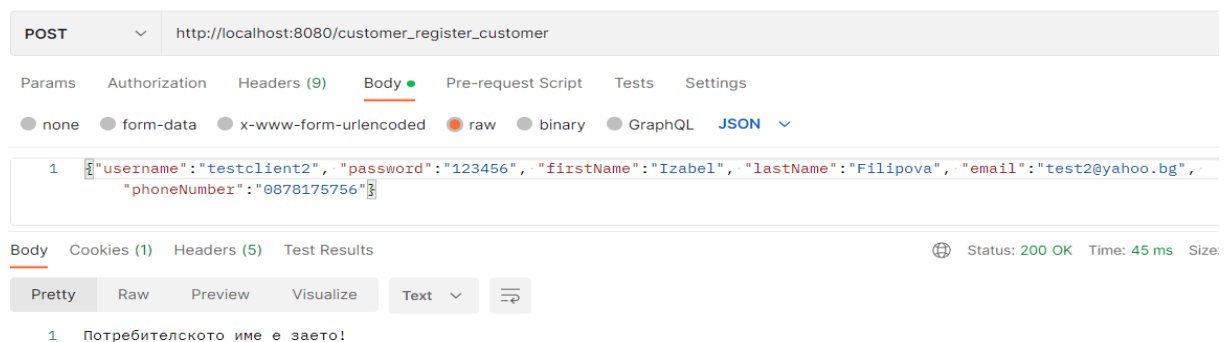
Фигура 4.2 Успешно добавен потребител

- **Опит за регистрация на потребител с вече съществуващ имейл в базата:**



Фигура 4.3 Регистрация с невалиден имейл

- **Опит за регистрация на потребител с вече съществуващо потребителско име в базата:**



Фигура 4.4 Регистрация с невалидно потребителско име

- **Вход/изход в приложението чрез лог-ин форма „formLogin()“**

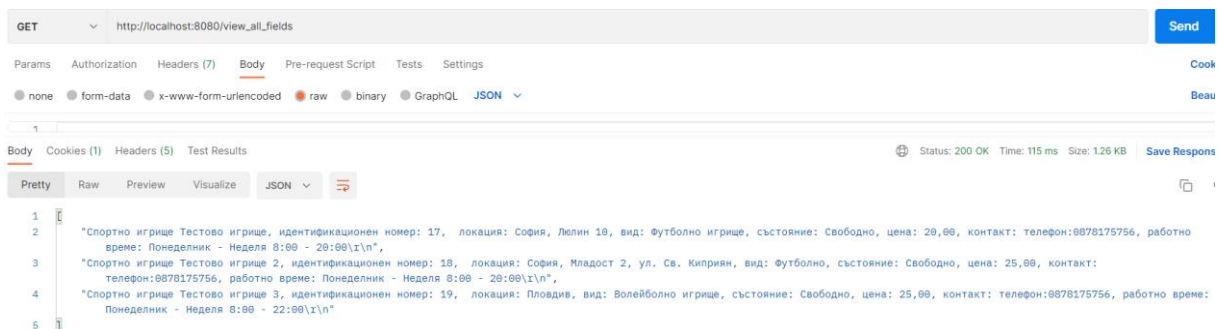


Фигура 4.5 Лог-ин форма

- **Вход с грешни данни:**

Фигура 4.6 Невалиден имейл или парола

- **Преглед на всички игрища:**



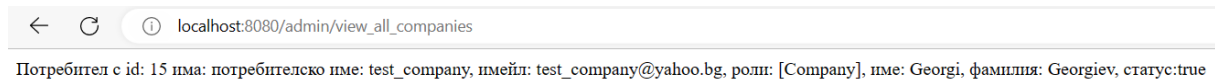
Фигура 4.7 Заявка за всички игрища

- **Търсене на игрище за град Пловдив**



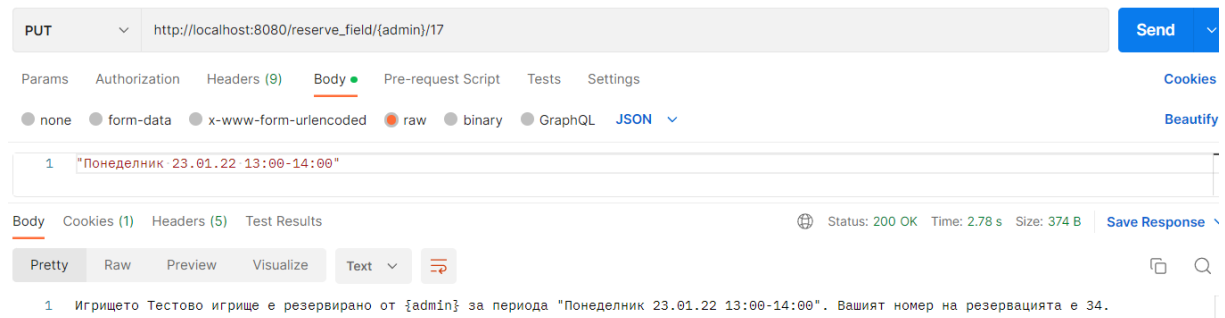
Фигура 4.8 Търсене на игрище по град

- **Достъп на админ до потребителите от тип компания:**



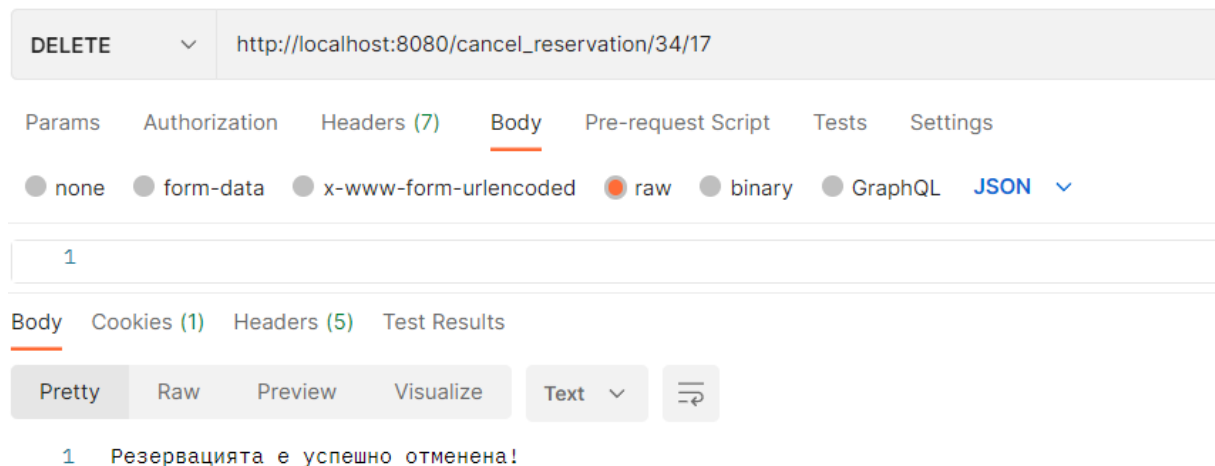
Фигура 4.9 Списък на всички компании

- **Извършване на резервация:**



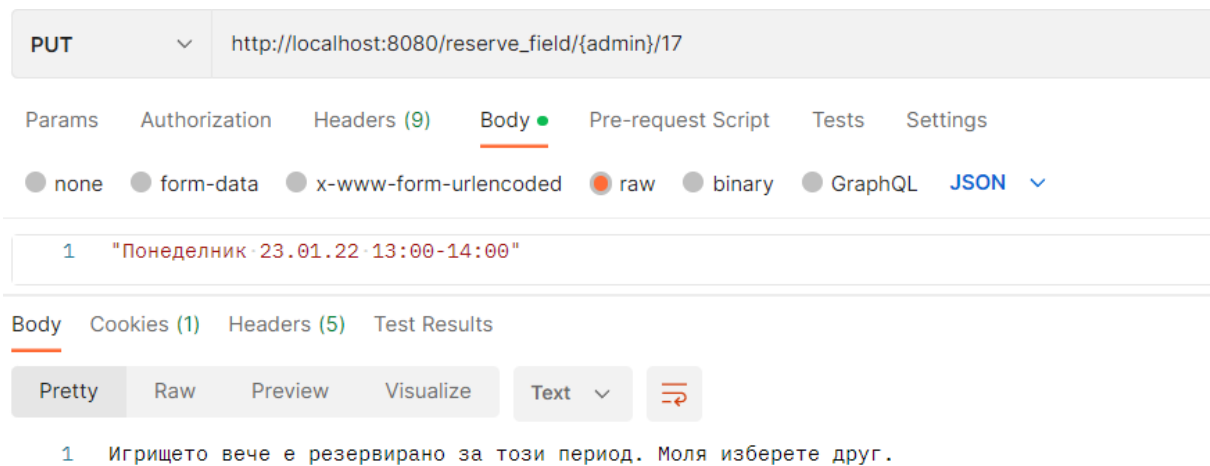
Фигура 4.10 Резервация на игрище

- **Отмяна на резервация:**



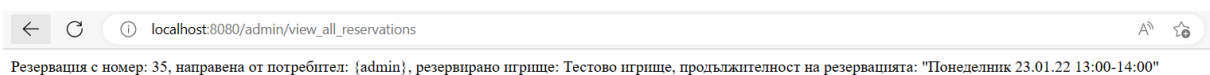
Фигура 4.11 Отмяна на резервация

- **Опит за резервация през вече зает период:**



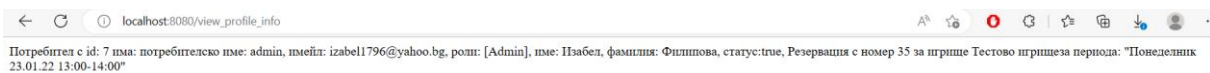
Фигура 4.12 Невалиден период на резервация

- **Преглед на всички резервации от админ:**



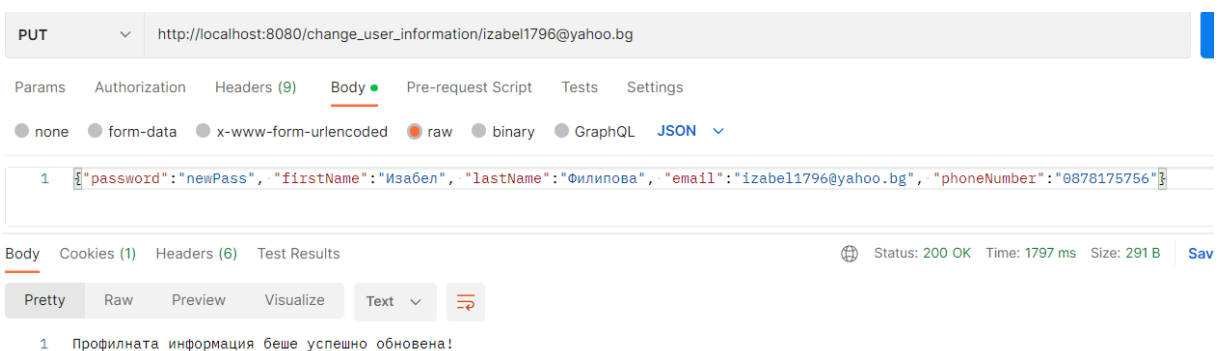
Фигура 4.13 Списък на всички резервации

- **Преглед на профилни данни и направени резервации:**



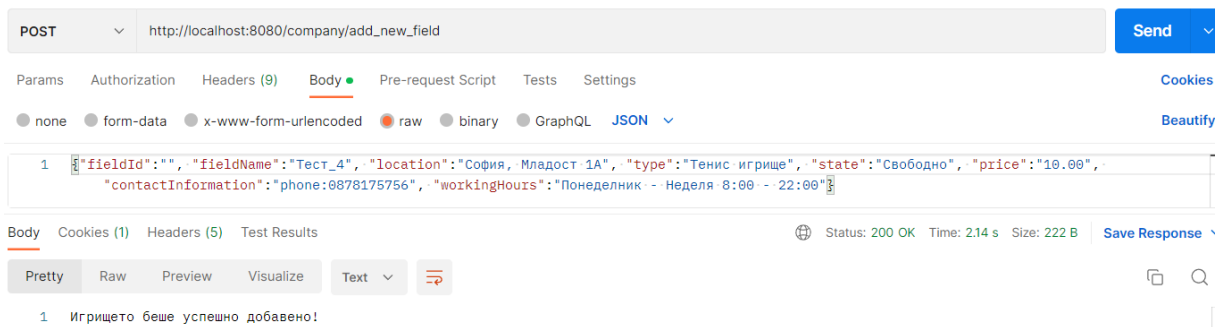
Фигура 4.14 Профил на потребителя

- **Ъпдейт на профилни данни:**



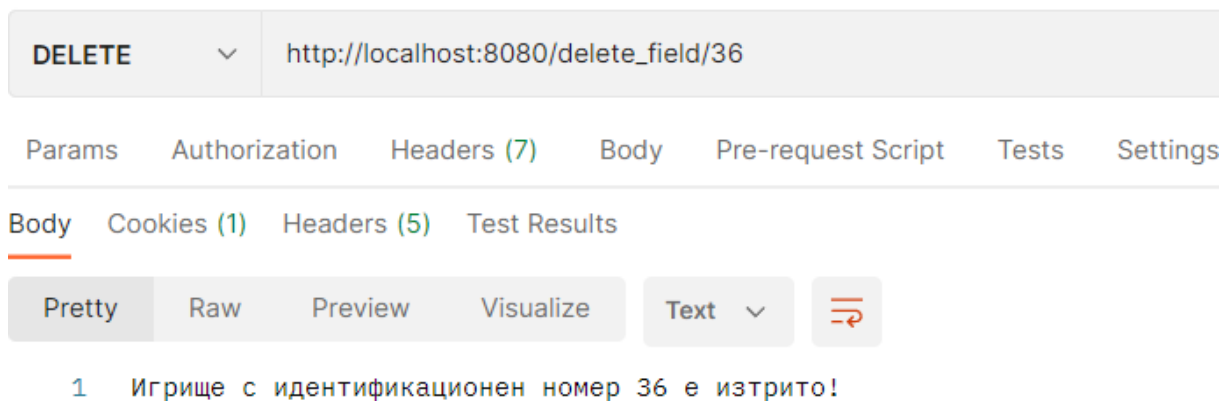
Фигура 4.15 Промяна на потребителски данни

- **Добавяне на игрище:**



Фигура 4.16 Добавяне на игрище

- **Премахване на игрище:**



Фигура 4.17 Изтриване на игрище

4.2 Изводи

Спрямо разгледаната програмна реализация в трета глава, както и експерименталните данни изложени в точка 4.1, може да се види, че всички посочени технологии в дипломното задание са използвани за неговото осъществяване.

Предимствата, които дипломният проект трябва да има, спрямо разгледаните софтуерни продукти в първа глава бяха следните - свободно извършване на резервации, свободната им отмяна, липса на ограничения за локацията на спортните обекти, достъп без нужда от вход и регистрация до данните за спортните игрища, достъп на потребителя до направените от него резервации. Видно от програмната реализация в глава трета те са спазени.

4.3 Варианти за бъдеща оптимизация

Основна бъдеща оптимизация на приложението ще бъде добавянето на фронт-енд интерфейс, чрез който всички операции да се извършват с лекота от потребителя. Това може да включва не само визуално по-приятно представяне на информацията, а и интегриране на модул със седмичен график, съдържащ информация за текущите резервации през който потребителите да могат да направят нова резервация.

Към приложението може да се добави конфигурация с навигация като “Google карти”, така че не просто да излиза адреса на дадено игрище, а потребителят да може веднага да види къде се намира визуално и как може да стигне до него.

Би могла да се добави и платформа за чат, с която потребителите да си комуникират, да разменят информация за различните игрища или да си намерят допълнителен човек за сформирание на отбор.

Използвани литературни източници:

1. **Официален сайт на приложението** - <https://tereni.bg/>
2. **Официален сайт на приложението** - <https://easybook.bg/>
3. **Официален сайт на приложението** - <https://www.sport4all.bg/>
4. **Cosmas, Nwakanma Ifeanyi, C. Etus, I. U. Ajere, and Agomuo Uchechukwu Godswill.** "Online bus ticket reservation system." *Iard International Journal Of Computer Science And Statistics* (2015).
5. **Kwadwo, Tenagyei Edwin, Kwadwo Kusi, and Patamia Agbeshi Rutherford.** "Design and Implementation of Hospital Reservation System on Android." *International Journal of Computer Science and Information Security (IJCSIS)* 17, no. 10 (2019).
6. **Virginus, Ugwu Nnaemeka, Nelson Ogechukwu Madu, Okafor Loveth Ijeoma, Anusiobi Chinenye Loveline, Ugwuanyi Peace Nkiruka, Ndunelo Paul Tobeckukwu, and Ani Chinonso Darlington.** "A Bus Reservation System On Smartphone." *International Journal of Progressive Sciences and Technologies* 25, no. 1 (2021): 240-250.
7. **Techopedia**, "Round Robin Scheduling (RRS)", techopedia, 2018. [Online]. Available: <https://www.techopedia.com/definition/9236/round-robin-scheduling-rrs>
8. **Sanjaykumar, Singh Siddharthkumar, Sheikh Mannan Sohail, Ismail Badri, Mohammed Gadi, and Deeksha Hegde.** "Application for online booking of unreserved ticket for Indian Railways." (2019).
9. **Java Official Documentation** <https://docs.oracle.com/en/java/>
10. **Yilmaz, Rahime, et al.** "Object-Oriented Programming in Computer Science." *Encyclopedia of Information Science and Technology*, Fourth Edition. IGI Global, 2018. 7470-7480.
11. **Mak, Gary.** "Spring MVC framework." *Spring Recipes*. Apress, 2008. 321-393.
12. **Deacon, John.** "Model-view-controller (mvc) architecture." [Online][Citado em: 10 de março de 2006.] <http://www.jdl.co.uk/briefings/MVC.pdf> (2009).
13. **Krasner, G. E., & Pope, S. T.** (1988). A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3), 26-49.

14. **Mumbaikar, Snehal, and Puja Padiya.** "Web services based on soap and rest principles." International Journal of Scientific and Research Publications 3.5 (2013): 1-4.
15. **Cesare Pautasso, Olaf Zimmermann, Frank Leymann,** "RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision" WWW 2008, April 21–25, 2008, Beijing, China. ACM 978-1-60558-085-2/08/04.
16. **Fatna Belqasmi and Roch Glitho, Concordia University Chunyan Fu, Tekelec** "RESTful Web Services for Service Provisioning in Next-Generation Networks: A Survey ",IEEE, IEEE Communications Magazine, December 2011
17. **Deitel, P., & Deitel, H.** (2012). Java How to Program (9th ed.). Pearson Education Limited.
18. **Spring Framework Documentation** - <https://docs.spring.io/spring-framework/docs/3.0.x/spring-framework-reference/html/overview.html>
19. **Garcia-Molina, Hector, Jeffrey D. Ullman, and Jennifer Widom.** "Database systems: the complete book." (2009).
20. **MySQL Official Documentation** - <https://dev.mysql.com/doc/refman/8.0/en>
21. **Hibernate Official Documentation**- <https://hibernate.org/orm/documentation/5.5/>
22. **Eclipse Official Webpage** - <https://www.eclipse.org/ide/>
23. **Postman Official Webpage and Documentation** - <https://learning.postman.com/docs/getting-started/introduction/>
24. **Софтуер за създаване на графики и диаграми** - <https://www.smartdraw.com/software/smartdraw-online.htm>

Списък на използваните съкращения:

ООП – Обектно-ориентирано програмиране

JVM – Java Virtual Machine

JDK – Java Development Kit

SE – Standard Edition

AOP – Aspect Oriented Programming

IoC – Inversion of Control

EE - Enterprise Edition

JDBC - Java Database Connectivity

ORM - Object-relational mapping

OXM - Object XML Mapping

JMS - Java Message Service

POJO - Plain old Java Objects

MVC – Model-View-Controller

AOP - Alliance-compliant aspect-oriented programming

DBMS - Database Management System

SQL - Structured Query Language

IDE - Integrated Development Environment

API - Application Programming Interface

JTA – Java Transaction API

RMI – Remote Method Invocation

IETF – Internet Engineering Task Force

GSS – Generic Security Service

SASL – Simple Authentication and Security Layer

DOM – Document Object Model

AWT – Abstract Window Toolkit

UI - User Interface

REST - Representational State Transfer

URI – Uniform Resource Identifier
XML - Extensible Markup Language
HTTP – Hypertext Transfer Protocol
JNDI - Java Naming and Directory Interface
JPA - Java Persistence API
JDO - Java Data Objects
JAXB - Java Architecture for XML Binding
SAML - Security Assertion Markup Language
ER – Entity Relationship
HRS – Hospital Reservation System
JSON - JavaScript Object Notation
HTML - HyperText Markup Language
QR – Quick Response