

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 разработваното дипломно задание.....	18
II глава. Анализ и описание на използваните технологии.....	20
2.1 Програмен език Java. MVC модел. REST архитектурен стил.....	20
2.1.1 Model-View-Controller модел.....	20
2.1.2 REST архитектурен стил.....	21
2.2 Технологична рамка с отворен код Spring Framework.....	22
2.3 Базы данни. Релационни бази данни.....	24
2.3.1 Релационни бази данни.....	25
2.4 Система за мениджмънт на релационни бази данни MySQL.....	25
2.5 Технологична рамка Hibernate ORM.....	25
2.5.1 Архитектура на Hibernate.....	26
2.6 Среда за разработка на приложения Eclipse Integrated Development Environment.....	28
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 Сървиси (Services).....	33
3.2.4 Контролери (Controllers).....	42
3.2.5 Конфигурация на защитата на приложението (Security/SecuirtyConfig).....	47
3.2.6 ApplicationProperties на приложението.....	51
3.2.7 Диаграма на връзките между обектите (ER – Entity Relationship diagram).....	52
IV глава. Експериментални данни и изводи.....	53
4.1 Експериментални данни.....	53
4.2 Изводи.....	58
4.3 Варианти за бъдеща оптимизация.....	59
Използвани литературни източници.....	60
Списък на използваните съкращения.....	62

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 следващите глави 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.

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.

Накрая 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](http://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 настилка, 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 обекти, 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 обект 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.

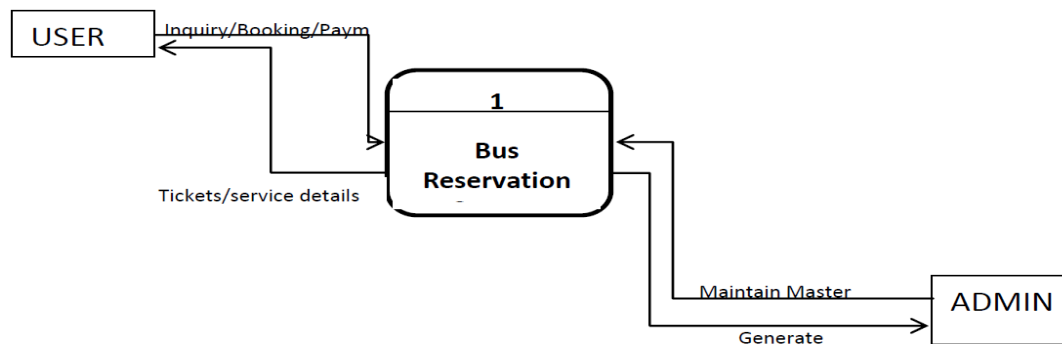
Платформата, както и предходните разполага с график на свободните и заети часове, а резервациите се извършват само от потребители, влезли в профила си. Плащането се извършва чрез кредитна или дебитна карта и няма опция да се извърши на място. След успешно извършено плащане системата генерира билет, който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

1.4 Онлайн система за резервация на автобусни билети

Системата е уеб базирана, като фронт-енд частта на разглеждания софтуер е създадена с помощта на PHP, а за бек-енд частта е използван MySQL. Функционалностите, с които разполага ще бъдат разгледани на няколко нива.

Основните взаимодействия в системата са представени на фиг. 1.1:

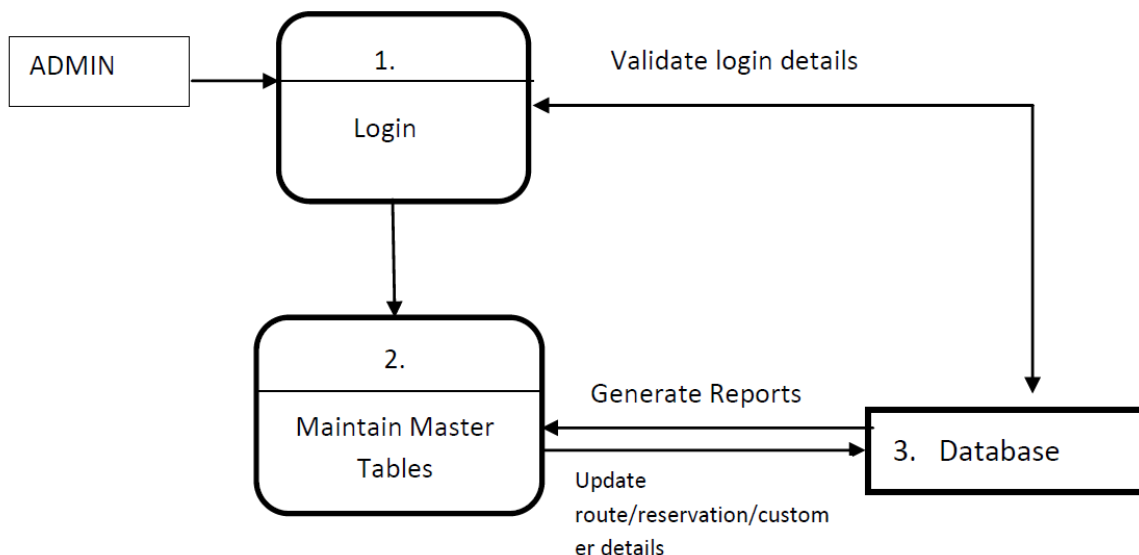
Level 0



фиг. 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 менажира системата, показан на фиг. 1.2.

LEVEL 1

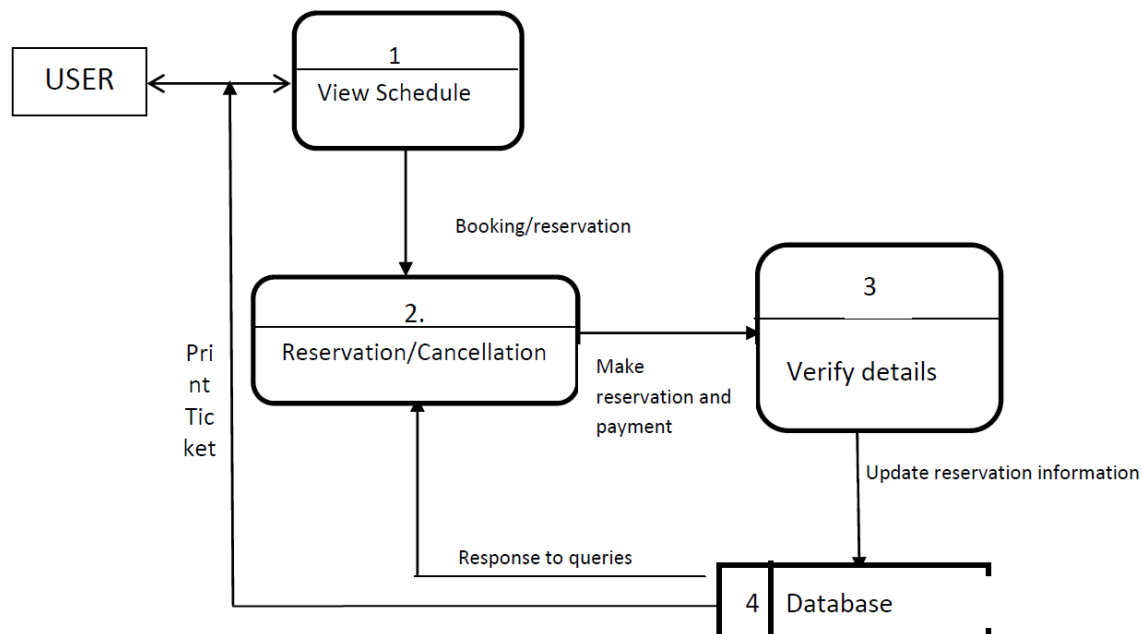


фиг. 1.2 Администраторски функции в системата, [4]

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

Възможностите на потребителя са показани на фиг. 1.3:

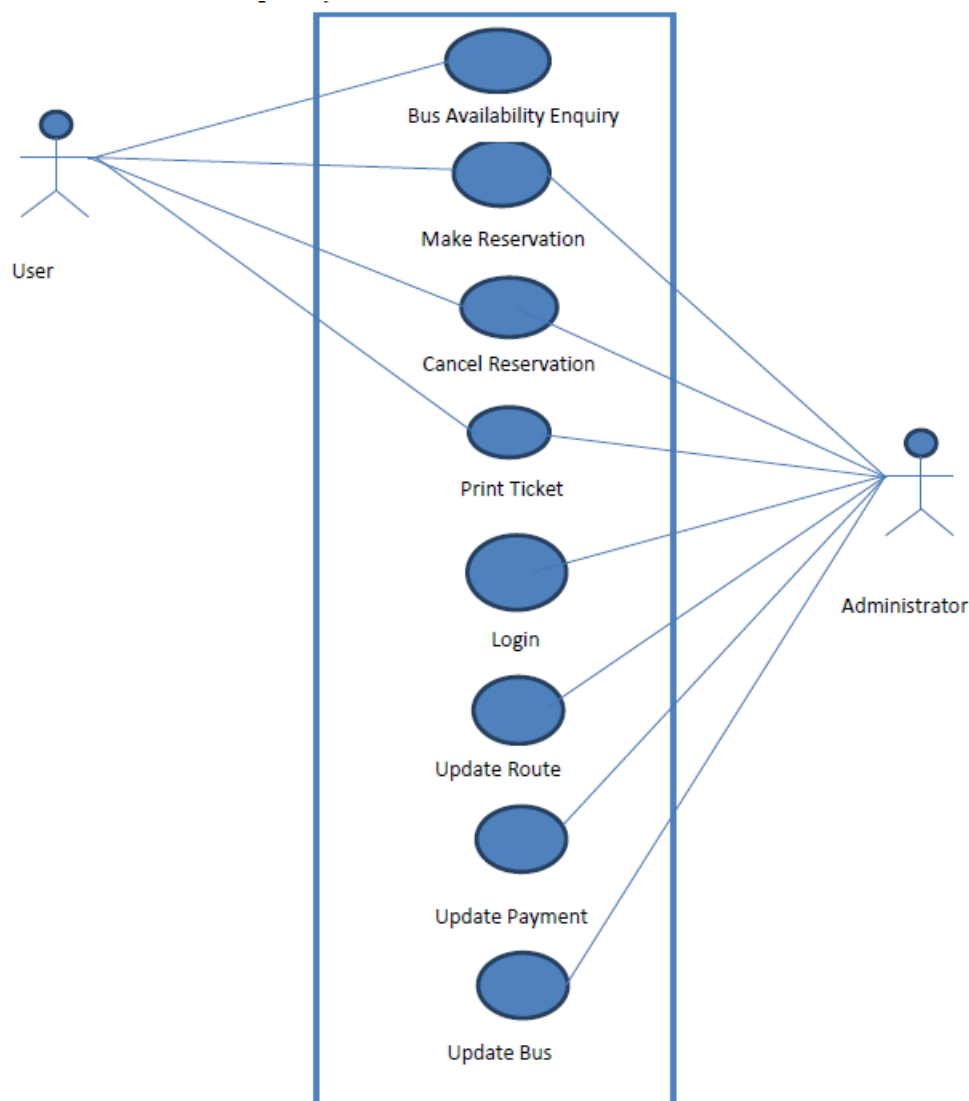
Level 2



фиг. 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 use-case диаграма обобщава функциите, както на потребителя така и на администратора и показва кои от тях се припокриват:



фиг. 1.4 Use-case диаграма за системата, [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 например. [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.

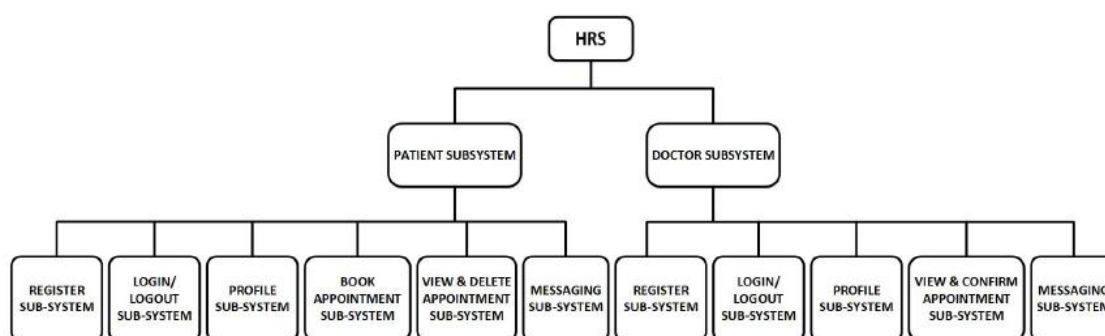
Приложението разполага с панели както за клиентите, така и за медицински лица. Всички имат възможност да се регистрират, като всеки лекар въвежда своя идентификационен номер, който се използва при влизане в системата.

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.

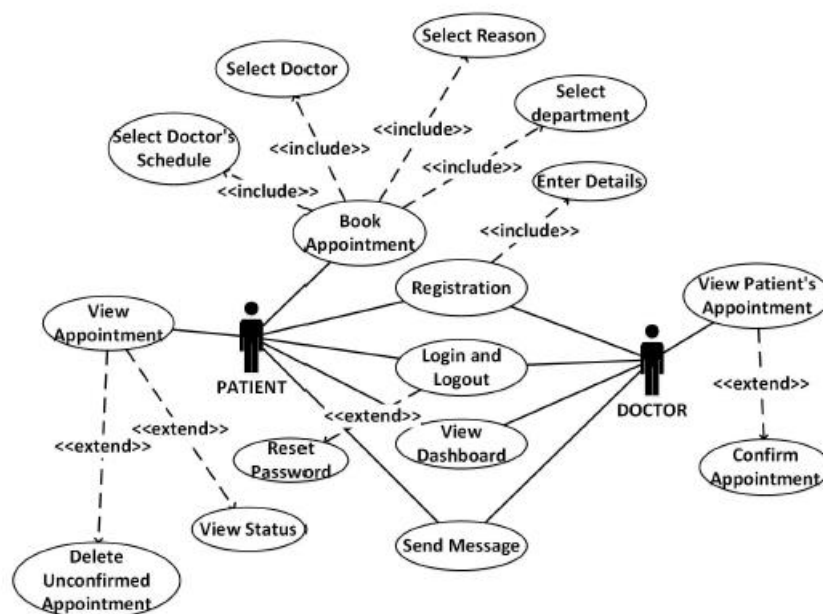
Панелът за лекари от своя страна се състои от профил, секция за преглед и потвърждаване на заявки за записване на час, платформа за съобщения и възможност за изход от профила.

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 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 use-case диаграмата на приложението, както и модулите, от които се състои системата:



фиг. 1.5 Модули, от които се състои системата, [5]



фиг. 1.6 Use-case диаграма на системата, [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 ID 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 желаня 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 отделението, 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.

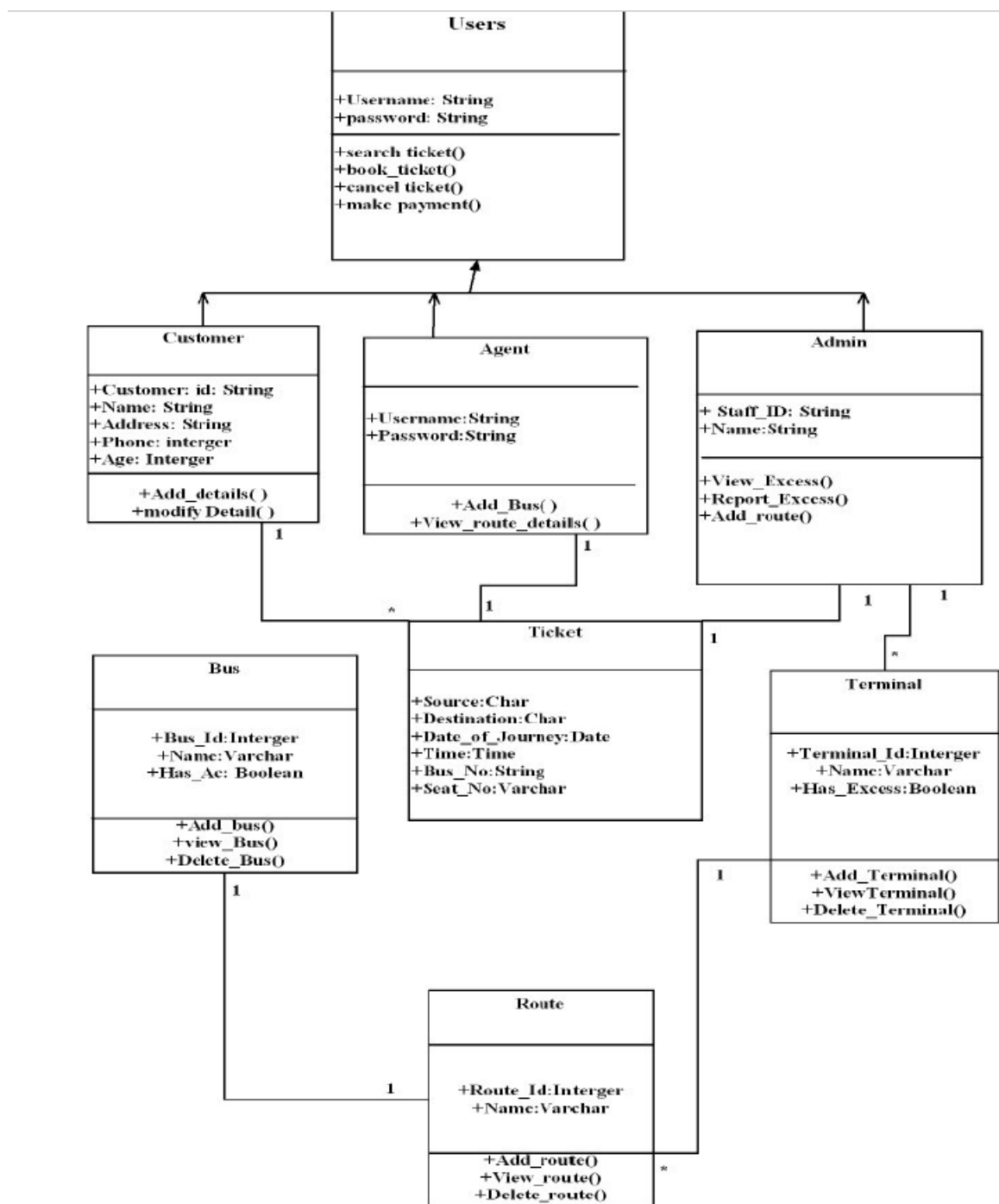
Приложението разполага и с възможност за комуникация между пациенти и лекари, като при изпращане на съобщение се получават и известия. [5]

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 разработено, използвайки PHP, MySQL, JavaScript и HTML.

Използва се 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.

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



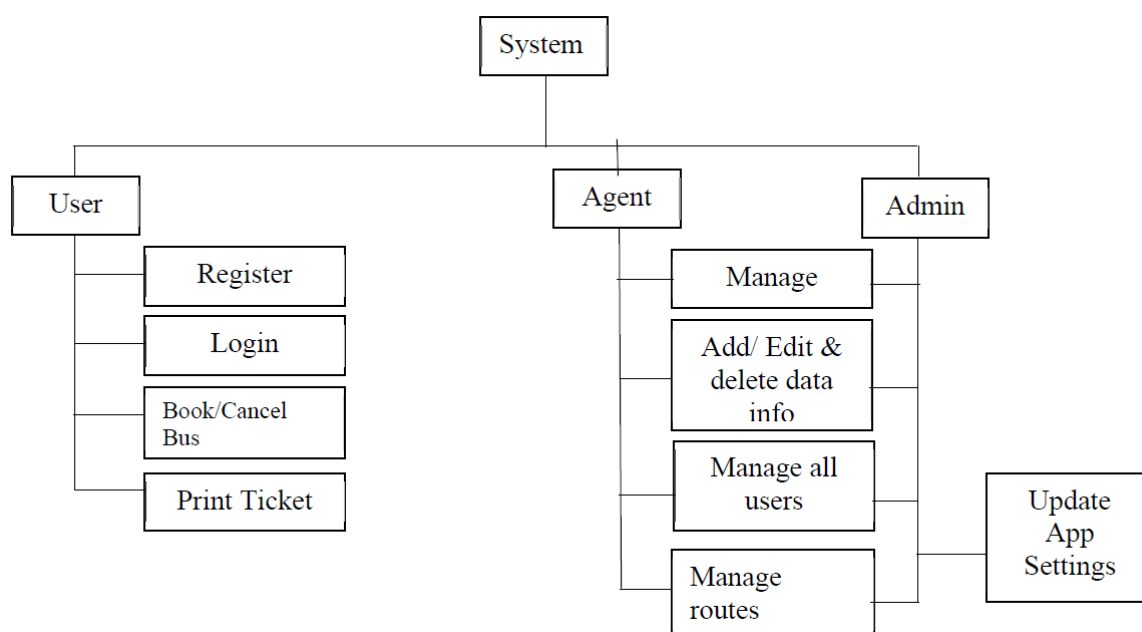
фиг. 1.7 Клас диаграма на приложението, [6]

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 менажира системата. Всички влизат в системата чрез потребителско име и парола.

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

Архитектура на системата може да се види на следващата фигура:



фиг. 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 Индийските 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 отделен сървис – 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 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 прашане на имейл за потвърждение на транзакцията, съдържащ PDK 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.

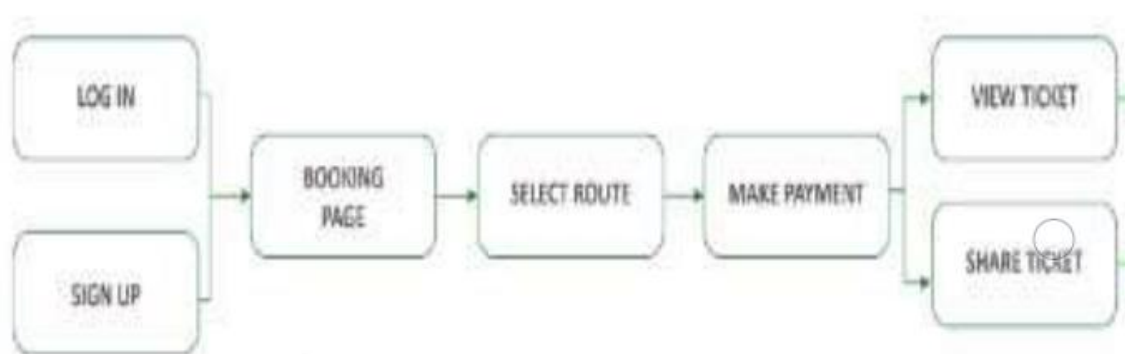
Приложението използва трислойна архитектура:

- **Слой база данни (Database layer)** – в случая е използвана база данни, запазваща информацията в json формат.

- **Сървис слой (Application Service layer)** – тук са разпределени системните функции и бизнес логиката на приложението. Слойът отговаря също и за въвеждане по базата данни, според заявките от най-горно ниво.

- **Слой Клиентски интерфейс (Customer Interface layer)**

Flow диаграмата на приложението е представена на следващата графика:



фиг. 1.9 Flow диаграма на системата, [7]

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 валидира 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 QR код, видим в приложението, като към потребителя се изпраща и имейл с потвърждение. [7]

1.8 Изводи за разгледаните до момента платформи

В предходните параграфи разгледахме различни софтуерни продукти, 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 платорформите – 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 си направиш резервация там. 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 платформата за резервация на автобусни билети от точка 1.6 е липсата на история за извършени плащания.

Системата 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 желаният 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 разработваното дипломно задание

Позовавайки се на разгледаните до момента софтуерни продукти следва да оформим какви възможности трябва да има разработваният дипломен проект.

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 система за резервация на спортни игрища, като тя ще е уеб базирана. Ще се използва 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 и отмяна резервация на игрище. Останалите вариации на потребители ще го наследяват, всеки със специфичните си функционалности. Всички ще имат възможност за вход в системата посредством логин формата на 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. 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, ще могат да се добавят от администратор, а влизайки в профила си ще имат възможност да добавят свои игрища за резервации, 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 и премахва фирми, да вижда пълен списък на регистрираните потребители, както и да трие клиентски профили и спортни игрища.

В следващите глави предстои да бъдат описани и разгледани използваните технологии за разработка на дипломното задание, както и неговата архитектурна 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-ориентиранят 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 използван Spring Security. Основните характеристики на всяка от тези технологии ще бъдат разгледани в следващите параграфи.

2.1 Програмен език Java. MVC модел. REST архитектурен стил.

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.[8-9] Тази универсалност и широка разпространеност на езика е и една от основните причини да бъде избран за разработка на приложението.

2.1.1 Model-View-Controller модел

Model-View-Controller е често използван модел за дизайн при разработка на приложения. Чрез него бизнес логиката се разделя от потребителския интерфейс – UI (User Interface) като се разпределят роли на модел, изглед и контролер 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.

- 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.

- 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.

- Controller

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

- Комуникация Model (Controller) към View

Моделът не трябва да знае за съществуващите view-та. Комуникацията между контролера, модела и view се осъществява, чрез събития, често идващи от контролера. 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. [10-11]

2.1.2 REST архитектурен стил

REST (Representational State Transfer – Репрезентационен трансфер на състояния) 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 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 URI. Използва се стандартизиран интерфейс с фиксирана група от HTTP 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 са също независими.

GET, PUT, POST и DELETE са някои от основните http методи, използвани при този тип приложения за извличане, създаване, промяна и изтриване ресурси. [12 - 13]

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 – Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, Test. Ще разгледаме накратко всяка от тези категории.

- **Core Container**

В този слой се съдържат модулите Core, Beans, Context и Expression Language.

Core и Beans осигуряват фундаментални функции на Spring като Inversion of Control (IoC) и Dependency Injection.

Core модульт предоставя възможност да се достъпват обекти по начин, подобен на JNDI (Java Naming and Directory Interface) регистър. Този модул наследява характеристики от Beans модула и добавя поддръжка за интернационализация, разпространение на събития (event propagation), зареждане на ресурси, и прозрачно създаване на контексти. Модульт поддържа и Java EE (Enterprise Edition) функционалности.

Expression Language модульт осигурява мощен експресивен език за създаване на заявки и манипулация на обектни графи по време на изпълнение на приложението. Езикът поддържа get и set методи за променливите, 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 Spring IoC Container. Поддържа също създаване и селектиране на списъци, както и чести обединения на списъци.

- **Data Access/Integration**

Този слой е съставен от модулите Java Database Connectivity (JDBC), Object-relational mapping (ORM), Object XML Mapping (OXM), Java Message Service (JMS) и Transaction.

JDBC модульт осигурява абстрактен слой, който премахва нуждата от JDBC кодиране и преобразуване на специфични кодове за грешки, свързани с базата данни.

ORM модульт предоставя интеграционни слоеве за популярни интерфейси за обектно-релационен мапинг като JPA (Java Persistence

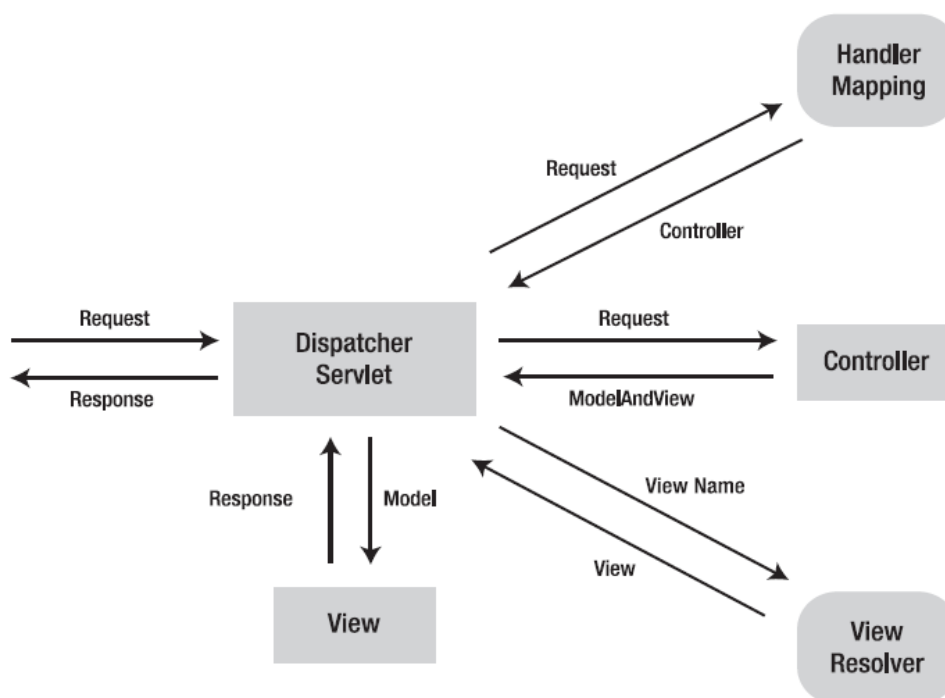
API)), Hibernate, JDO (Java Data Objects) и други, за да могат да се използват заедно с всички функции, предоставени от Spring.

OXM модулет поддържа Обектни/XML мапинг имплементации за JAXB (Java Architecture for XML Binding), Castor, XML Beans и други. JMS съдържа функции за създаване и унищожаване на съобщения.

Transaction модулет поддържа програмен и декларативен мениджмънт на транзакции за класове, които имплементират специални интерфейси и за всички POJO (plain old Java Objects).

- **Web**

Модулет се състои от следните слоеве - Web, Web-Servlet, Web-Struts и Web-Portlet. Web модулет предоставя основни функции като качване на многокомпонентни файлове, инициализиране на IoC контейнера, използвайки servlet listeners и уеб-ориентиран контекст на приложението. Модулет съдържа и свързаните с уеб компоненти на отдалечената поддръжка на Spring. Web Servlet съдържа MVC (Model – View – Controller) имплементацията на Spring за уеб приложения. Графично представяне на моделът може да се види на фигура 2.2.



фиг. 2.1 – Графично представяне на Spring MVC, [10]

Spring-Struts модулет пък се състои от поддържащи класове за интегриране на класически Struts Web tier в дадено Spring приложение.

Web-Portlet модулет осигурява възможност MVC моделът да бъде

използван в портлет среда и като функционалност е подобен на Web-Servlet модула.

- **AOP and Instrumentation**

Този модул на Spring осигурява AOP (Alliance-compliant aspect-oriented programming) имплементация, която дава възможност да се отделят части на от имплементацията на кода, които логически трябва да са разделени.

Тук се съдържа отделен Aspects модул, осигуряващ интеграция с AspectJ.

Instrumentation модулът поддържа организация на класове и classloader имплементации, които да се използват при някои сървъри за приложения.

- **Test**

Модулът поддържа тестването на Spring компоненти с Junit или TestNG. Осигурява непрекъснато зареждане на Spring ApplicationContexts и кеширането на тези контексти. Осигурява и “фиктивни” обекти, с които кодът да се тества изолирано. [14]

2.3 Базы данни. Релационны базы 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. [15]

2.4 Система за мениджмънт на релационни бази данни MySQL

MySQL е система за мениджмънт на релационни бази данни с отворен код, създадена и поддържана от Oracle. Работи с 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. [16]

2.5 Технологична рамка Hibernate ORM

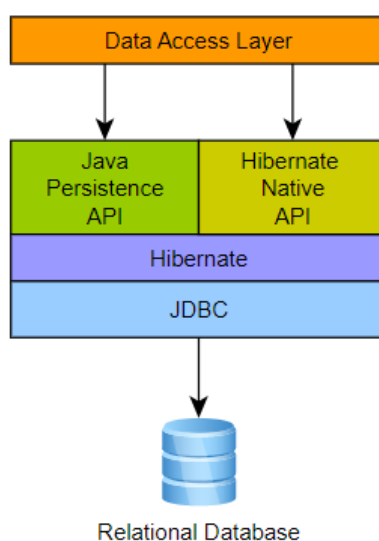
Hibernate ORM осигурява технологична рамка за свързване на обектно-ориентирано приложение към релационна база данни. Hibernate заменя директния достъп до базата данни с функции от високо ниво, обработващи обектите.

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.

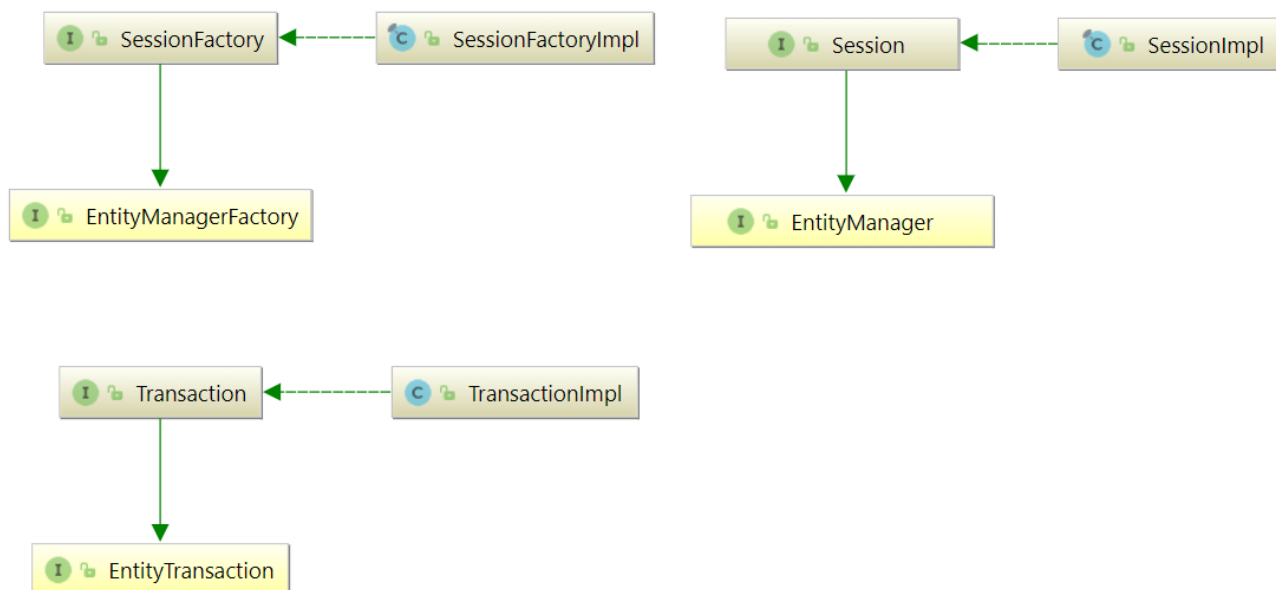
2.5.1 Архитектура на Hibernate

Hibernate като ORM решение стои между нивото за достъп до данни на Java приложението и релационната база данни, както може да се види на фигура 2.3. Приложението използва Hibernate APIs за зареждане, съхранение, заявки и др.



фиг. 2.2 – Връзка между приложението и базата чрез Hibernate, [17]

Hibernate имплементира спецификациите на Java Persistence API, а връзките между двете са представени на диаграмата на фигура 1.3.



фиг. 2.3 – Връзки между Java Persistence API и Hibernate, [17]

Следва да бъдат разгледани основните Hibernate APIs:

- **SessionFactory**

Представяне на мапинга между домейн модела на приложението и базата данни. Играе роля на фабрика за Session инстанции. Поддържа услуги, които Hibernate използва за всички сесии, като кеш памет от второ ниво, пулове за връзка, интеграция на транзакционни системи и др. EntityManagerFactory в JPA е еквивалентът на Sessionfactory в Hibernate и двете се обединяват в еднаква имплементация. SessionFactory е само една за дадено приложение, тъй като е скъпа за създаване.

- **Session**

Представява едно-нишков обект с кратък живот и моделира “Unit of Work”. Сесията обвива Connection в JDBC и играе роля на фабрика за Transaction инстанциите.

- **Transaction**

Обект, използван от приложението за разграничаване на границите на отделните физически транзакции. Еквивалентът в JPA е EntityTransaction и двете играят роля на абстрактно API за да изолират приложението от основно използваната транзакционна система (JDBC / JTA).[17]

2.6 Среда за разработка на приложения Eclipse Integrated Development Environment

За разработка на приложението ще бъде използвана Eclipse IDE (Integrated Development Environment – Интегрирана среда за разработка), 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. [18]

2.7 Платформа с отворен код Postman

Postman представлява платформа с отворен код, даваща възможност за създаване, тестване, мониторинг и разработка на документация за APIs (Application Programming Interface). [19]

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

III глава. Архитектура на програмната реализация

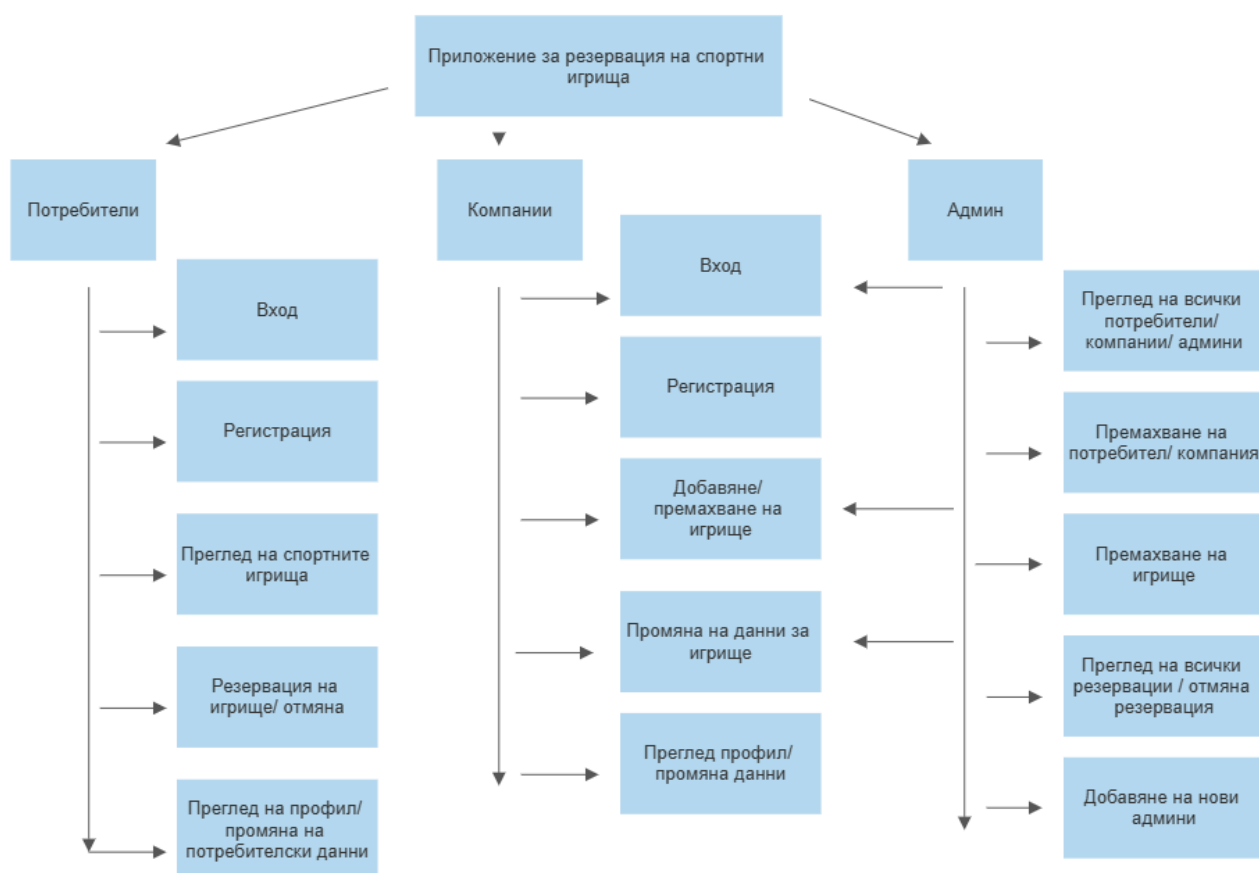
В тази глава следва да бъде разгледана програмната реализация на разработвания дипломен проект, неговите структура, компоненти и функционалности.

За направата на приложението за резервация на спортни игрища са използвани Java 17, Spring Boot 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 SpringSecurity, чрез което се осъществява лог-ин на системата и менажиране достъпа до функциите ѝ, спрямо различните потребители.

3.1 Основна архитектура на приложението

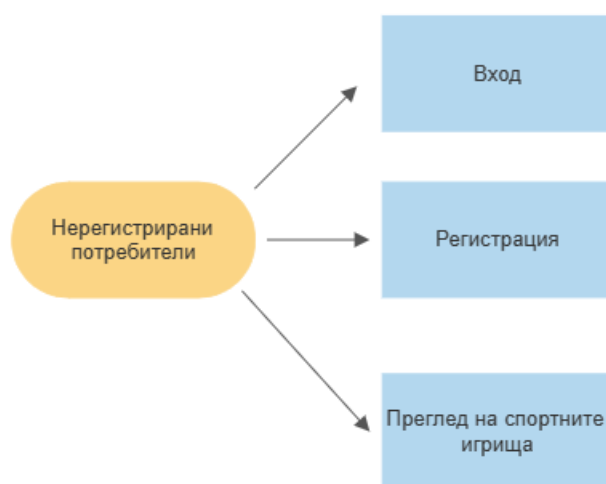
Ще започнем с преглед на основната архитектура на приложението за резервация на спортни игрища, представена на фигура 3.1:



фиг. 3.1 Архитектура на приложението, създадена чрез SmartDraw [20]

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 фиг. 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.



фиг. 3.2 Графика на операции, извършвани от анонимен потребител, създадена чрез SmartDraw [20]

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:

- **Entities** – обектите, необходими за реализация на приложението;
- **Services** – основната логиката на приложението, операции с базата данни;
- **Controllers** – обработка на заявките и връзка между бизнес логиката и базата данни;
- **Repository** – връзка с базата данни;
- **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 – Users (Потребители), Role (Роли), Reservation (Резервации), Field (Спортните игрища).

Users Entity се използва за създаване на потребителите на приложението. Съдържа като променливи основни данни, необходими за всеки потребител – `username` (потребителско име), `password` (парола), `email` (имейл), `phoneNumber` (телефонен номер), `firstName` (име), `lastName` (фамилия), `active`(статус на потребителя). 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.

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

Users и Role класовете са обвързани помежду си с @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 users_roles, в която всеки от двата обекта е представен със своето id.

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.

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.

Всяко Entity разполага с getter/setter методи, 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 **UserRepository** са дефинирани следните методи:

- findByEmail() – търси потребителят по даден имейл и връща резултат от тип Users;
- findByUsername() – търси потребителят по потребителско име и връща резултат от тип Users;
- findById() - търси потребителят по идентификационния му номер и връща резултат от тип Users;
- deleteById() – трие потребителят, посредством идентификатор;
- findByRolesIn() – връща списък с всички потребителите, имащи даден тип роля.

RoleRepository – има метод findByName, търсещ ролята в базата посредством нейното име.

FieldRepository разполага със следните опции:

- deleteById() – изтрива дадено игрище, посредством неговото id;
- FindByFieldId() – открива игрище чрез id;

- `FindByFieldName()` – отива игрището чрез неговото име.

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 Сървиси (Services)

Приложението разполага с четири `service` класа, отговарящи на четирите разгледани `entities` – `UserService`, `RoleService`, `ReservationService`, `FieldService`. Ще разгледаме методите, съдържащи се във всеки от тях.

UserService, съдържа логиката необходима за операциите, свързани с потребителите на приложението – тяхното добавяне, премахване, промяна на данни в базата и други, посредством `userRepository`. Класът разполага със следните методи:

- **`addRegisteredCustomer()`** – чрез този метод се добавят нови потребители от тип `Customer` в базата данни. Програмният код може да се види на фиг. 3.3

Приема като вход обект от тип `Users` с въведени от потребителя 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 `Customer`. Паролата се хешира с помощта на енкодер, за да се осигури нейната сигурност, а за статус на потребителя (`active`) се задава `true`. Клиентът се запазва в базата чрез `userRepository`, 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!";
    }
}

```

фиг. 3.3 метод addRegisteredCustomer, част от UserService

- **addCompany()** – аналогичен на горния метод, чрез който се добавят нови компании. Разликата се състои в добавяната роля – в този случай тя е Company.;
- **addAdmin()** – аналогичен на горния метод, чрез който се добавят нови администратори. Ролята в този случай е Admin.;
- **viewAllCompanies()** – 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;
    }
}

```

фиг. 3.4 метод viewAllCompanies, част от UserService

Прави се първоначална проверка има ли въведени потребители в базата данни, ако поради някаква причина няма такива методът връща

съответен string “Няма регистрирани потребители!”;

- **getAllAdmins()** – аналогичен на `viewAllCompanies`, връща списък на всички администратори;
- **viewAllCustomers()** - аналогичен на `viewAllCompanies`, връща списък на всички клиенти;
- **removeUser()** – метод за изтриване на потребители от базата данни. За целта се използва идентификационният номер на потребителя. Както може да се види от фиг. 3.5 се създава временен обект от тип `Users` - `checkIfExists`, а в базата данни, потребителят се търси по `id`, 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 `checkIfExists` е `null` 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!";
    }
}
```

фиг. 3.5 метод `removeUser`, част от `UserService`

- **changePersonalInformation()** – 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 фиг. 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!";
}
```

фиг. 3.6 метод `changePersonalInformation`, част от `UserRepository`

Входни данни за метода са обект от тип Users, съдържащ променените данни на нашия потребител, както и неговият имейл. Чрез userRepository се търси потребителя в базата и се запазва в customerToEdit. Използвайки get и set методи извличаме новите данни, който сме получили и ги заменяме в нашия потребител, след което го запазваме обратно в базата данни.

- **viewProfileInfo()** – 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!";
    }
}
```

фиг. 3.7 метод viewProfileInfo, част от UserRepository

Следва да разгледаме следващият сървис – **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 фиг. 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();
}
```

фиг. 3.8 методи addRole и getAllRoles, част от RoleService

При добавянето на нова роля, методът получава обект от тип `Role` 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 `IllegalArgumentException` със съобщение, че ролята вече съществува. Ако ли не, тя се запазва. В базата предварително са добавени трите необходими роли – `Customer`, `Admin`, `Company`, но при бъдещо разрастване на приложението и необходимост могат да се добавят нови.

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

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()** – метод за добавяне на нови спортни игрища към базата данни. Получава като вход обект от тип `Field` 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()** – изтриване на игрище чрез негово `id`. `fieldRepository` 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()** – метод, връщащ списък на всички въведени игрища, с допълнителна проверка дали няма никакви въведени такива.
- **getAllFieldsByCity()** – тази функция връща списък на спортните игрища, намиращи се в зададен от потребителя град. Програмната й 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;
}

```

фиг. 3.9 метод getAllFieldsByCity, част от 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. 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.

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

- **getFieldById()** – 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()** – 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);
}

```

фиг. 3.10 метод changeFieldState, част от 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 toEdit. 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()** – функция за резервиране на дадено спортно игрище. Нека разгледаме програмната ѝ реализация на фиг. 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);
    }
}

```

фиг. 3.11 метод reserve, част от FieldEntity

Вход на тази функция са следните променливи – madeBy (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()** – метод за промяна данните на вече съществуващо игрище. Като вход получава обект от тип Field с новите данни, както и id 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/set методи старите данни се заменят с нови, базата се обновява, а потребителят получава съобщение “Игрището е успешно обновено!”.

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

Последният сървис, който ще разгледаме е **ReservationService**, където обработваме операциите по резервация на дадено спортно игрище. Ще разгледаме четирите метода, които се съдържат в този клас – `reserveField()`, `cancelReservation()`, `getReservationHistory()`, `viewAllReservations()`:

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

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

фиг. 3.12 метод `reserveField`, част от `ReservationService`

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

- **cancelReservation()** – 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!";  
    }  
}
```

фиг. 3.13 метод `cancelReservation`, част от `ReservationService`

Функцията приема за вход номера на резервацията и ид на спортното игрище. Посредством `reservationRepository` резервацията се търси чрез нейното `id` и се запазва във временен обект. Извиква се `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()** – метод, който приема за вход потребителско име и връща списък с всички резервации, направени от дадения потребител.
- **viewAllReservations()** – функция, връщаща списък с всички направени резервации

3.2.4 Контролери (Controllers)

За реализация на приложението са използвани два контролера – `MainController` и `UserController`, в които чрез REST заявки и адреси и методите на вече разгледаните сървиси се извършва комуникацията между сървъра и потребителя.

MainController включва методи за преглед на спортните игрища, за добавяне и преглед на роли и хоум станица, 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();
}

```

фиг. 3.14 MainController

Приложението разполага с три метода за достъп до спортните игрища от потребителя – преглед на всички добавени - `getAllFields`, и игрища филтрирани според града, в който се намират или типа на самото игрище. 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 - `User`, `Company`, `Customer`, но е оставена възможност за бъдещо прибавяне на нови. Операциите с роли, могат да се извършват само от администратор.

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);
}

```

фиг. 3.15 UserController, част 1

customerProfile() - 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 userService.

changeUserInformation() – към адреса на метода потребителя дописва имейл адреса си, който се подава към userService, заедно с обект от тип user, съдържащ въведените от потребителя параметри, които ще се променят, заедно с новите им стойности.

removeField() – 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 – потребителското име на клиента, правещ резервацията и идентификационния номер на полето. Потребителя допълнително въвежда стринг с периода на резервация.

cancelReservation() – 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 фиг. 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;
}
```

фиг. 3.16 UserController, част 2

Тук по специфични са само `deleteUser()`, който дава възможност на админа да премахва потребител. За целта към адреса на страницата се добавя идентификатор на потребителя, който да се подаде към `removeUser()` метода от `UserService`.

Функцията `addNewAdmin()` регистрира нов администратор, като този процес може да се извърши само от вече влязъл в профила си админ. За целта потребителя попълва всички данни за новия админ, които се да приемат като `User` обект от заявката и се предават на сървиса.

- 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);
}
```

фиг. 3.17 UserController, част 3

addNewField() приема обект от тип Field с попълнени всички данни за игрището, а changeFieldInformation() променя данните за вече добавено игрище. За целта чрез @RequestBody 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>";
}
```

фиг. 3.18 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 фиг. 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 `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;
}
```

фиг. 3.19 UserController, част 5

3.2.5 Конфигурация на защитата на приложението (Security/SecuirtyConfig)

В тази секция ще бъде разгледана реализацията на класовете CustomUserDetails, CustomUserDetailsService и SecurityConfig, необходими за осигуряване на защитата на приложението.

Да започнем от **CustomUserDetails**, който имплементира UserDetails класът от 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;
    }
}
```

фиг. 3.20 Имплементация на клас 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();
}

```

фиг. 3.21 Имплементация на клас 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 `UserDetails`:

- `getAuthorities()` – тук се извличат ролите, които има потребителя;
- `getPassword()` – извиква се паролата на потребителя;
- `getUsername()` – 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` – проверка дали потребителят е с активен статус.

Следващият необходим клас е CustomUserDetailsService, имплементиращ UserDetailsService от SpringSecurity. Неговият код е представен на фиг. 3.16:

```
1
2@Service
3@Transactional
4public 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}
```

фиг. 3.22 Имплементация на CustomUserDetailsService

Тук се заменя само един метод на UserDetailsService – loadUserByUsername(), 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 UsernameNotFoundException, в противен случай се създава нова инстанция на CustomUserDetails с потребителят от базата данни.

Накрая остава да бъде разгледан SecurityConfig класът, наследяващ WebSecurityConfigurerAdapter 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;
    }
}
```

фиг. 3.23 Имплементация на 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());
}

```

фиг. 3.24 Имплементация на 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();
}

```

фиг. 3.25 Имплементация на SecurityConfig, част 3

Тук за `userDetailsService()` методът се извиква инстанция на `CustomUserDetailsService()`, който беше разгледан в предходните параграфи, така че да използваме промените, направени за текущото приложение, а не класът, който идва по подразбиране със `SpringSecurity`.

За `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 `userDetailService()` и `passwordEncoder()` удостоверява потребителското име и паролата, които потребителят е въвел в лог-ин формата.

В метода `configure(WebSecurity web)` са дефинирани пътищата в приложението, които 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.

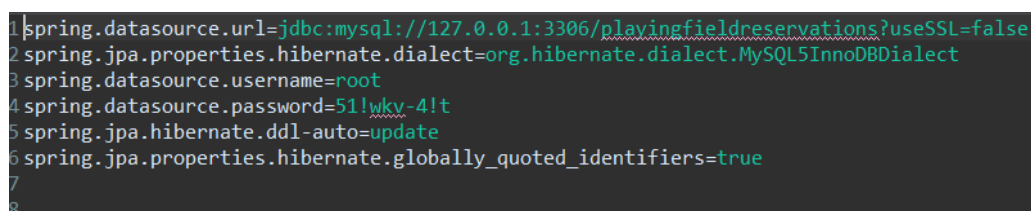
Методът `configure(HttpSecurity http)` е пренаписан, така че бъде разграничен достъпът до отделните ресурси в приложението.

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 и ако тя не отговаря на зададената в конфигурацията чрез `has Authority()`, 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 `hasAnyAuthority`. 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 `formLogin()`.

3.2.6 ApplicationProperties на приложението

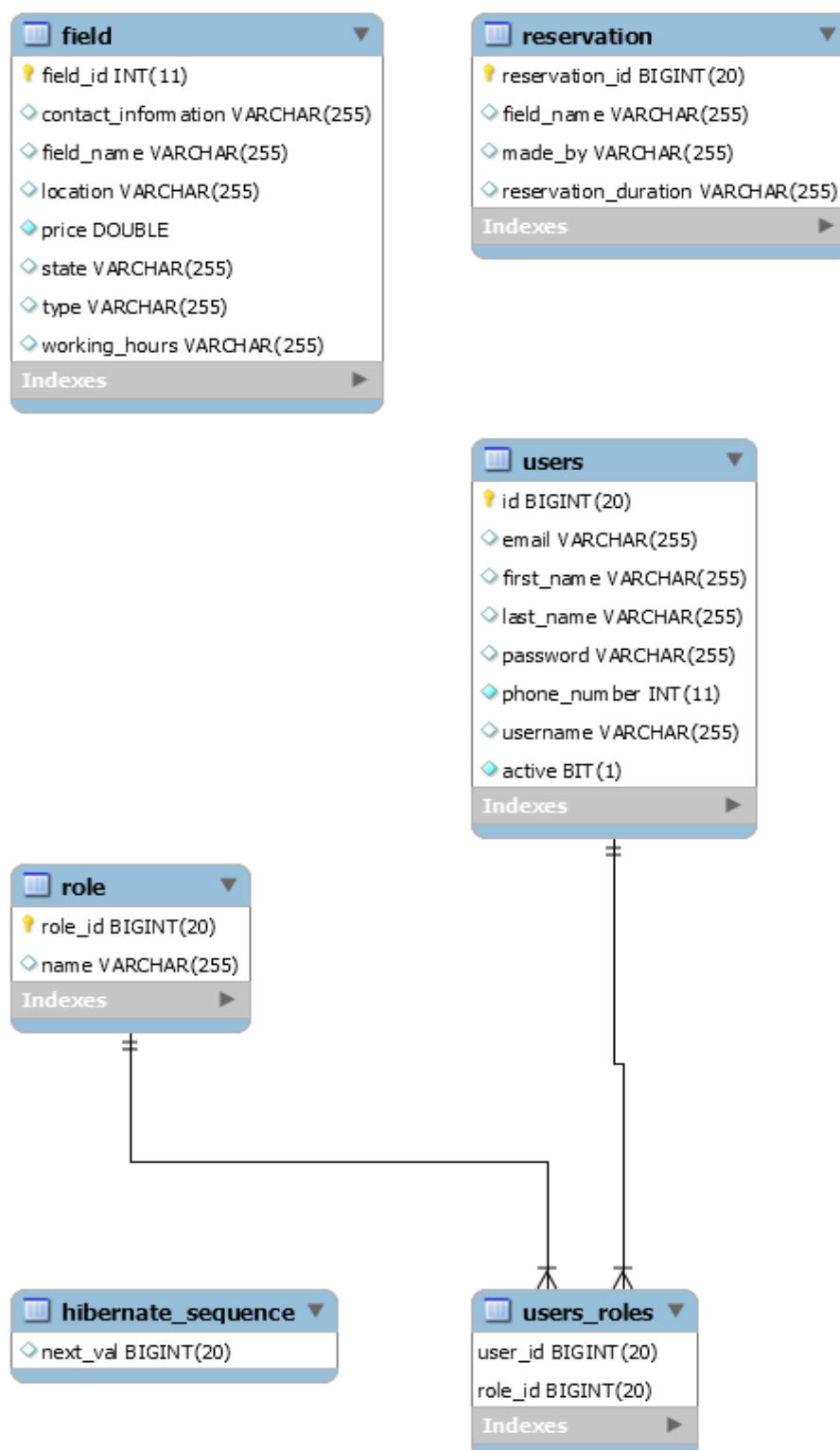


```
1 spring.datasource.url=jdbc:mysql://127.0.0.1:3306/playingfieldreservations?useSSL=false
2 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
3 spring.datasource.username=root
4 spring.datasource.password=51!wky-4!t
5 spring.jpa.hibernate.ddl-auto=update
6 spring.jpa.properties.hibernate.globally_quoted_identifiers=true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

фиг. 3.26 ApplicationProperties на приложението

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

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



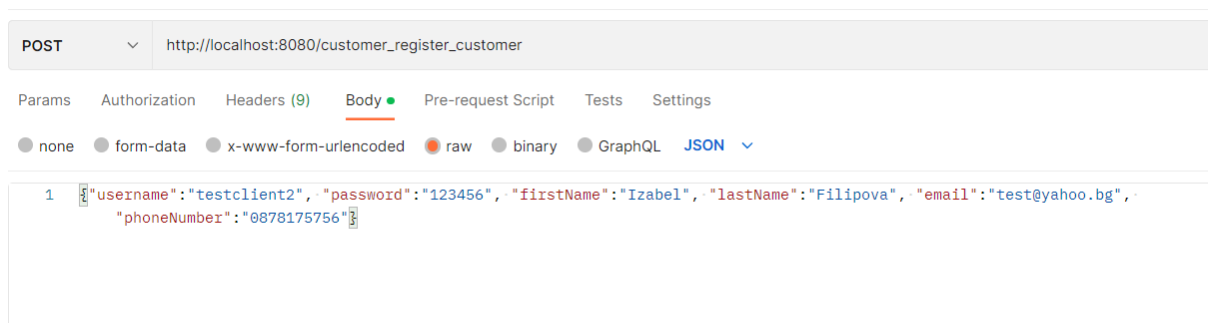
фиг. 3.27 ER диаграма, генерирана чрез MySQL Workbench

IV глава. Експериментални данни и изводи

4.1 Експериментални данни

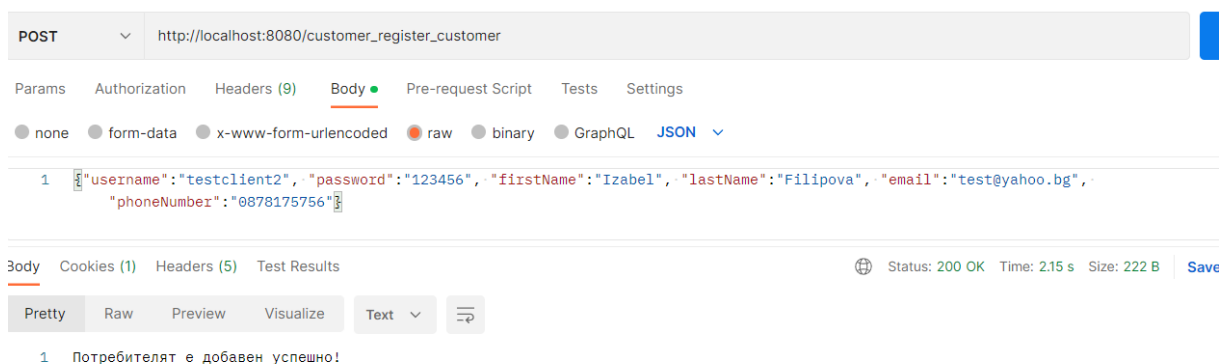
В тази секция нагледно ще бъдат разгледани част от функциите на приложението. Тестването се извършва с Postman за заявки различни от GET, тъй като проектът е на ниво бек-енд и няма как останалите заявки да се обработят директно през браузъра.

- **Регистрация на нов клиент:**



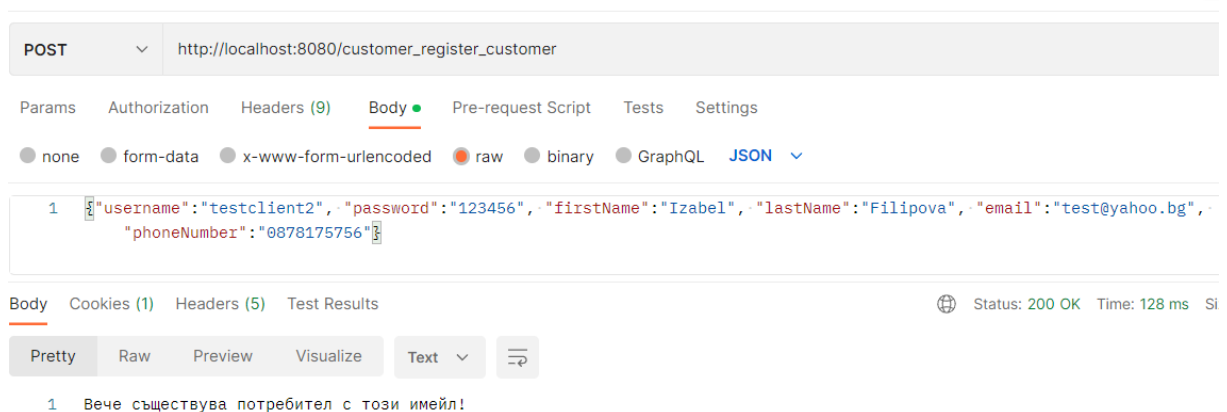
фиг. 4.1 Регистриране на клиент

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



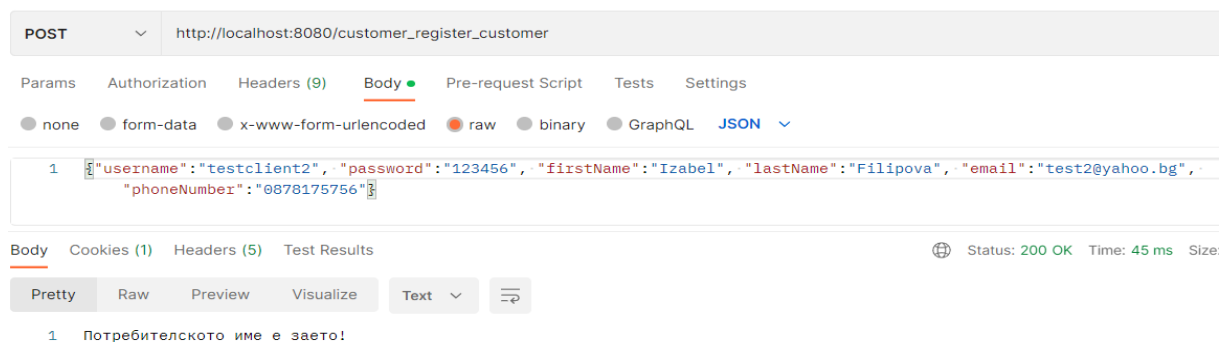
фиг. 4.2 Успешно добавен потребител

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



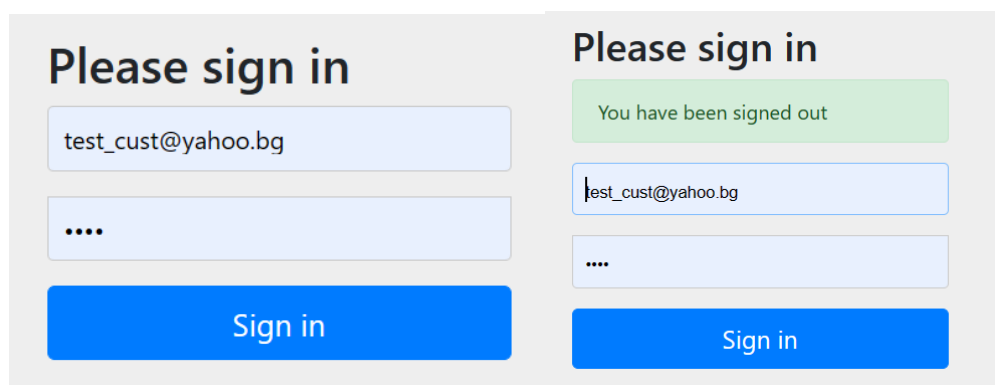
фиг. 4.3 Регистрация с невалиден имейл

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



фиг. 4.4 Регистрация с невалидно потребителско име

- **Вход/изход в приложението чрез formLogin()**

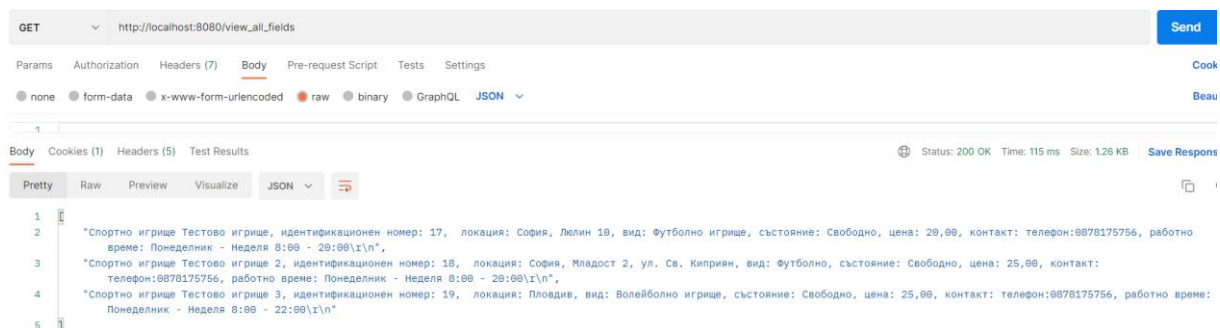


фиг. 4.5 Лог-ин форма

- **Вход с грешни данни:**

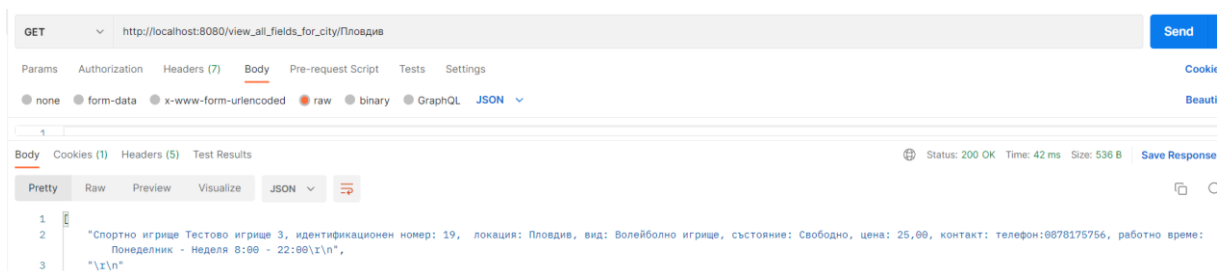
фиг. 4.6 Невалиден имейл или парола

- **Преглед на всички игрища:**



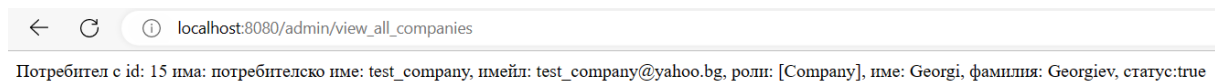
фиг. 4.7 Заявка за всички игрища

- **Търсене на игрище за град Пловдив**



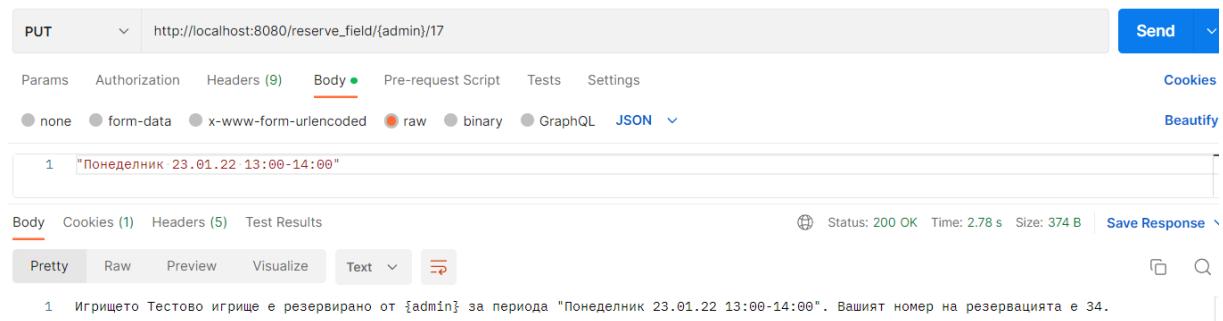
фиг. 4.8 Търсене на игрище по град

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



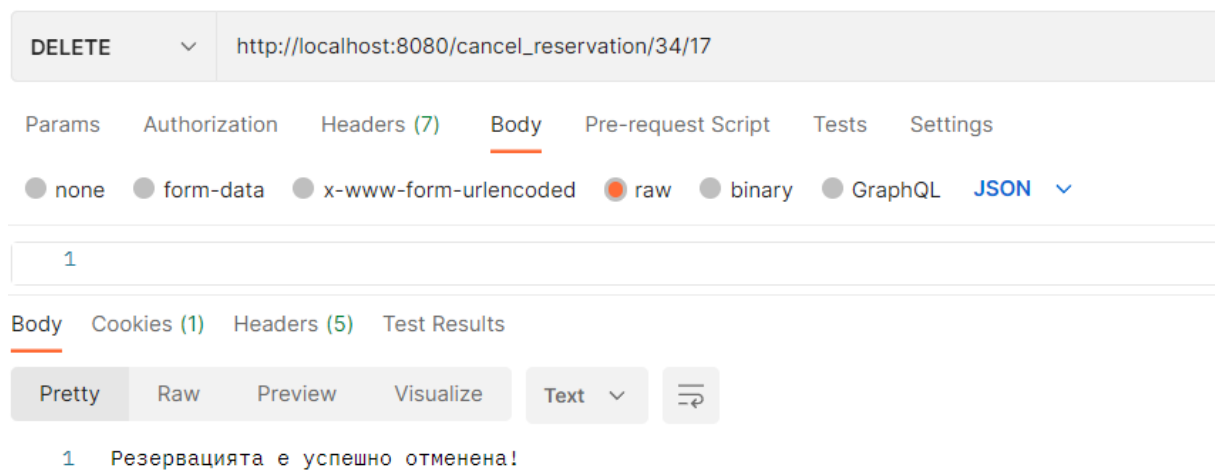
фиг. 4.9 Списък на всички компании

- **Извършване на резервация:**



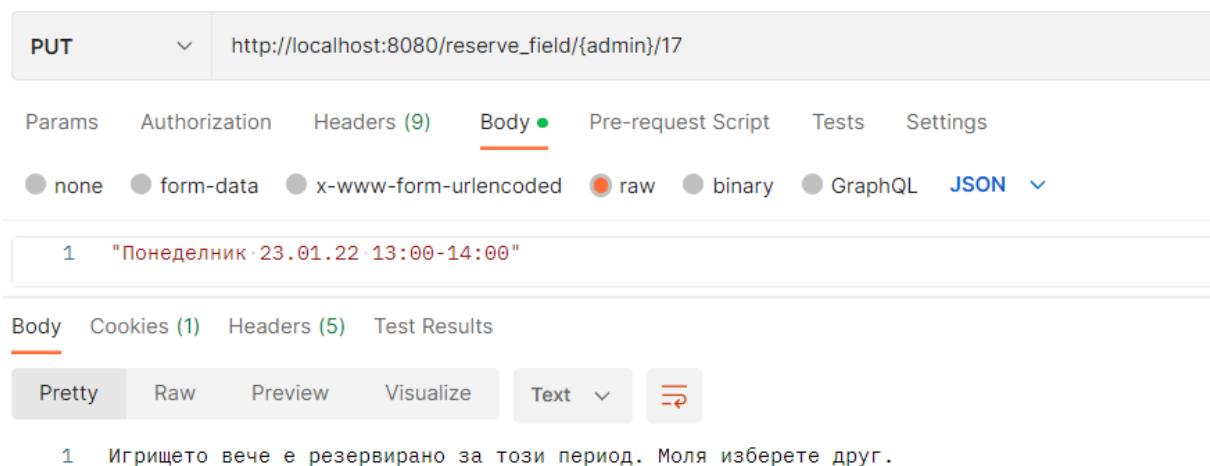
фиг. 4.10 Резервация на игрище

- **Отмяна на резервация:**



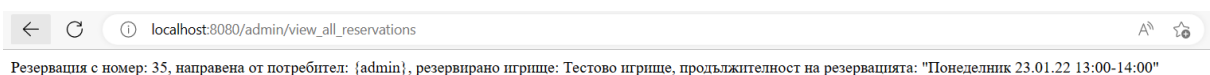
фиг. 4.11 Отмяна на резервация

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



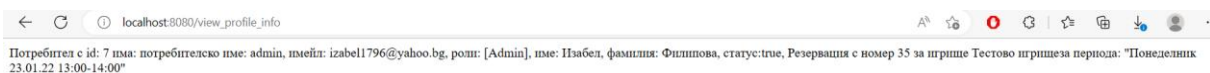
фиг. 4.12 Невалиден период на резервация

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



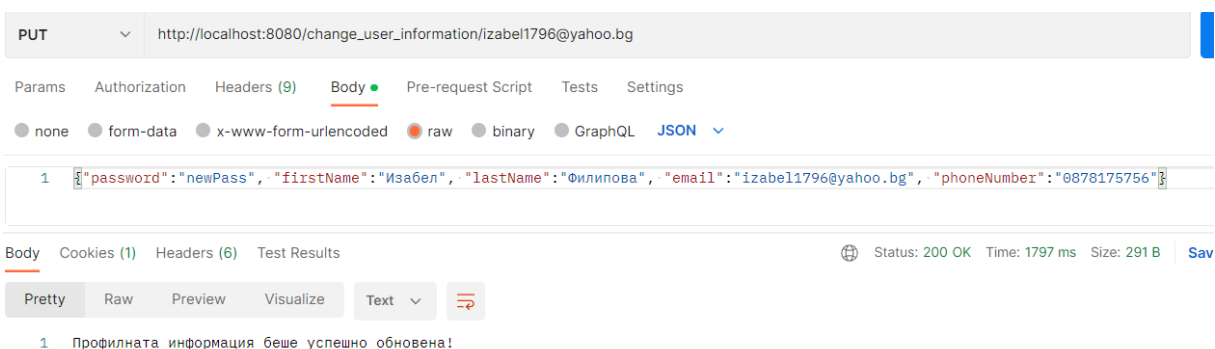
фиг. 4.13 Списък на всички резервации

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



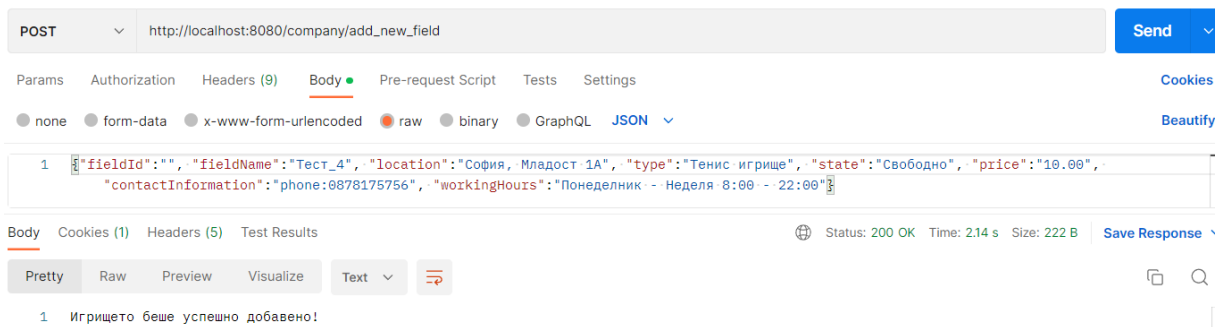
фиг. 4.14 Профил на потребителя

- **Ъпдейт на профилни данни:**



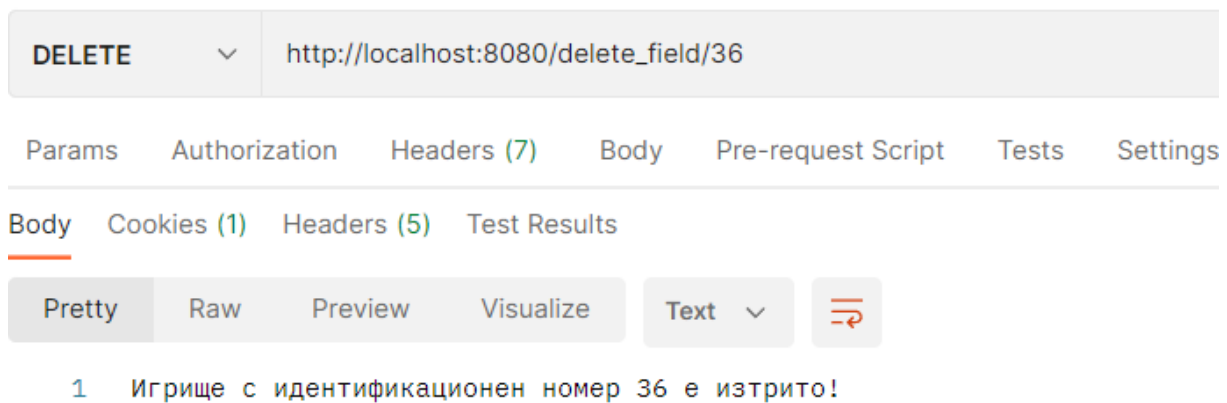
фиг. 4.15 Промяна на потребителски данни

- **Добавяне на игрище:**



фиг. 4.16 Добавяне на игрище

- **Премахване на игрище:**



фиг. 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 зададени 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 три 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 например на модул със седмичен график, даващ информация за текущите резервации и през който потребителите да могат да направят нова резервация.

Към приложението може да се добави конфигурация с 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." *liard 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 Tobechukwu, and Ani Chinonso Darlington.** "A Bus Reservation System On Smartphone." *International Journal of Progressive Sciences and Technologies* 25, no. 1 (2021): 240-250.
7. **Sanjaykumar, Singh Siddharthkumar, Sheikh Mannan Sohail, Ismail Badri, Mohammed Gadi, and Deeksha Hegde.** "Application for online booking of unreserved ticket for Indian Railways." (2019).
8. **Java Official Documentation** <https://docs.oracle.com/en/java/>
9. **Yilmaz, Rahime, et al.** "Object-Oriented Programming in Computer Science." *Encyclopedia of Information Science and Technology*, Fourth Edition. IGI Global, 2018. 7470-7480.
10. **Mak, Gary.** "Spring MVC framework." *Spring Recipes*. Apress, 2008. 321-393.
11. **Deacon, John.** "Model-view-controller (mvc) architecture." *Online*[Citado em: 10 de março de 2006.] <http://www.jdl.co.uk/briefings/MVC.pdf> (2009).
12. **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.
13. **Deitel, P., & Deitel, H.** (2012). *Java How to Program* (9th ed.). Pearson Education Limited.

14. **Spring Framework Documentation** - <https://docs.spring.io/spring-framework/docs/3.0.x/spring-framework-reference/html/overview.html>
15. **Garcia-Molina, Hector, Jeffrey D. Ullman, and Jennifer Widom.** "Database systems: the complete book." (2009).
16. **MySQL Official Documentation** - <https://dev.mysql.com/doc/refman/8.0/en>
17. **Hibernate Official Documentation**- <https://hibernate.org/orm/documentation/5.5/>
18. **Eclipse Official Webpage** - <https://www.eclipse.org/ide/>
19. **Postman Official Webpage and Documentation** - <https://learning.postman.com/docs/getting-started/introduction/>
20. **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

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