**1. Първа 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 опциии 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 три системи– две за резервация на автобусни билети, и една за запазване на часове за преглед в болница. Техният принцип на работа и функционалности силно се припокриват с начинът, по-който трябва да функционира приложението за резервации на спортни съоръжения.

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

**1.1 Платформа за резервации Tereni.bg**

Tereni.bg е платформа за резервация на футболни игрища, намиращи се в град София. Предоставя основна информация за това дали обектът е закрит или открит, каква е настилката, какви са неговите размери, каква е цената за резервация. Разполага със снимки на обектите, визуализация на тяхната локация чрез google maps 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 съоражения.

**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:



фиг. 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.



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:



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 тръгване 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 й 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 желания 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 на 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 Система за резервация на самолетни билети, използваща единична самоличност, базирана на безжична комуникация (NFC – Near Field Communication)**

**1.8 Сравнение на разгледаните платформи**

На база разгледаните до момента софтуерни продукти следва да оформим какви възможности трябва да има разработваният дипломен проект.

Ще бъде разработена бек-енд частта на система за резервация на спортни игрища, като тя ще е уеб базирана. Ще се използва Java, Spring, релационна база данни.

Потребителите на приложението ще са разделени на три основни роли – 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 собствен профил или регистрация, като за целта ще се използва системата за мениджмънт Keycloak.

Клиентите свободно ще могат да разглеждат качените в приложението спортни игрища и информацията, свързана с тях. 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, както и да трие клиентски профили и спортни игрища.

Плюсът на приложението е, че ще бъде изцяло безплатно, в него няма да се извършват плащания, ако има такива, те ще бъдат направени на място в самия обект.

Свободно ще могат да се добавят обекти, които са както платени, така и със свободен достъп, няма да има ограничение в тяхната локация. Освен онлайн резервация клиентът ще има достъп и до телефон на отговорното лице за конкретното игрище за директна комуникация и допълнителни въпроси. Под този формат освен удобство за запазване на часове, приложението ще се оформи и като добра информационна платформа за наличните спортни игрища.

**II глава. Анализ и описание на използваните технологии**

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

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

Java e 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.[7-8] Тази универсалност и широка разпространеност на езика е и една от основните причини да бъде избран за разработка на приложението.

**2.1.2** **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. [9-10]

**2.1.3 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 сетовете 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. [11]

**2.2 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 модулът 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. Модулът поддържа и Java EE (Enterprise Edition) функционалности.

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

ORM модулът предоставя интеграционни слоеве за популярни интерфейси за обектно-релационен мапинг като JPA (Java Persistance API)), Hibernate, JDO (Java Data Objects) и други, за да могат да се използват заедно с всички функции, предоставени от Spring.

OXM модулът поддържа Обектни/XML мапинг имплементации за JAXB (Java Architecture for XML Binding), Castor, XMLBeans и други.

JMS съдържа функции за създаване и унищожаване на съобщения.

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

* **Web** Модулът се състои от следните слоеве - Web, Web-Servlet, Web-Struts и Web-Portlet. Web модулът предоставя основни функции като качване на многокомпонентни файлове, инициализиране на IoC контейнера, използвайки servlet listeners и уеб-ориентиран контекст на приложението. Модулът 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. 2.2 – Графично представяне на Spring MVC, [9]

Spring-Struts модулът пък се състои от поддържащи класове за интегриране на класически Struts Web tier в дадено Spring приложение.   
 Web-Portlet модулът 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. [12]

**2.3 Бази данни. Релационни бази данни**

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

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

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

**2.3.1 Релационни бази данни**  
 При този тип бази данни, данните се представят под формата на двумерни таблици, наречени релации. Колоните в таблицата се наричат атрибути, а редовете представляват отделните инстанции на обекта. За всеки ред от релацията е зададен конкретен първичен тип данни като String или int. Не е позволено да се използва тип данни, чиито стойности могат да се разбият на по-малки компоненти.   
 Всеки атрибут също има свой първичен тип данни, наречен домейн, който може да бъде включен в описанието на схемата. Релациите представляват сетове от инстанции на даден обект. Съответно атрибутите на една релация могат да се подредят по различни начини, без да бъде променена самата релация.

**Ключове**

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

Съществуват и външни ключове, които се използват връзка между две отделни релации. Копие от първичният ключ от едната релация се включва към структурата на втората релация, за която той се явява външен. [13]

**2.4 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. [14]

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

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

**2.5.1 Архитектура на 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. 2.3 – Връзка между приложението и базата чрез Hibernate, [15]

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



фиг. 2.4 – Връзки между Java Persistance API и Hibernate, [15]

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**

Представлява 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).[15]

**2.6 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. [16]

**2.7 Postman**

Postman представлява платформа с отворен код, даваща възможност за създаване, тестване, мониторинг и разработка на документация за APIs (Application Programming Interface). [17]

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

**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." Iiard 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] Java Official Documentation https://docs.oracle.com/en/java/

[8]Yilmaz, Rahime, et al. "Object-Oriented Programming in Computer Science." Encyclopedia of Information Science and Technology, Fourth Edition. IGI Global, 2018. 7470-7480.

[9]Mak, Gary. "Spring MVC framework." Spring Recipes. Apress, 2008. 321-393.

[10]Deacon, John. "Model-view-controller (mvc) architecture." Online][Citado em: 10 de março de 2006.] http://www. jdl. co. uk/briefings/MVC. pdf (2009).

[11]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.

[12] – Spring Framework Documentation - https://docs.spring.io/spring-framework/docs/3.0.x/spring-framework-reference/html/overview.html

[13] Garcia-Molina, Hector, Jeffrey D. Ullman, and Jennifer Widom. "Database systems: the complete book." (2009).

[14] MySQL Official Documentation - https://dev.mysql.com/doc/refman/8.0/en

[15] Hibernate Official Documentation- https://hibernate.org/orm/documentation/5.5/

[16] Eclipse Official Webpage - https://www.eclipse.org/ide/

[17] Postman Official Webpage and Documentation <https://learning.postman.com/docs/getting-started/introduction/>

**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