МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль)/специализация)

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка компьютерной модели управления процессом записи сотрудников на медкомиссию»

Обучающийся А.А. Горбатюк
(И.О. Фамилия) (личная подпись)

Руководитель М.А. Тренина
(ученая степень, звание, И.О. Фамилия)

Консультант к.п.н., доцент Т.С. Якушева
(ученая степень, звание, И.О. Фамилия)

Тольятти 2023

## Аннотация

Выпускная квалификационная работа посвящена следующей теме: «Разработка компьютерной модели управления процессом записи сотрудников на медкомиссию». Создание приложения для автоматизации процессов по получения услуг является очень актуальным направлением разработки программного обеспечения в наше время. Целью данной работы является разработка и программная реализация модели управления процессов записи сотрудников на медкомиссию.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- проанализировать существующие модели управления процессом записи сотрудников на медкомиссию;
- выделить требования, к приложению на основе потребностей пользователей;
- создать архитектуру приложения UML-диаграммы, на основе потребностей пользователей;
- выполнить программною реализацию выбранной модели управления процессом записи сотрудников на медкомиссию;
- провести тестирование разработанного приложения;
- сформировать вывод о применимости разработанной модели управления процессом записи на медкомиссию в реальных условиях работы организаций.

В первой главе выпускной квалификационной работы произведён анализ и выбор уже существующих моделей, затем были проанализированы потребности пользователей и составлены требования, разработана архитектура и UML-диаграммы. Вторая глава посвящён логическому проектированию. В третьей главе описывается процесс реализации и тестирования программного продукта.

## Abstract

The title of the graduation work is: «Development of a computer model for managing the registration of employees for a medical examination».

Creating applications to automate the processes of obtaining services is a very relevant area of software development.

The aim of the work is the development and software implementation of a model for managing the process of registering employees for a medical examination.

To achieve this goal, it is necessary to perform the following tasks:

- analyze existing models for managing the process of registering employees for a medical examination;
- highlight the requirements for the application based on user needs;
- create a UML diagrams and application architecture based on user needs;
- perform software implementation of the selected model for managing the process of registering employees for a medical examination;
- test the developed application;
- draw a conclusion about the applicability of the developed model for managing the process of recording for a medical examination in the real working conditions of organizations.

In the first chapter of the graduation work, an analysis and selection of existing models were made, then user needs were analyzed and requirements were drawn up, architecture and UML diagrams were developed. The second chapter deals with the creation of a logical data model. The third chapter describes the process of implementing and testing a software product.

**Оглавление**

# Введение

Основная идея систем, автоматизирующих процесс получения каких-либо услуг, заключается в значительном упрощении жизни людям, а также в экономии времени. В данный момент во многих сферах процесс записи сотрудников на медкомиссию осуществляется традиционными способами, подход оказывает негативное влияния на рабочие процессы, например, если у преподавателя в конкретный день должны состояться медкомиссия, которую невозможно отменить или перенести, и учебные занятия, то, конечно, у него возникнут сложности, в связи с пропущенными занятиями.

Для решения описанной выше проблемы будет разработана модель управления процессом записи сотрудников на медкомиссию, а также будет разработано приложение, с помощью которого каждый сотрудник сможет выбрать удобный для себя день и пройти медкомиссию таким образом, чтобы не нарушать рабочие процессы. Приложение поможет решить ряд проблем, связанных с прохождением медкомиссии, таких как неудобные даты посещений медицинской организации, долгие очереди из-за неравномерного распределения сотрудников по доступным датам и необходимость личного присутствия сотрудника для оформления заявки.

В современном мире, где всё больше людей предпочитают самостоятельно решать свои вопросы, модель самообслуживания может стать ключевым элементом конкурентоспособности для многих компаний и организаций. Основываясь на этом, создание подобных приложений может иметь большой потенциал в коммерческом плане, поскольку многие организации и компании, занимающиеся оказанием услуг, могут заинтересоваться внедрением подобных систем самообслуживания для своих клиентов.

Актуальность бакалаврской работы обусловлена тем, что она позволяет рассмотреть различные аспекты проектирования и разработки программного обеспечения в контексте решения конкретной задачи в области

здравоохранения и корпоративной медицины. Кроме того, тема является актуальной с точки зрения использования современных технологий и подходов, таких как самообслуживание и онлайн-запись, которые становятся все более распространенными в бизнес-сфере.

Также данная работа может иметь практическое значение для компаний, которые заботятся о здоровье своих сотрудников и хотят оптимизировать процесс прохождения медкомиссий.

Цель данной работы – разработать и реализовать модель управления процессом записи сотрудников на медкомиссию.

Для достижения поставленной цели необходимо решить следующие задачи:

1. изучить существующие подходы к управлению процессом записи на медкомиссию в организациях и проанализировать их преимущества и недостатки;
2. разработать требований к программному обеспечению для управления процессом записи на медкомиссию, учитывая особенности работодателя и потребности сотрудников;
3. разработать модель управления процессом записи на медкомиссию, опираясь на существующие подходы и требования;
4. выполнить программною реализацию модели управления процессом записи сотрудников на медкомиссию;
5. провести тестирование разработанного приложения;
6. сформировать вывод о применимости разработанной модели управления процессом записи на медкомиссию в реальных условиях работы организаций.

Первая глава работы посвящена исследование предметной области, выявлению основных требования к разрабатываемой модели. Используя полученные результаты, была создана функциональная модель различных элементов системы и проведено моделированию ключевых сценариев. В

конце были определены задачи, которые нужно будет решить при разработке системы.

Во второй главе было выполнено моделирование процессов передачи данных и взаимодействия компонентов системы, а также создана логическая модель данных.

В третьей главе были исследованы различные архитектурные решения для прикладных программных продуктов и выбраны технологии для хранения данных и разработки программного обеспечения. На основе выбранной СУБД была создана физическая модель данных. Затем была выполнена программная реализация компонентов системы, основанная на выбранной архитектуре, и после завершения реализации было произведено тестирование приложения.

Результатом работы является реализованная модель управления процессом записи сотрудников на медкомиссию в виде Rest API, а также web-интерфейс.

# Глава 1 Разработка модели управления процессом записи сотрудников на медкомиссию

## 1.1 Анализ существующих моделей управления процессом записи сотрудников на медкомиссию

Существует несколько моделей управления процессом записи сотрудников на медкомиссию:

- ручной процесс – сотрудник должен связаться с отделом, ответственным за медкомиссию, и выбрать удобное время для прохождения медицинского обследования;
- процесс с использованием электронной почты – сотрудник отправляет запрос на запись на медкомиссию по электронной почте, после чего отдел, ответственный за медкомиссию, связывается с ним для уточнения деталей и назначения даты;
- автоматизированный процесс с использованием онлайн-сервиса – сотрудник авторизуется на сайте или в приложении, система предоставляет список доступных дней, из которых пользователь выбирает удобную для себя дату.

Модель «автоматизированного процесса» является более эффективной, чем предыдущие две модели, так как она позволяет сотрудникам организации самостоятельно выбрать удобную дату и время прохождения медкомиссии через специальное приложение или веб-интерфейс. Это уменьшает нагрузку на отдел, ответственный за медкомиссию, и увеличивает удобство для сотрудников.

Таким образом, модель «автоматизированного процесса» является более предпочтительной для компаний, которые хотят повысить эффективность управления процессом записи сотрудников на медкомиссию и обеспечить удобство для своих сотрудников.

## 1.2 Определение требований к разрабатываемой системе

Для выявления требований к программному модулю были проанализированы пожелания представителей университета, проведен анализ существующих систем записи в медицинские учреждения, произведено ее моделирование.

Непосредственно от представителей университета были выделены следующие требования к конечному продукту:

- возможность выбора даты посещения;
- наличия доступа у администратора или вышестоящего руководителя добавлять или изменять дату прохождения медкомиссии, в случае, когда сотрудник не имеет возможности самостоятельно это сделать;
- инициация службы-алгоритма автоматического распределения сотрудников по свободным датам посещения медкомиссии;
- возможность создания расписания медкомиссии администратором;
- ограничение для отдела на количество сотрудников, проходящих медкомиссию в конкретный день – не более 51% от численности отдела;
- наличие серверной части приложения (API);
- наличие web-интерфейса приложения.

На основе общих требований необходимо сформировать более точные, для лучшего понимания функционала конечной системы, для чего было произведено моделирование процессов взаимодействия с ней пользователей.

Перед созданием модели было необходимо определиться с методологией, которая будет использована для моделирования процессов. На сегодняшний день существует большое количество методологий проектирования информационных систем, самыми популярными из которых являются SADT, BPMN, UML.

Методология SADT представляет собой совокупность методов, правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области. Функциональная модель SADT отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями. SADT в первую очередь направлена на моделирование бизнес-процессов предприятия, моделирование взаимодействий, потоков данных [2]. Ее целесообразно использовать непосредственно на этапе анализа. Для моделирования программной реализации не является оптимальным выбором.

Спецификация BPMN описывает условные обозначения и их описание в XML для отображения бизнес-процессов в виде диаграмм бизнес-процессов. BPMN ориентирована как на технических специалистов, так и на бизнес-пользователей. BPMN поддерживает набор концепций, необходимых для моделирования бизнес-процессов. Моделирование иных аспектов находится вне зоны внимания BPMN [15].

UML представляет собой систему обозначений, которую можно применять для объектно-ориентированного анализа и проектирования. Его можно использовать для визуализации, спецификации, конструирования и документирования программных систем. Является наиболее универсальным инструментом, так как содержит большой список видов диаграмм, с использованием которых производится описание компонентов системы.

Используя различные инструменты моделирования, решаются разные задачи, поэтому перед выбором инструмента необходимо определить, какие задачи должны быть решены. В данной работе необходимо было разработать конкретную модель, поэтому главным акцентом было проектирование и реализация этой модели, а не детальный анализ бизнес-процессов и предметной области. Учитывая эти факторы, наиболее подходящим инструментом для решения задачи был выбран UML, который специально предназначен для разработки программного обеспечения, универсален и

содержит необходимые средства для моделирования, эта методология была использована для дальнейшего моделирования системы [6].

### 1.3 Постановка задачи на разработку

В рамках работы было необходимо реализовать систему онлайн-записи сотрудников на медкомиссию, которая бы позволяла сотрудникам выбирать удобную для себя дату посещения медицинского учреждения для прохождения медкомиссии из web-браузеров.

Общие функциональные возможности системы:

- выбор наиболее удобной даты посещения из предложенных;
- просмотр информации о планируемой медкомиссии;
- просмотр истории посещений медкомиссий;
- изменение запланированного посещения медицинской организации, с ограничением – не позднее, чем за 5 дней, до назначенной даты.

Специфичный функционал для сотрудника-руководителя:

- выбор и изменение даты прохождения запланированной медкомиссии за подчинённого;
- информационная таблица со списком подчинённых и информации о датах прохождения медкомиссий.

Специфичный функционал для администратора:

- добавление/удаление пользователей;
- просмотр истории всех медкомиссий пользователей;
- добавление и изменение расписания посещений медицинской организации сотрудниками;
- запуск службы-алгоритма автоматического распределения сотрудников по свободным датам посещения медкомиссии запуск обновления базы данных.

Необходимо реализовать серверную часть и web-интерфейс с возможностями руководителя и администратора.

Первоочередной задачей является проведение анализа функциональности проектируемой системы и взаимодействия пользователей с ней, а также моделирование системы и процессов, которые в ней происходят. Кроме того, необходимо изучить входные данные, с которыми система будет работать.

Далее следует построение модели данных, определение связей между основными сущностями и выбор наиболее подходящего типа, структуры и физической реализации системы хранения данных.

В конечном итоге требуется определить архитектуру системы и взаимодействие ее компонентов, изучив применяемые технологии и выбрав наиболее подходящие для данного проекта. После этого можно приступить к разработке системы с последующей корректировкой и промежуточным тестированием. Все эти этапы должны быть выполнены в соответствии с научными принципами и методами.

### 1.4 Моделирование системы

В первую очередь был проведён анализ областей ответственности для пользователей с разными ролями. Для наглядного представления была использована диаграмма прецедентов (рисунок 1), на которой изображены основные действующие лица и их функции. Важно отметить, что руководитель наследует доступ ко всем функциям сотрудника, и также обладает дополнительными правами по управлению подчинёнными. Администратор же имеет доступ ко всем функциям приложения, включая запуск дополнительных процессов. В то же время, сотрудник никак не взаимодействует с управляющей частью приложения.

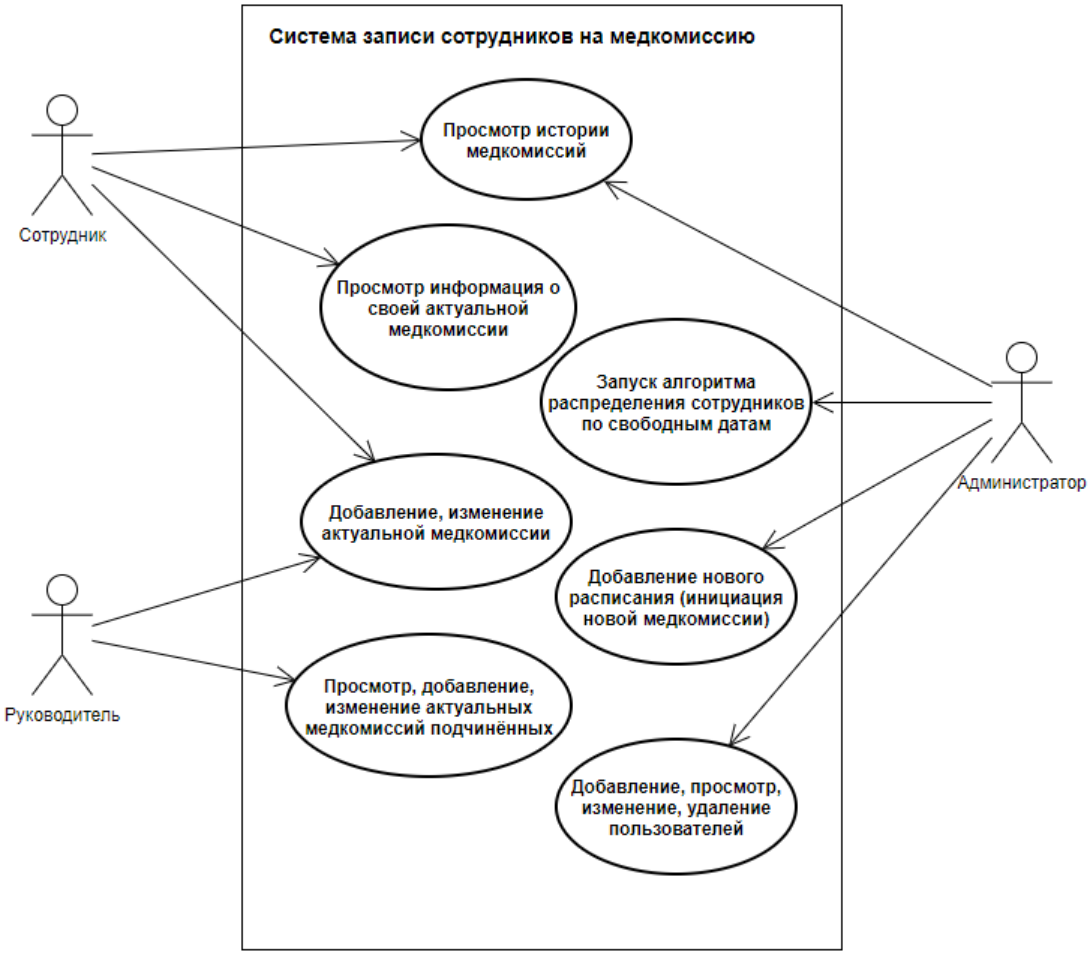


Рисунок 1 – Диаграмма прецедентов взаимодействия пользователей

Основной задачей системы является запись сотрудника на медкомиссию, эту функцию используют сотрудники, а также переиспользуют руководители, инициируя запись как для себя, так и для подчинённых. В случае манипуляций, со стороны руководителя, с добавлением или изменением даты прохождения медкомиссии подчинённым, система проверят руководителя по иерархии, так как такую операцию может совершить только непосредственный начальник или вышестоящий руководитель.

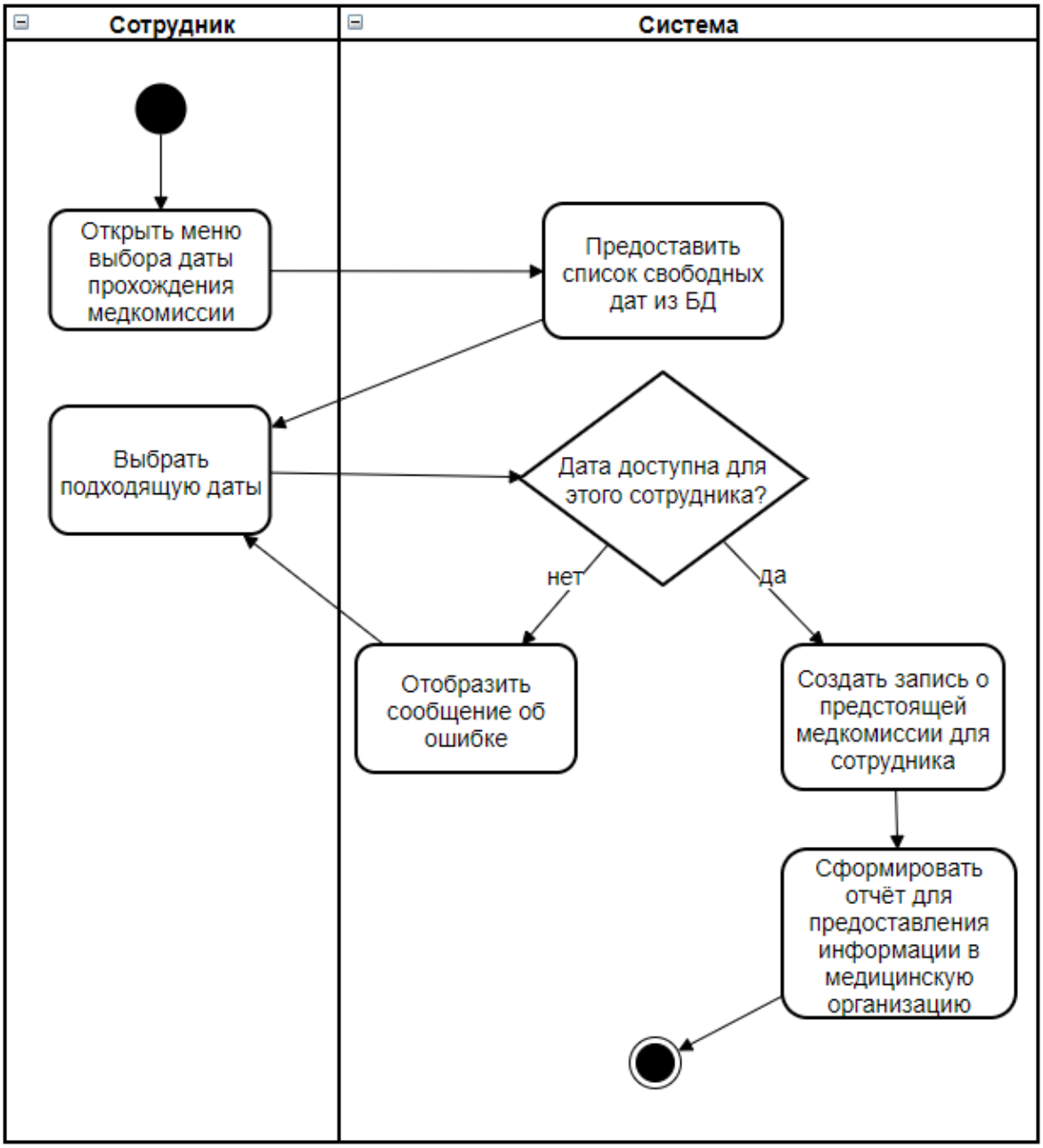


Рисунок 2 – Диаграмма видов деятельности «Запись на медкомиссию»

Условие доступности даты для сотрудника проверяет:

– количество человек из отдела сотрудника, записавшихся на выбранную дату, не должно превышать 50%;
– в момент создания записи количество сотрудников, записавшихся на выбранный день, не превышает установленные медицинской организацией рамки.

## 1.5 Разработка алгоритма распределения сотрудников на медкомиссию

Уникальной возможностью разрабатываемой системы является возможность запуска процесса равномерного распределения не записавшихся сотрудников на медкомиссию. Процесс с выполнением данного алгоритма инициируется непосредственно администратором системы.

Алгоритм учитывает все свободные дни, проверяя упомянутые ранее ограничения касающиеся количества сотрудников, одновременно записавшихся на конкретную дату из одного отдела и максимальное количество людей, которых может принять медицинская организация в день.

Блок-схема разработанного алгоритма представлена на рисунках 3 и 4.

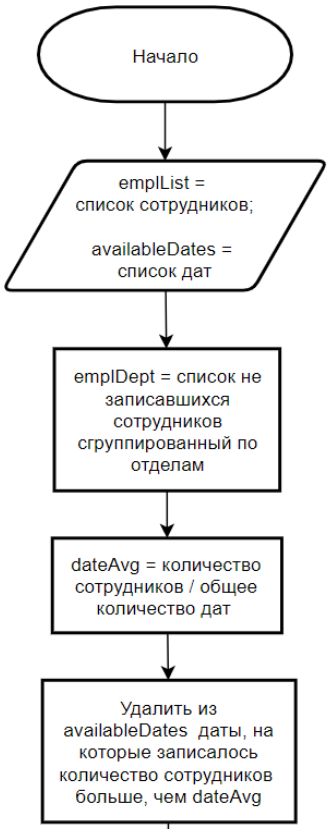


Рисунок 3 – Блок-схема алгоритма равномерного распределения не записавшихся сотрудников на медкомиссию (Часть 1)

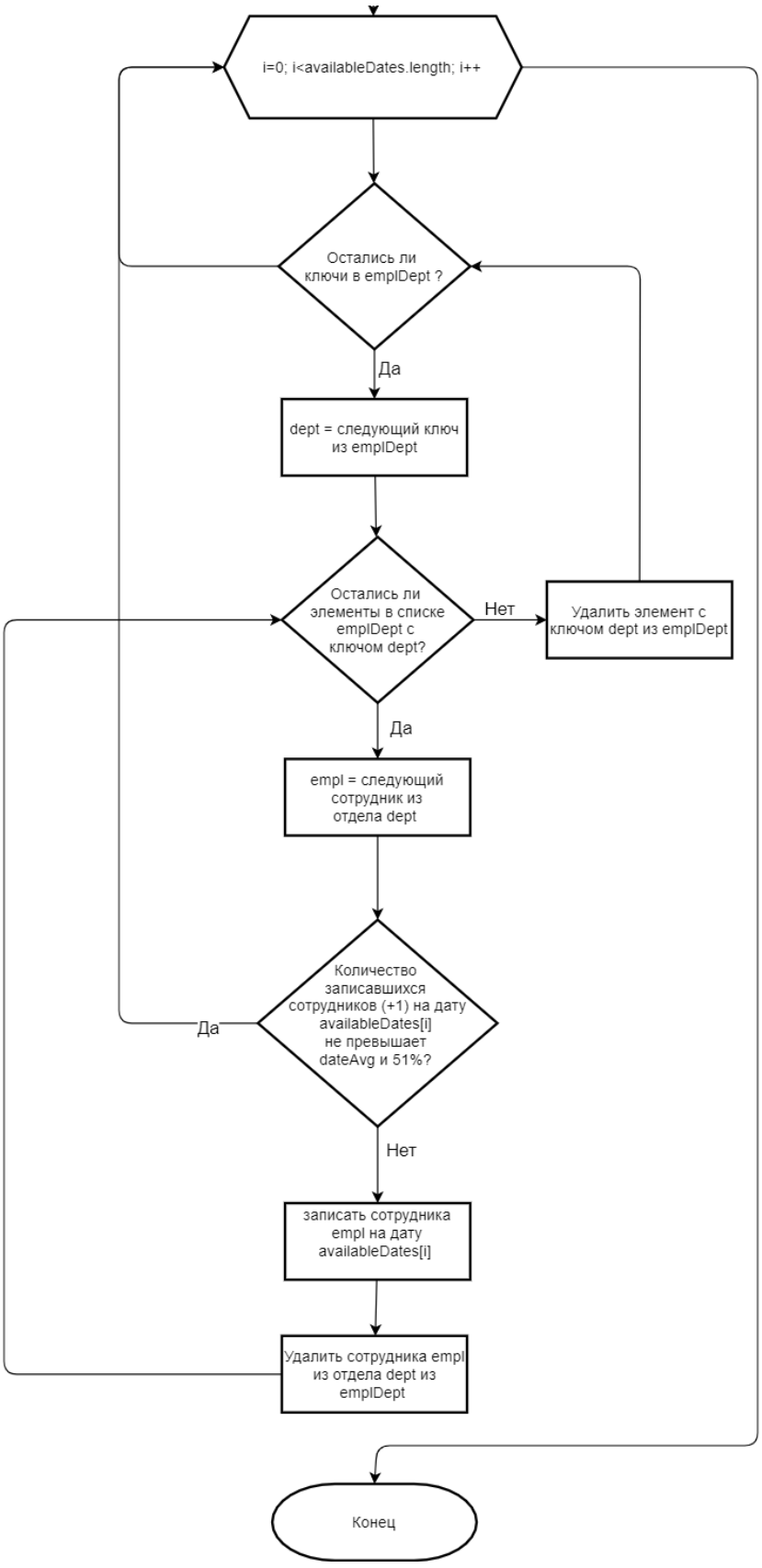


Рисунок 4 – Блок-схема алгоритма равномерного распределения не записавшихся сотрудников на медкомиссию (часть 2)

Выводы к главе 1

В рамках данного этапа была разработана модель управления процессом записи сотрудников на медкомиссию, также изучена предметная область, выделены основные требования к системе.

Для моделирования разрабатываемой системы была выбрана методология UML, поскольку она является всеобъемлющей и подходит для разработки программного обеспечения, что было одним из основных критериев в данной работе, так как разработка приложения производится с нуля, а не на базе существующей системы.

Используя разработанную модель системы, были получены данные для постановки задачи. В результате были выделены функциональные требования для разных пользователей, которые должны быть реализованы.

Основываясь на полученной информации, были смоделированы прецеденты взаимодействия пользователя с системой, которые заложили основу для дальнейшего проектирования и реализации.

# Глава 2 Логическое проектирование модуля записи сотрудников на медкомиссию

## 2.1 Выбор технологии логического моделирования

Создание логической модели базы данных – дин из ключевых этапов в процессе проектирования информационной системы. Правильный выбор технологии и языка моделирования для создания логической модели может значительно упростить процесс и улучшить качество результата.

Существует несколько языков для моделирования логических моделей баз данных, каждый из которых имеет свои преимущества и недостатки.

Один из самых популярных языков для моделирования логических моделей баз данных – это язык UML (Unified Modeling Language). UML предназначен для моделирования различных видов систем, включая программное обеспечение и базы данных. Он включает в себя различные типы диаграмм, такие как диаграммы классов, диаграммы прецедентов, диаграммы последовательности и др. Каждая диаграмма в UML представляет определенный аспект системы и позволяет более детально изучить ее структуру и функциональность.

Другим языком для моделирования логических моделей баз данных является ER-моделирование (Entity-Relationship Modeling). Он используется для создания графических представлений баз данных и описания отношений между сущностями в этих базах. ER-моделирование включает в себя диаграммы сущностей-связей (Entity-Relationship Diagrams, ERD), которые показывают, как сущности в базе данных связаны друг с другом.

Также существует язык IDEF1X (Integration Definition for Information Modeling), который используется для моделирования данных и был разработан для использования в среде проектирования баз данных. IDEF1X позволяет описывать сущности, атрибуты и связи между ними в базах данных.

Однако из перечисленных языков для моделирования логических моделей баз данных UML является более универсальным и мощным языком. Также UML имеет широкую поддержку в индустрии и доступен для множества инструментов и платформ. Поэтому выбор UML для моделирования логических моделей баз данных является более предпочтительным вариантом.

## 2.2 Описание логической модели системы

Одним из важнейших шагов в проектировании системы является составление диаграмм классов реализуемой системы. Этот шаг поможет понять какие сущности, методы и классы необходимы для функционирования приложения по заданным требованиям [3].

Для реализации главной задачи разрабатываемой модели необходимы три основных сервиса для управления сущностями сотрудников, расписаний и записей.

Сервис сотрудников позволяет добавлять, модифицировать и считывать информацию, касающуюся сотрудников.

Сервис расписаний медкомиссий, с помощью данного сервиса в системе можно добавить медицинскую организацию и даты её возможного посещения, после инициации новой медкомиссии сотрудники могут выбрать любой доступный день и создать запись для прохождения медкомиссии. Этот сервис доступен только администраторам.

С помощью сервиса записи сотрудники могут создавать и управлять записями на медкомиссии.

На рисунках 5 – 7 представлены диаграммы основных сервисов реализуемой модели управления процессом записи сотрудников на медкомиссию.

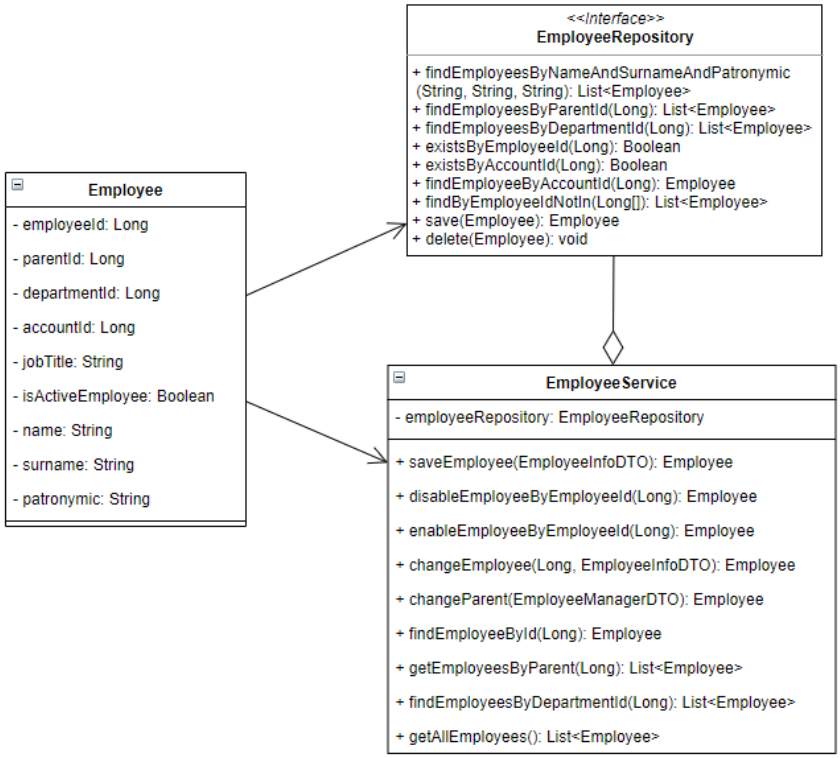


Рисунок 5 – Диаграмма классов сервиса сотрудников

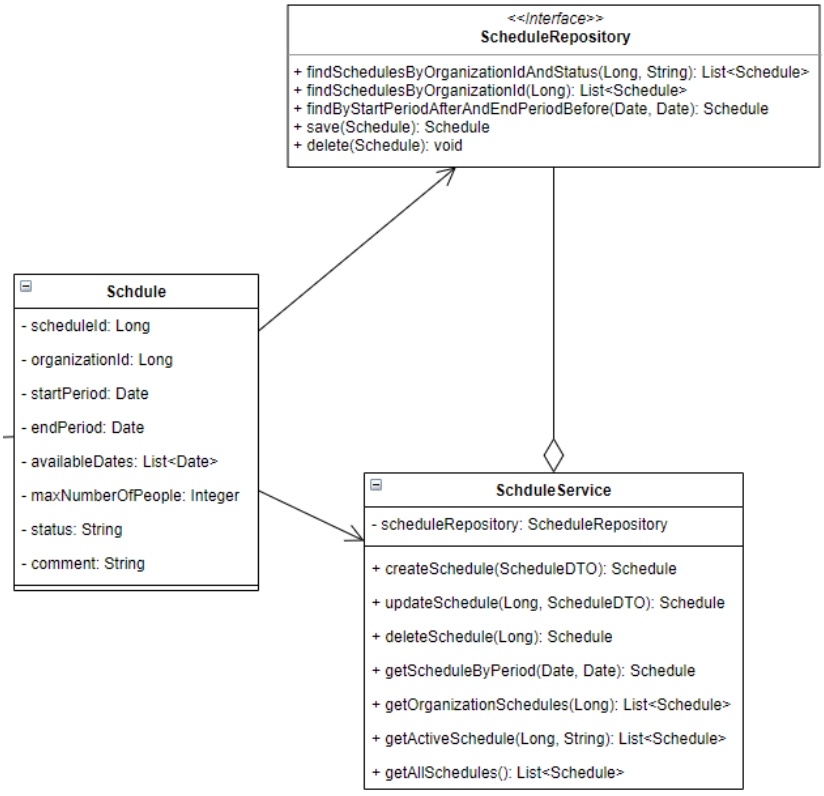


Рисунок 6 – Диаграмма классов сервиса расписаний медкомиссий

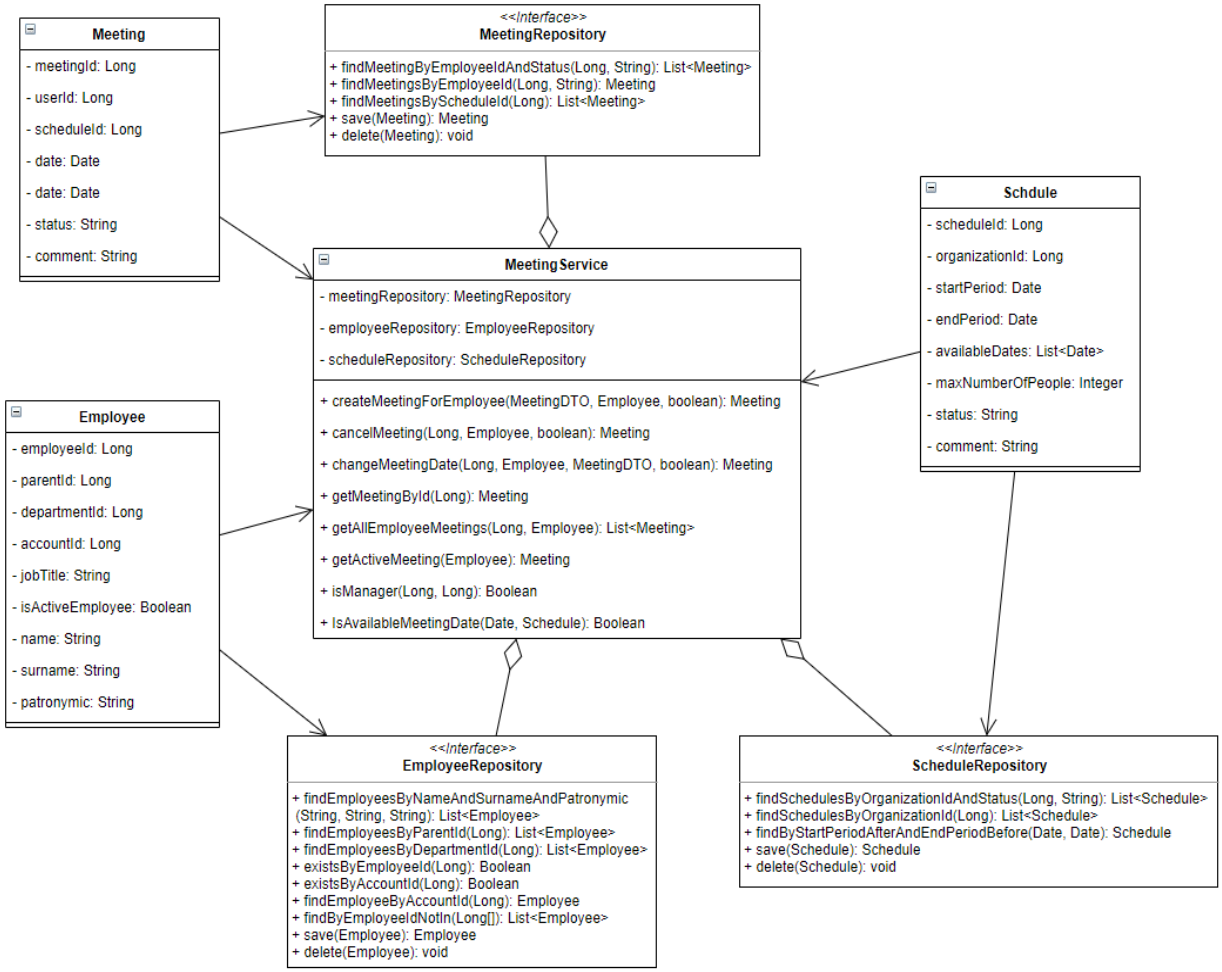


Рисунок 7 – Диаграмма классов сервиса записи на медкомиссию

Классы Employee, Meeting и Schedule являются сущностями. Классы с типом Repository отвечают за доступ к данным и их хранение. Классы-сервисы являются представителями сервисного слоя приложения и отвечают за бизнес-логику всей системы, они содержат методы для выполнения операций с данными, в них происходит обработка полученной информации от контроллеров. Использование сервисного слоя позволяет повысить модульность приложения, улучшить его тестируемость и сделать код более понятным и легким для поддержки. Кроме того, это позволяет разделить обязанности между различными слоями приложения и повысить его безопасность и производительность.

## 2.3 Создание логической модели данных

Следующим шагом после описания взаимодействия компонентов системы и существующих в ней сущностей была создана логическая модель данных, которая затем легла в основу физической.

Непосредственно перед моделированием было необходимо выбрать тип базы данных для понимания того, какие объекты в ней могут храниться и какие связи между ними могут существовать.

В современных технологиях баз данных можно выделить два основных направления: SQL и NoSQL. Они отличаются друг от друга в способе проектирования, поддерживаемых типах данных и методах хранения информации.

Реляционные БД, также известные как SQL-базы данных, являются одним из основных направлений в области баз данных. Они используют структуру таблиц, где каждая сущность связана с определённой темой, которая разбивается на столбцы и строки. Стоит отметить, что формат таблиц, в которых хранятся данные задаётся на этапе проектирования. SQL (Structured Query Language ) – стандартный язык  поддерживаемый реляционными БД, он используется для создания, изменения и чтения информации хранящейся в БД.

Реляционные базы данных имеют ряд преимуществ. Они стандартизируют данные, что делает обработку данных в системе простой и эффективной. Они позволяют удобно хранить и обрабатывать большие объемы данных. SQL-базы данных также могут использоваться для создания сложных отчетов, поддерживают транзакции, а также обеспечивают целостность данных.

Нереляционные базы данных, также известные как NoSQL-базы данных, это другой тип баз данных, который отличается от реляционных БД. В нереляционных базах данных нет жесткой структуры, которой нужно следовать, как в реляционных базах данных, и данные могут храниться в неструктурированном формате [13].

NoSQL-базы данных могут быть более гибкими, чем реляционные базы данных, и часто используются для хранения больших объемов неструктурированных данных, таких как изображения, видео, аудио и тексты. NoSQL БД также могут обеспечивать более быстрый доступ к данным, чем реляционные базы данных, особенно при работе с большими объемами данных [20].

Существует несколько различных типов NoSQL-баз данных, включая ключ-значение, документоориентированные, столбцово-ориентированные и графовые базы данных. Каждый тип базы данных имеет свои уникальные особенности и может быть лучше всего использован в зависимости от требований вашего проекта.

Рассматривая разрабатываемую модель управления процессом записи сотрудников на медкомиссию, можно выделить 5 основных сущностей, участвующих в процессе записи на медкомиссию:

- сотрудники;
- департаменты;
- записи на медкомиссию;
- расписания медкомиссий;
- медицинские организации.

И две технические:

- аккаунты;
- роли.

Согласно описанным выше сущностям разрабатываемая модель имеет высокую степень структурированности. Следовательно, наиболее подходящим решением в этом случае являются базы данных, использующие реляционную модель данных.

На рисунке 8 показана логическая модель данных, разработанная на основе выбранного типа базы данных и выделенных сущностей. Все сущности

связаны между собой, при это все связи, кроме одной технической, представляют тип «один ко многим» [7].

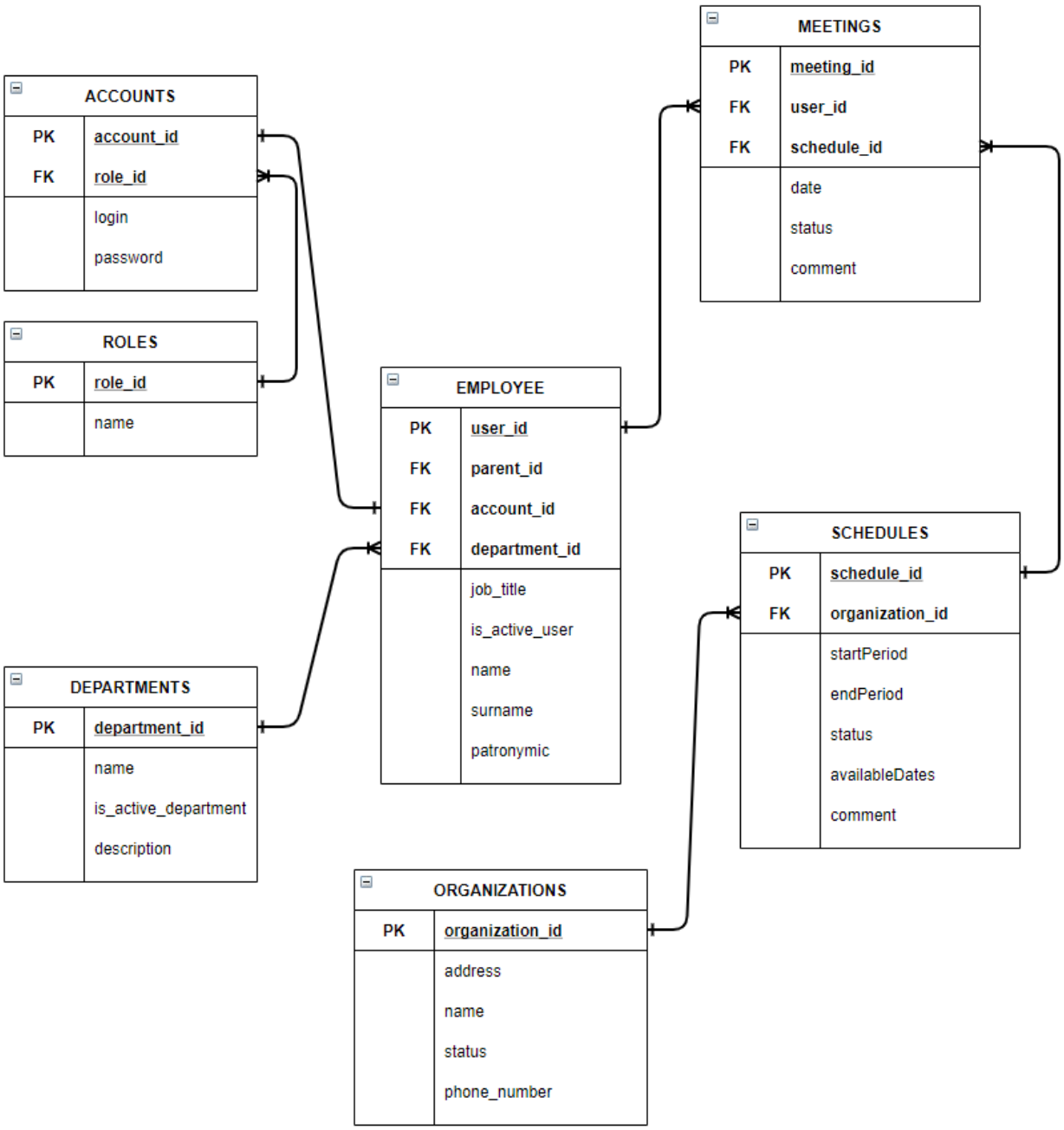


Рисунок 8 – Логическая модель данных

Выводы к главе 2

В данной главе была рассмотрена логическая модель системы записи сотрудников на медкомиссию.

Первым делом была выбрана технология моделирования. Учитывалась универсальность и распространённость языка моделирования. Этим критериям соответствовал UML.

После выбора технологии моделирования были составлены диаграммы классов основных сервисов для управления процессом записи сотрудников на медкомиссию.

На основе полученной информации была разработана логическая модель данных. Затем были выделены основные и технические сущности в разрабатываемой модели, а также связи между ними. В итоге получилось 7 связанных таблиц. Полученная логическая модель данных будет использована для создания физической модели, которая будет соответствовать конкретной СУБД.

# Глава 3 Программная реализация модели управления процессом записи сотрудников на медкомиссию

## 3.1 Выбор архитектуры системы

Существует множество различных архитектур для разработки программного обеспечения, каждая из которых имеет свои преимущества и недостатки. Рассмотрим наиболее популярные из них:

- монолитна архитектура (Monolithic Architecture);
- сервис-ориентированная архитектура (SOA – Service Oriented Architecture);
- микросервисная архитектура (Microservice Architecture).

Монолитная архитектура – это традиционный подход к разработке приложений, когда весь код находится в одном приложении. Все компоненты приложения, такие как контроллеры для обработки web-запросов клиента, сервисный слой с бизнес-логикой, слой доступа к данным, представление сущностей в виде классов, конфигурация защиты приложения находятся в одном месте. Такая архитектура обычно проста в реализации, обслуживании и масштабировании, но может стать сложной и неповоротливой с ростом приложения и его функционала [18].

SOA (Service-Oriented Architecture) – это подход к разработке программного обеспечения, который предполагает создание программных компонентов (сервисов), обеспечивающих выполнение определенных функций, и их последующую интеграцию в единую систему. Основная концепция SOA заключается в том, что каждый сервис должен быть максимально автономным и изолированным от остальных компонентов системы. Это позволяет быстро и гибко изменять систему, добавлять или удалять сервисы без нарушения работы других компонентов. Сервисы могут взаимодействовать между собой посредством определенных протоколов и интерфейсов, таких как SOAP или REST. Среди основных преимуществ SOA

можно выделить следующие: Высокая гибкость и масштабируемость, повышенная надёжность, распределенность и гетерогенность. Однако у такой архитектуры присутствует ряд недостатков: сложность разработки, отладки и интеграции из-за необходимости учитывать большое количество факторов, таких как протоколы, безопасность и другие требования корневой системы.

Микросервисная архитектура – это подход к разработке программного обеспечения, при котором приложение разбивается на небольшие автономные сервисы, которые могут работать независимо друг от друга и взаимодействовать между собой посредством API.

Основная концепция микросервисной архитектуры заключается в разделении приложения на более мелкие, легко управляемые компоненты, каждый из которых выполняет конкретную функцию. Каждый сервис может быть разработан, тестирован и развернут независимо от других сервисов.

Преимущества микросервисной архитектуры включают:

- гибкость: каждый сервис может быть разработан протестирован и развёрнут по отдельности, что позволяет более гибко управлять и изменять приложение, а также помогает увеличить продуктивность команды разработчиков за счёт уменьшения количества внешних факторов, влияющих на реализацию конкретной задачи;
- масштабируемость: каждый сервис может быть масштабирован по отдельности;
- отказоустойчивость: в случае выхода из строя какого-ибо из сервисов, вся системы не перестанет работать, и пользователь сможет продолжить работать с рабочей частью приложения;
- распределение ресурсов: разделение системы на сервисы позволяет балансировать нагрузку, добавляя или уменьшая ресурсы к каждому микросервису по отдельности.

Недостатки микросервисной архитектуры включают:

- усложнение: микросервисная архитектура требует более высокого уровня абстракции и знания более широкого спектра технологий, что усложняет разработку и поддержку приложения;
- увеличение количества точек отказа: каждый сервис имеет свои точки отказа, что может усложнить управление и отладку;
- необходимость управления связями: сервисы должны взаимодействовать между собой посредством API, что может усложнить управление связями между сервисами, а также общий подход к защите приложения.

Сервисная архитектура – это более общий подход SOA, при котором приложение разбивается на отдельные сервисы, каждый из которых представляет собой отдельно работающий модуль. Подобная архитектура может включать в себя не только SOA, но и другие подходы, такие как микросервисная архитектура и монолитная. Сервисы могут быть организованы как на одном сервере, так и на нескольких, и могут быть написаны на разных языках программирования. Этот подход позволяет более гибко управлять масштабированием и обновлением приложения, однако требует дополнительных усилий по организации взаимодействия между сервисами.

Согласно информации, полученной в предыдущих главах, разрабатываемая система является цельной, подразумевающей под собой выполнение возможность записи сотрудников на медкомиссию. Также стоит помнить, что у приложения должен быть web-интерфейс. Соответственно наиболее верным решением будет разделить систему на базу данных и два сервиса: серверная часть и клиентская. Так как предполагается, что эти два сервиса будут написаны на разных языках программирования.

Серверная часть (Backend) приложения обрабатывает полученные запросы и в соответствии с бизнес-логикой извлекает, отображает или изменяет данные [9].

Серверный модуль приложения разделён на несколько слоёв [9]:

– слой обработки http-запросов;
– слой бизнес-логики приложения;
– слой доступа к данным;
– слой представления сущностей приложения.

Клиентский сервис (Frontend) представляет из себя отдельный модуль, который отвечает за взаимодействие с пользователем и является посредником между клиентом и серверной частью приложения.

В процессе анализа разрабатывались диаграммы последовательности, для документации сервисов и взаимодействия между ними. Наиболее показательными являются диаграммы, описывающие процесс авторизации в системе (рисунок 9), переход в личный кабинет сотрудника (рисунок 10) и процесс записи на медкомиссию (рисунок 11). На разработанных диаграммах приведены модели взаимодействия пользователя с web-приложением.

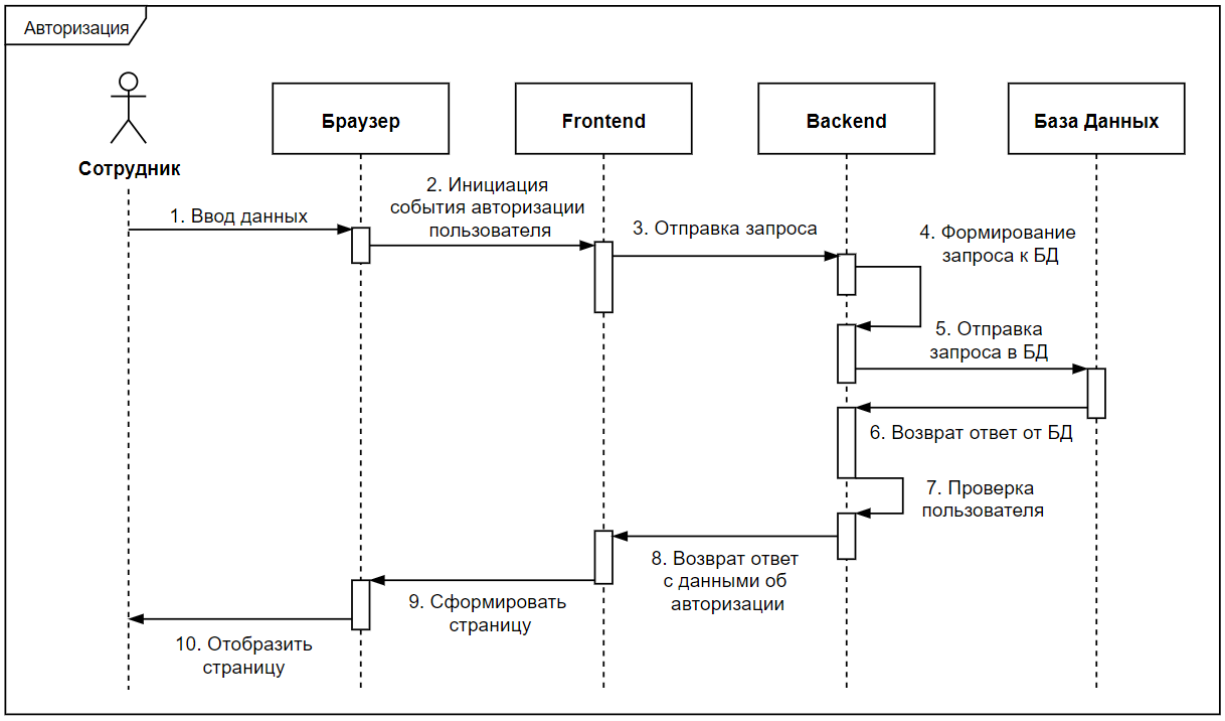


Рисунок 9 – Диаграмма последовательности для прецедента «Авторизация»

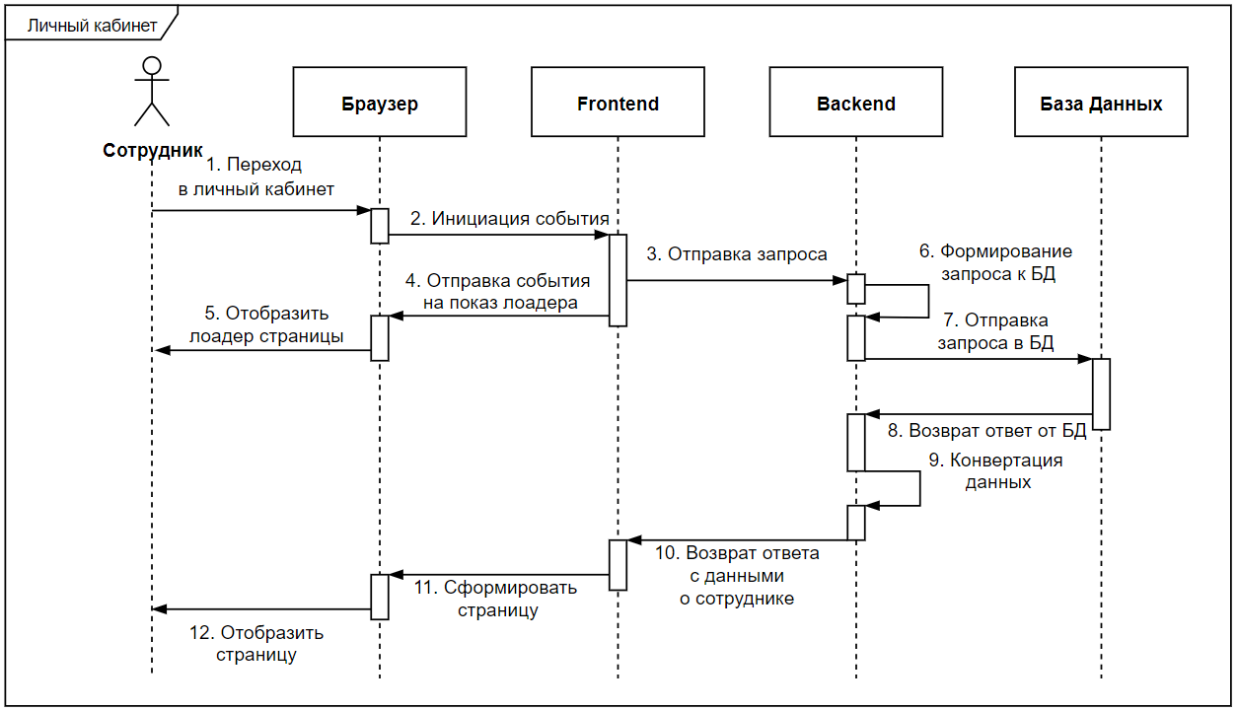


Рисунок 10 – Диаграмма последовательности для прецедента «Вход в личный кабинет сотрудника»

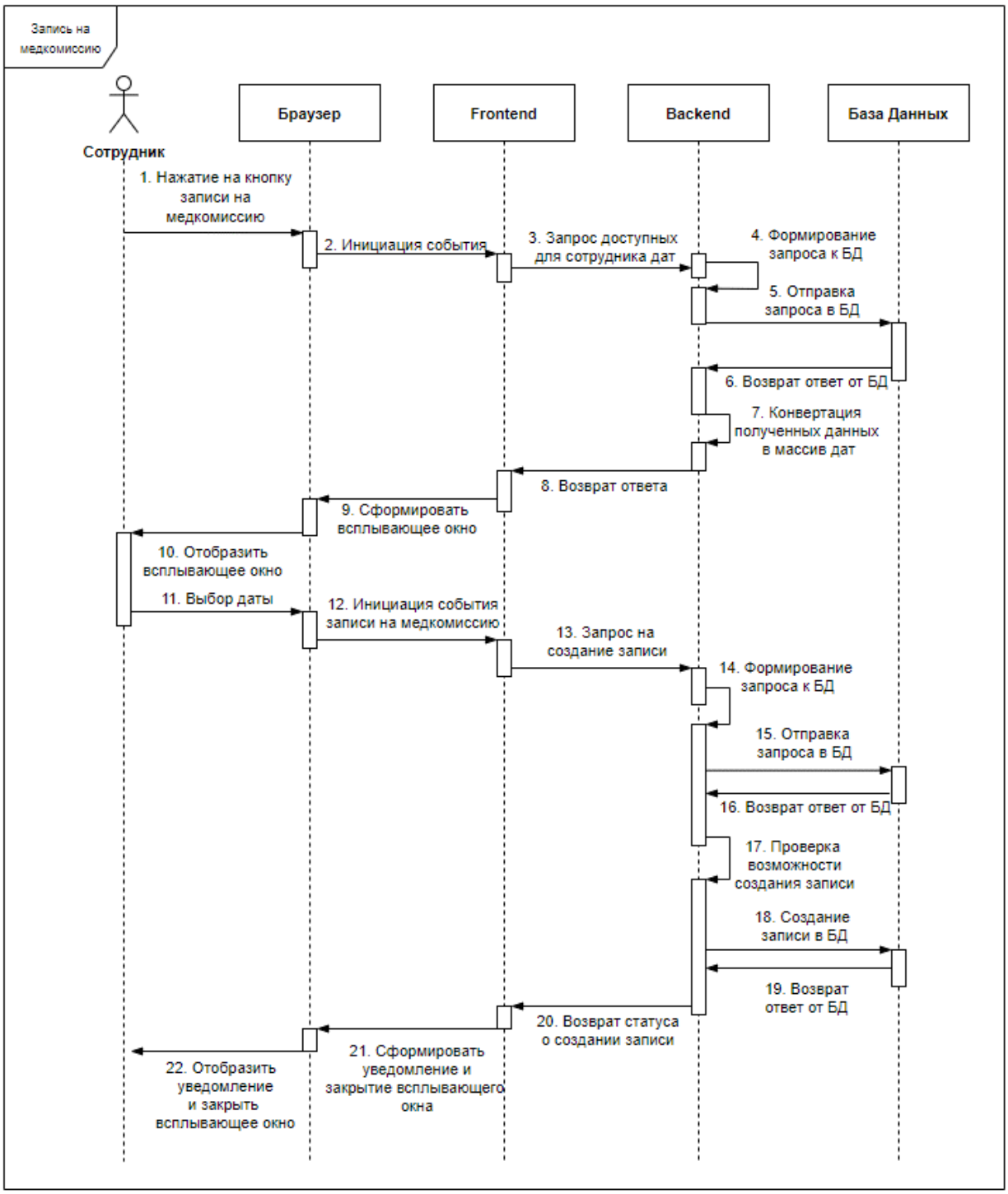


Рисунок 11 – Диаграмма последовательности для прецедента «Запись сотрудника на медкомиссию»

### 3.2 Выбор СУБД для базы данных

Первым шагом были выделены основные требования к СУБД. Ранее было определено, что в разрабатываемой системе будет использоваться реляционная база данных. Соответственно необходимо выделить самые

распространённые реляционные системы управления базами данных (РСУБД), а затем выбрать наиболее подходящую.

Существует множество СУБД с открытым исходным кодом, которые могут использоваться как в коммерческих, так и не коммерческих проектах. Некоторые из наиболее популярных СУБД с открытым исходным кодом включают: MySQL, PostgreSQL, SQLite, Firebird [17].

MySQL: одна из самых популярных СУБД в мире с открытым исходным кодом. Может использоваться в коммерческих и не коммерческих проектах. Владельцем MySQL является Oracle Corporation. Основными плюсами это СУБД являются поддержка большого количества языков программирования и большое сообщество пользователей и разработчиков, что означает наличие обширной документации и готовых решений. Однако есть и минусы: не всегда стабильное поведение в высоконагруженных системах, возможны проблемы с блокировкой и сетевыми задержками; некоторые функции и возможности могут быть более сложными в использовании, чем у конкурентов, что может повлечь за собой сложность поддержки и разработки [19].

SQLite: легковесная СУБД, которая хранит данные в одном файле, что делает его идеальным выбором для приложений с небольшим объемом данных. Из плюсов данной СУБД можно выделить легковесность и малый размер. SQLite занимает очень мало места и не требует большого объема оперативной памяти, поэтому она может быть использована на устройствах с ограниченными ресурсами. Минусы – ограниченность по производительности, а также отсутствие многопользовательского режима, что ограничивает ее использование в крупных проектах с большим количеством пользователей и одновременных запросов к базе данных.

Firebird – открытая СУБД, которая имеет широкие возможности, включая многопоточность, транзакции и полнотекстовый поиск. Эта СУБД является мультиплатформенной, масштабируема, а также проста в установке. Из недостатков стоит упомянуть низкую производительность при работе с

большими данными, а также ограниченность функционала по сравнению с некоторыми другими СУБД.

PostgreSQL – это одна из наиболее популярных реляционных СУБД с открытым исходным кодом, которая имеет ряд преимуществ и недостатков [21].

Плюсы:

- открытый исходный код: PostgreSQL является открытым программным обеспечением, что означает, что пользователи могут свободно использовать, изменять и распространять его без ограничений;
- полная совместимость с языком SQL, что позволяет разработчикам использовать стандартные запросы к базе данных и упрощает интеграцию с другими приложениями;
- надежность и безопасность: PostgreSQL имеет высокую степень надежности и безопасности благодаря своей многоуровневой системе безопасности, а также возможности резервного копирования и восстановления данных;
- расширяемость: PostgreSQL имеет мощную систему расширений, которая позволяет пользователям создавать свои собственные функции, типы данных и модули.

Из минусов можно выделить – ложность настройки: PostgreSQL может быть сложным для установки и настройки, особенно для пользователей с ограниченным опытом работы с базами данных.

Учитывая требования к безопасности данных и возможность расширения функционала в будущем, PostgreSQL является наиболее оптимальным выбором в качестве СУБД для модели управления процессом записи сотрудников на медкомиссию. Богатый набор функций и поддержка многопоточности позволяют эффективно обрабатывать большие объемы данных и удовлетворять потребности различных групп пользователей. Кроме

того, PostgreSQL отличается открытым и расширяемым кодом, что обеспечивает гибкость и возможность внедрения дополнительных функций и инструментов в будущем.

### 3.3 Создание физической модели данных

На основе логической модели была разработана физическая модель данных. Для реализации использовалось приложение Draw.io. На рисунке 12 представлена итоговая модель данных.

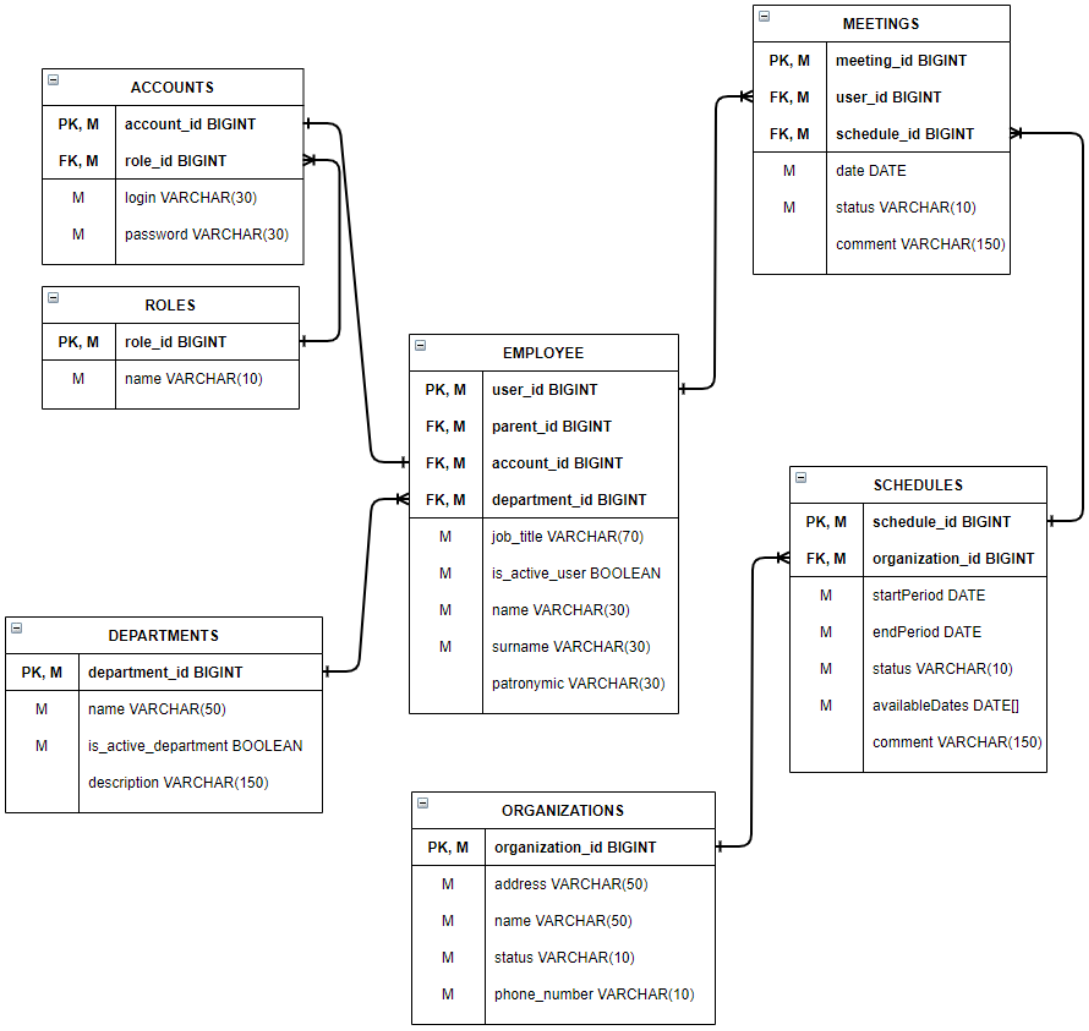


Рисунок 12 – Физическая модель данных

Первым шагом сущности логической модели данных были преобразованы в физические. Модель постоянно перерабатывалась до тех пор, пока не была достигнута третья нормальная форма. Для каждого атрибуты был выбран соответствующих тип данных поддерживаемый СУБД. Все идентификаторы имеют числовой тип BIGINT и генерируются автоматически. Атрибуты, помеченные маркером «M» (Mandatory), являются обязательными [5].

Также в физической модели данных присутствуют системные атрибуты с логическим типом данных, которые являются маркером актуальности конкретной строки.

В заключение можно отметить, что физическая модель данных представляет собой детальное описание структуры базы данных, которое учитывает все требования, предъявляемые к хранению и обработке данных. Разработка физической модели данных на основе логической модели является важным этапом проектирования базы данных, так как она позволяет определить типы данных, индексы, ограничения целостности, связи таблиц и возможности СУБД.

## 3.4 Реализация серверной части системы

### 3.4.1 Выбор инструментов для реализации

Серверная часть приложения является ядром системы. Она отвечает за работу всей основной логики, обрабатывает запросы пользователей, подключается к базам данных и

Серверная часть приложения отвечает за подключение к базам данных и обработку полученной информации, а также за выполнение бизнес-логики приложения. Взаимодействуя с клиентской частью приложения, серверная часть может обрабатывать запросы, получать и отправлять данные, а также контролировать доступ и безопасность приложения. В целом, серверная часть

приложения является фундаментом, на котором строится работа приложения и обеспечивает его стабильность и производительность.

Так модель управления процессов записи сотрудников на медкомиссию разрабатывается с нуля, то выбор технологий для реализации достаточно свободный. В то же время необходимо выделить базовые и общепринятые требования для более эффективного выбора:

- обработка запросов и отправка ответов в формате JSON;
- реализация REST API, включающего все необходимые методы для обмена данными между клиентом и сервером, такие как GET, POST, PUT и DELETE;
- аутентификация и авторизация пользователей при доступе к защищенным ресурсам;
- обработка исключений и ошибок, возникающих при обработке запросов, и отправка соответствующих кодов ошибок клиенту;
- обработка запросов в асинхронном режиме, чтобы избежать блокировки сервера;
- надежное хранение данных в базе данных, обеспечение безопасности данных и защиту от SQL-инъекций и других атак;
- реализация механизма логирования, чтобы отслеживать работу сервера и быстро реагировать на ошибки и проблемы.

В то же время выбранный инструмент не должен быть перегружен лишним функционалом, но достаточно популярен на рынке разработки ПО и задокументирован [10].

В качестве технологий, удовлетворяющей вышеперечисленные требования, лучше всего подходит язык программирования Java и фреймворк Spring Boot, а также Maven в качестве инструмента для автоматизации сборки проекта.

Фреймворк – это набор готовых инструментов и библиотек, который упрощает процесс создания приложений. Он предоставляет программистам

шаблоны и структуры для решения конкретных задач, таких как обработка входных данных, маршрутизация запросов, работа с базами данных и т.д. Фреймворк часто используется для ускорения разработки, улучшения качества и безопасности кода, а также снижения затрат на поддержку приложений в будущем [16].

Java – это объектно-ориентированный язык программирования, который используется для создания широкого спектра приложений, включая веб-приложения, мобильные приложения, настольные приложения и другие. Он хорошо зарекомендовал себя в разработке серверных приложений благодаря своей масштабируемости и надежности [12].

Spring Boot – то фреймворк для разработки серверных приложений на Java, который облегчает создание микросервисов и RESTful API. Он предоставляет удобный способ настройки и конфигурирования приложения, обеспечивает высокую производительность и надежность, а также хорошо масштабируется. Spring Boot также имеет большое сообщество разработчиков и богатую документацию, что упрощает разработку и поддержку приложения [22].

Выбор языка Java и фреймворка Spring Boot для разработки серверной части приложения обусловлен их популярностью, мощными возможностями и развитой экосистемой, которая облегчает разработку, тестирование и развертывание приложений. Кроме того, Java и Spring Boot обеспечивают высокую производительность и безопасность, что является важным фактором для программных продуктов.

Maven – это инструмент для автоматизации сборки проектов на языке Java. Он упрощает процесс управления зависимостями проекта и его сборки. Конфигурация для сборки проектов с помощью Maven описывается в файле «pom.xml» [14].

Основные преимущества Maven заключаются в следующем:

- удобное управление зависимостями. Maven автоматически скачивает необходимые зависимости из центрального репозитория и устанавливает их в проект;
- простота в использовании. Maven основывается на принципе "Конвенция перед конфигурацией" (Convention over Configuration), что упрощает настройку и сборку проекта;
- поддержка плагинов. Maven имеет множество плагинов, которые позволяют расширять его функциональность;
- совместимость с CI/CD. Maven является частью инфраструктуры CI/CD и может быть интегрирован практически любыми средствами автоматизации сборки и развертывания;
- выбор Maven оправдан, если в проекте используется множество зависимостей, необходимо автоматизировать процесс сборки проекта, или если нужна интеграция с CI/CD платформой.

Использование Maven вместе с Spring Boot позволяет автоматически управлять зависимостями и сборкой приложения, что значительно упрощает процесс разработки и поддержки приложения.

### 3.4.2 Программная реализация компонентов сервиса

В первую очередь был инициализирован проект Spring Boot и добавлены все необходимые зависимости в файл «pom.xml».

Далее были определены модели сущностей в объектном виде. Созданные классы, представляющие сущности были помечены соответствующим образом аннотацией «Entity». Соответственно для полей классов, представляющие колонки таблиц, было задано наименование, тип и формат данных, а также ограничения.

После создания сущностей необходимо было инициализировать классы-репозитории с помощью Spring Data JPA, которые являются связующим звеном между программным кодом и базой данных. В соответствии с

диаграммой классов в созданных репозиториях были выделены основные методы, необходимые для общения с базой данных [23].

Следующим шагом были реализованы REST-контроллеры, которые отвечают за обработку HTTP-запросов и возвращение HTTP-ответов с соответствующими статусами и данными в формате, заданном в заголовках запросов [1]. Дополнительно в слое контроллеров был реализован обработчик исключений, который «ловит» исключения, выбрасываемые в ходе работы сервисного слоя или возникающие при обработке некорректных запросов от клиента, а после возвращает ответ клиенту с информативной ошибкой и подходящим HTTP-статусом [4].

Далее был реализован сервисный слой приложения (Service layer), который отвечает за работу бизнес-логики приложения. Он является посредником между слоем контроллеров и репозиториев. Классы этого слоя помечены аннотацией «Service».

Затем были написаны DTO (Data Transfer Object) классы, которые используются для передачи информации между разными слоями приложения. DTO-объекты позволяют скрыть детали реализации слоя доступа к данным от остальных компонентов приложения. Таким образом, изменения в базе данных не будут влиять на работу REST-контроллеров или слоя сервисов, так как DTO-объекты предоставляют общий интерфейс доступа к данным.

В завершении была реализована логика для защиты приложения с помощью Spring Boot Security и JWT (JSON Web Token).

Spring Boot Security – это модуль фреймворка Spring Boot, предназначенный для обеспечения безопасности веб-приложений. Он предоставляет различные инструменты для авторизации и аутентификации пользователей, а также контроля доступа к различным ресурсам приложения.

JWT (JSON Web Token) – это формат токенов, используемых для передачи информации между клиентом и сервером в формате JSON. JWT состоит из трех частей: заголовка, полезной нагрузки и подписи. Он

использован вместе с Spring Boot Security для аутентификации и авторизации пользователей в приложении [11].

Дополнительная настройка фреймворка Spring Boot происходит с помощью файла «application». Например, нём указаны данные о подключении к БД, такие как логин, пароль, url и название класса-драйвера, а также подпись для JWT и порт, на котором развёртывается приложение.

## 3.5 Реализация клиентской части web-приложения

### 3.5.1 Выбор инструментов для реализации

Существует два основных подхода для реализации клиентской части веб-приложения: клиентский рендеринг (CSR) и серверный рендеринг (SSR).

Клиентский рендеринг – это подход, при котором HTML-код страницы генерируется на стороне клиента, с помощью JavaScript-кода, который выполняется в браузере пользователя. Сначала браузер получает только базовый HTML-шаблон, а затем он загружает JavaScript-код, который дополняет этот шаблон и делает страницу интерактивной. Этот подход обычно используется для создания сложных интерактивных приложений, таких как социальные сети или онлайн-магазины.

Серверный рендеринг – это подход, при котором HTML-код страницы генерируется на сервере и отправляется в браузер пользователя. Это означает, что браузер получает уже полностью отрендеренную страницу, а не базовый HTML-шаблон. Этот подход используется для создания страниц, которые должны быть отображены быстро, без задержек, а также для лучшей оптимизации под поисковые системы.

В современном мире для реализации клиентской части web-приложений используется JavaScript и основанные на нём инструменты. Разработку web-интерфейс приложения можно осуществлять на чистом JavaScript, но в большинстве случаев для этого используются специализированные

библиотеки и фреймворки. Это обусловлено тем, что различные браузеры поддерживают разные версии языка.

На данный момент наиболее популярными фреймворками для разработки клиентской части web-приложений являются React, Vue и Angular, так называемая большая тройка.

Angular.js – это фреймворк для разработки веб-приложений, который создала Google. Он предлагает множество инструментов для разработки сложных приложений, включая систему управления состоянием, маршрутизацию, создание пользовательских интерфейсов и многие другие. Angular.js также имеет свой язык шаблонов, который помогает разработчикам легко создавать интерфейсы приложений и обеспечивает высокую производительность.

React.js – это библиотека JavaScript для создания пользовательских интерфейсов, которую разработала компания Facebook. Она предоставляет инструменты для создания компонентов, которые можно повторно использовать в различных частях приложения. React.js также предлагает удобные инструменты для управления состоянием, маршрутизации и других задач, которые необходимы для разработки веб-приложений.

Vue.js – это современный фреймворк для создания пользовательских интерфейсов, который предлагает простой и интуитивно понятный синтаксис, быструю скорость работы, а также удобные инструменты для создания компонентов и управления состоянием приложения. Vue.js также имеет широкую поддержку со стороны сообщества разработчиков и предоставляет множество сторонних библиотек и плагинов, что делает его гибким и универсальным инструментом для создания веб-приложений [24].

Таким образом, основной выбор стоял между Vue и React исходя из того, что оба этих фреймворка являются легковесными. Окончательным выбором стал Vue.js, так как порог входа для разработки приложения на данном фреймворке казался ниже, чем на React. Также вместе с Vue будет использоваться вспомогательный фреймворк Nuxt.js.

Nuxt.js – это фреймворк базируемый на Vue.js и служит для создания универсальных веб-приложений, которые могут быть отрендерены как на стороне клиента, так и на стороне сервера. Его использование позволяет значительно ускорить загрузку страниц и улучшить производительность приложения. Этот фреймворк также предоставляет множество настроек по умолчанию, которые облегчают создание приложений и уменьшают необходимость настройки многих вещей с нуля. Кроме того, Nuxt.js имеет открытое и активное сообщество разработчиков, которые создают новые инструменты и плагины для упрощения процесса разработки. Все это делает Nuxt.js идеальным выбором для создания масштабируемых и производительных веб-приложений.

### 3.5.2 Определение структуры проекта

Ключевым элементом в приложении являются однофайловые компоненты, которые объединяют в себе шаблон, логику и стили, что упрощает создание и поддержку приложения.

За глобальные состояние и методы отвечает официальная библиотека Vue – Vuex. Использование Vuex позволяет легко и эффективно управлять состоянием приложения и обмениваться данными между компонентами.

Для совершения HTTP-запросов к серверной части приложения использовался библиотека Axios.

Конфигурация роутеров выполнена с помощь библиотеки VueRouter и файла конфигурации Nuxt.js, что позволяет ограничить доступ к страницам в зависимости от роли пользователя, а также вызывать компоненты Vue при обращении по указанному пути.

### 3.5.3 Определение структуры проекта

Ключевым элементом в приложении являются однофайловые компоненты, которые объединяют в себе шаблон, логику и стили, что упрощает создание и поддержку приложения. В разрабатываемом приложении шесть основных компонент: авторизация, личный кабинет, страница

добавления и изменения записи на медкомиссию, история медкомиссии, список подчинённых, страница создания расписания для новой медкомиссии.

Для реализации глобального состояние и его методов использовалась библиотека Vuex – это официальная библиотека управления состоянием приложения для Vue.js. Она используется для управления данными, которые могут быть использованы в разных компонентах приложения. Центральным понятием в Vuex является Store – это объект, который содержит в себе всё состояние приложения. Store предоставляет API для чтения и изменения состояния, а также позволяет определить действия, которые могут быть вызваны для изменения состояния в Store. Использование Vuex позволило легко и эффективно управлять состоянием приложения и обмениваться данными между компонентами.

Маршрутизация выполнена с помощь библиотеки VueRouter и конфигурации Nuxt.js, что позволяет ограничить доступ к страницам в зависимости от роли пользователя, а также сопоставить компоненты и URL адреса, которым они соответствуют.

Для реализации HTTP-запросов была использована библиотека Axios. Эта библиотека представляет собой надстройку над XMLHttpRequest, который является стандартным инструментом для работы с запросами в JavaScript. Также с помощью Axios в приложении обрабатываются ошибки доступа и отсутствия каких-либо запрашиваемых данных. В случае получения ошибки 401, что означает, что пользователь не авторизован в системе, приложение автоматически перенаправит пользователя на страницу входа. Axios выполняет все запросы асинхронно, что означает, что основной поток приложения не блокируется во время их выполнения, и позволяет системе выполнять другие задачи. По завершении запроса результат передается обратно в основной поток для дальнейшей обработки.

Для аутентификации пользователя используется форма входа, при вводе логина и пароля они отправляются на сервер. В ответ сервер возвращает токен

авторизации, который сохраняется в браузере и используется при каждом обращении к серверу в заголовке запроса.

Для создания пользовательского интерфейса веб-приложения использовался фреймворк Vuetify на базе Vue.js. Он предоставляет готовые компоненты, стили и элементы управления, которые можно использовать для быстрого и удобного создания интерфейса без необходимости написания большого количества кода с нуля. В итоге, использование Vuetify позволило значительно ускорить разработку, а также обеспечить стильный и современный дизайн пользовательского интерфейса [25].

### 3.6 Описание функциональности системы

Итоговая функциональность реализованной модели полностью соответствует определенным в начале работы требованиям. Сценарии работы разделяются на три основных потока: работа от имени сотрудника, руководители и администратора.

В личном кабинете сотрудник может просматривать информацию об актуальной медкомиссии, а также изменять актуальную запись на медкомиссию или добавлять новую в соответствии с ограничениями, диктуемыми логикой приложения (рисунок 13).

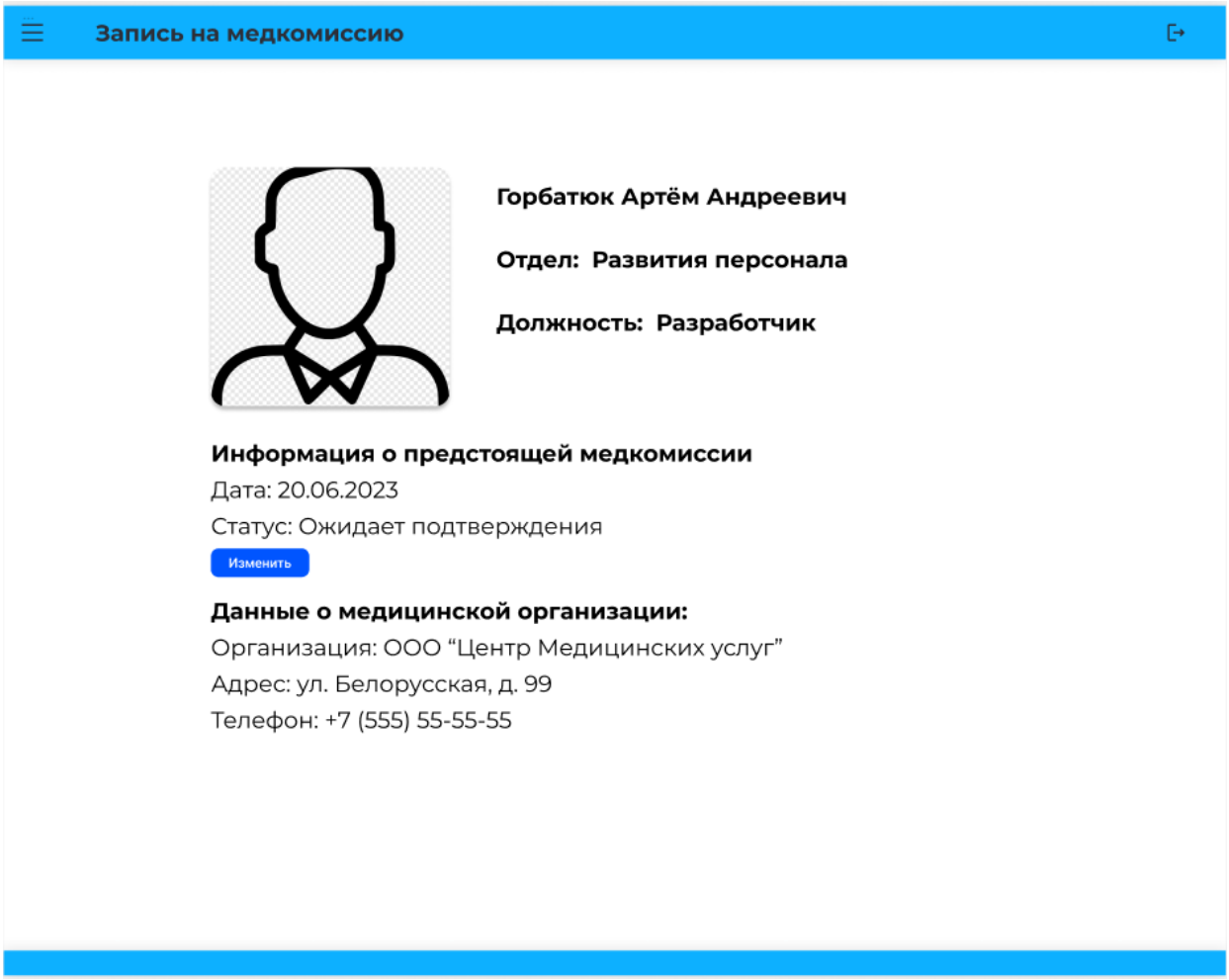


Рисунок 13 – Личный кабинет сотрудника

Что касается руководителя, то он может просматривать информацию об актуальных записях на медкомиссию своих подчинённых, а также инициировать и изменять новые записи для них и для себя.

Администратор имеет доступ ко всем вышеупомянутым функциям и в дополнение может создавать новые расписания медкомиссий, а также инициировать запуск алгоритма распределения не записавшихся сотрудников на свободные даты для прохождения медкомиссии.

Каждый пользователь приложения должен быть зарегистрирован для работы с ним.

### 3.7 Тестирование системы

Тестирование системы проводилось в несколько этапов. Первый этап заключался в покрытии автоматическими тестами, с использованием Junit и Mockito, всего сервисного слоя приложения. Подобные тесты помогут проверить бизнес-логику приложения во всех возможных сценариях. Тестирование также обеспечивает дополнительную защиту от ошибок, которые могут возникнуть в результате изменений в коде в будущем.

Во втором этапе тестирования был использован сервис Postman, для отправки запросов к API. Postman – это приложение для тестирования API, которое позволяет отправлять HTTP-запросы, указывая метод, тело запроса и путь, а после проверять ответы на корректность. Такой способ тестирования является неотъемлемым при разработке web-интерфейса. В процессе тестирования на каждую реализованную точку входа в приложение отправлялись корректные и не корректные запросы с различным набором данных. При этом отслеживалось поведение системы, обработка исключения и отправка соответствующих ответов. По результатам тестирования в программный код вносились корректировки до тех пор, пока все описанные сценарии работы модели выполнялись без каких-либо неожидаемых ошибок.

Заключительным этапом тестирования являлась проверка web-интерфейса приложения. Все функциональные элементы web-страницы должны были отрабатывать в соответствии со вторым этапом тестирования.

Выводы к главе 3

В данной главе были рассмотрены современные методы и технологии построения прикладных программных продуктов. Были изучены основные архитектурные принципы, используемые при разработке программных систем, проведен анализ этих принципов и выбрана подходящая архитектура для реализации модели управления процессом записи сотрудников на медкомиссию. Для реализации была выбрана сервисная архитектура.

Далее были рассмотрены современные системы управления базами данных, которые в настоящий момент активно используются в разработке программного обеспечения. Для реализации проекта было необходимо выбрать наиболее подходящую систему и разработать для нее физическую модель данных, которая соответствовала основным требованиям. Среди критериев выбора были: реляционная модель данных, некоммерческая лицензия, высокая стабильность и надежность, наличие регулярных обновлений. В результате, было решено выбрать PostgreSQL.

После завершения подготовительных этапов была осуществлена непосредственно программная разработка компонентов системы, для которых были выбраны оптимальные технологии. При выборе технологий было уделено особое внимание строгой типизации, высокой производительности и распространённости. Во время разработки каждого компонента были использованы общие модели данных для поддержания единообразия компонентов.

Последним шагом в реализации модели было проведено тестирование приложения с помощью Postman и Junit.

# Заключение

В данной работе была произведена реализация модели управления процессов записи сотрудников на медкомиссию.

В рамках работы над проектом был произведён анализ предметной области и существующих моделей записи на медкомиссию. На основе полученной информации, также пожеланий пользователей и представителей администрации университета были выявлены основные требования к реализации и функциональные разрабатываемого приложения. Произведено проектирование различных аспектов реализуемой модели. Для проектирования была выбрана методология UML, с помощью которой были спроектированы диаграммы: прецедентов, видов деятельности, классов и последовательности. В результате была поставлена задача на реализацию web-приложения.

В рамках процесса реализации был произведён анализ существующих архитектур и выбрана наиболее подходящая. Выбор пал на сервисную архитектуру из-за того, что такая архитектура позволяет создавать приложения, состоящие из независимых компонентов, которые могут быть развернуты и масштабированы отдельно друг от друга, в нашем случае это приложение клиента (web-интерфейс) и сервера (REST API). Это позволяет упростить разработку и поддержку приложения, а также обеспечить более высокую отказоустойчивость и масштабируемость.

На основе полученной информации был определён стек технологий для реализации программного обеспечения, который отвечал основным требованиям, архитектуре, производительности, удобству разработки и поддержки. Затем с их помощью была произведена реализация сервисов web-приложения. В процессе и по завершении реализации было произведено тестирования системы.

Итогом работы является реализованная модель управления процессом записи сотрудников на медкомиссию в виде web-приложения.

## Список используемой литературы


1. Архитектура REST / [Электронный ресурс] URL: https://habr.com/ru/post/38730/, (дата обращения: 25.12.2022).
2. Гради Буч. Введение в UML от создателей языка / Гради Буч, Джеймс Рамбо, Ивар Якобсон. – ДМК Пресс, 2015. – 496 с.
3. Моделирование данных: обзор / [Электронный ресурс]. URL: https://habr.com/ru/articles/556790/, (дата обращения: 03.02.2023).
4. Обзор REST. Часть 3: создание RESTful сервиса на Spring Boot / [Электронный ресурс] URL: https://javarush.com/groups/posts/2488-obzor-rest-chastjh-3-sozdanie-restful-servisa-na-spring-boot, (дата обращения: 13.02.2023).
5. Основы правил проектирования базы данных / [Электронный ресурс]. URL: https://habr.com/ru/articles/514364/, (дата обращения: 12.01.2023).
6. Репин В.В., Елиферов В.Г. Процессный подход к управлению. Моделирование бизнес-процессов / Владимир Репин, Виталий Елиферов. – Манн, Иванов и Фербер, 2018. – С. 543.
7. Связи между таблицами базы данных / [Электронный ресурс]. URL: https://habr.com/ru/articles/488054/, (дата обращения: 05.01.2023).
8. Чистая Архитектура для веб-приложений / [Электронный ресурс]. URL: https://habr.com/ru/articles/493430/ (дата обращения: 01.02.2023).
9. Чистая архитектура. Искусство разработки программного обеспечения / Роберт Сесил Мартин. – Издательский дом «Питер», 2018. – 490 с.
10. Чистый код: создание, анализ и рефакторинг / Роберт Сесил Мартин. – Издательский дом «Питер», 2018. – 464 с.
11. Что такое JWT токен? / [Электронный ресурс]. URL: https://struchkov.dev/blog/ru/what-is-jwt/, (дата обращения: 15.04.2023).
12. Шилдт Герберт «Java. Полное руководство» / Герберт Шилдт. – Издательство «Альфа-книга», 2019. – 1488 с.

13. Alon Brody. SQL Vs NoSQL: The Differences Explained / [Электронный ресурс] URL: https://blog.panoply.io/sql-or-nosql-that-is-the-question, (дата обращения: 08.01.2023).

14. Apache Maven – основы / [Электронный ресурс]. URL: https:// oracleplsql.ru/postgresql-manual.html, (дата обращения: 03.02.2023).

15. BPMN Specification – Business Process Model and Notation / [Электронный ресурс] URL: https://www.bpmn.org/, (дата обращения: 06.04.2023).

16. Framework: что это такое и для чего нужен / [Электронный ресурс] URL: https://blog.skillfactory.ru/glossary/framework/, (дата обращения: 12.03.2023).

17. Mark Drake. SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems / Mark Drake / [Электронный ресурс] URL: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems, (дата обращения: 15.01.2023).

18. Mohit Malhotra. Everything About Software Architecture / [Электронный ресурс] URL: https://medium.com/ swlh/everything-aboutsoftware-architecture-dfd2b9351ef4, (дата обращения: 08.01.2023).

19. MySQL/Руководство для начинающих / [Электронный ресурс]. URL: https://wiki.gentoo.org/wiki/MySQL/Startup_Guide/ru, (дата обращения: 23.12.2022).

20. NoSQL: виды, особенности и применение / [Электронный ресурс]. URL: https://cloud.yandex.ru/blog/posts/2022/10/nosql, (дата обращения: 04.02.2023).

21. PostgreSQL учебник / [Электронный ресурс]. URL: https://habr.com/ru/articles/77382/, (дата обращения: 04.02.2023).

22. Spring Boot Reference Documentation / [Электронный ресурс] URL: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/, (дата обращения: 15.03.2023).

23. Spring Data JPA / [Электронный ресурс]. URL: https://habr.com/ru/articles/435114/, (дата обращения: 13.02.2023).

24. Vue.js для сомневающихся. Все, что нужно знать / [Электронный ресурс]. URL: https://habr.com/ru/articles/329452/, (дата обращения: 25.02.2023).

25. Vuetify – создаем свое простое приложение / [Электронный ресурс]. URL: https://habr.com/ru/articles/575050/, (дата обращения: 25.02.2023).