**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.[1] 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. [3-4]

**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. [5]

**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 - взета от [3]

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. [6]

**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. [7]

**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. [8]

**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 – взета от [9]

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 – взета от [9]

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).[9]

**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. [10]

**2.7 Postman**

Postman представлява платформа с отворен код, даваща възможност за създаване, тестване, мониторинг и разработка на документация за APIs (Application Programming Interface). [11]

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