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»

|  |  |  |
| --- | --- | --- |
| Студент | А.А. Горбатюк  (И.О. Фамилия) | (личная подпись) |
| Руководитель | М.А. Тренина  (ученая степень, звание, И.О. Фамилия) | |
| Консультант | М.А. Тренина  (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:

1. Проанализировать существующие модели управления процессом записи сотрудников на медкомиссию;
2. Выделить требования, к приложению на основе потребностей пользователей;
3. Создать архитектуру приложения UML-диаграммы, на основе потребностей пользователей;
4. Выполнить программною реализацию выбранной 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.

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**

Speak English

Оглавление

[Введение 3](#_Toc133663276)

[Глава 1 Разработка модели управления процессом записи сотрудников на медкомиссию 6](#_Toc133663277)

[1.1 Анализ существующих моделей управления процессом записи сотрудников на медкомиссию 6](#_Toc133663278)

[1.2 Определение требований к разрабатываемой системе 7](#_Toc133663279)

[1.3 Постановка задачи на разработку 9](#_Toc133663280)

[1.4 Моделирование системы 10](#_Toc133663281)

[1.5 Разработка алгоритма распределения сотрудников на медкомиссию 13](#_Toc133663282)

[Выводы к главе 1 13](#_Toc133663283)

[Глава 2 Логическое проектирование модуля записи сотрудников на медкомиссию 14](#_Toc133663284)

[2.1 Выбор технологии логического моделирования 14](#_Toc133663285)

[2.2 Описание логической модели системы 15](#_Toc133663286)

[2.3 Создание логической модели данных 17](#_Toc133663287)

[Выводы к главе 2 20](#_Toc133663288)

[Глава 3 Программная реализация модели управления процессом записи сотрудников на медкомиссию 21](#_Toc133663289)

[3.1 Выбор архитектуры системы 21](#_Toc133663290)

[3.2 Выбор СУБД для внутренней базы данных 23](#_Toc133663291)

[3.3 Создание физической модели данных 25](#_Toc133663292)

[3.4 Реализация серверной части системы 27](#_Toc133663293)

[3.4.1 Выбор инструментов для реализации 27](#_Toc133663294)

[3.4.2 Определение структуры проекта 31](#_Toc133663295)

[3.4.3 Программная реализация компонентов сервиса 32](#_Toc133663296)

[3.5 Реализация клиентской части web-приложения 34](#_Toc133663297)

[3.5.1 Выбор инструментов для реализации 34](#_Toc133663298)

[3.5.2 Определение структуры проекта 36](#_Toc133663299)

[3.5.1 Программная реализация компонентов приложения 37](#_Toc133663300)

[3.6 Настройка сборки и разворачивания компонентов системы 39](#_Toc133663301)

[3.7 Описание функциональности системы 41](#_Toc133663302)

[3.8 Тестирование системы 42](#_Toc133663303)

[Выводы к главе 3 43](#_Toc133663304)

[Заключение 45](#_Toc133663305)

[Список используемой литературы 47](#_Toc133663306)

[Приложение К 50](#_Toc133663307)

[Приложение Л 51](#_Toc133663308)

[Приложение М 52](#_Toc133663309)

[Приложение Н 53](#_Toc133663310)

[Приложение П 54](#_Toc133663311)

[Приложение Р 55](#_Toc133663312)

[Приложение С 56](#_Toc133663313)

[Приложение Т 57](#_Toc133663314)

# 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.

# Разработка модели управления процессом записи сотрудников на медкомиссию

## 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.

## 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 50% 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 [12].

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. Однако универсальность часто относят к недостаткам UML, так как она приводит уменьшению точности отображения моделей [1].

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.

## 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.

## Моделирование системы

В первую очередь был проведён анализ областей ответственности для пользователей с разными ролями. Для наглядного представления была использована диаграмма прецедентов (см. рисунок 1.1), 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.

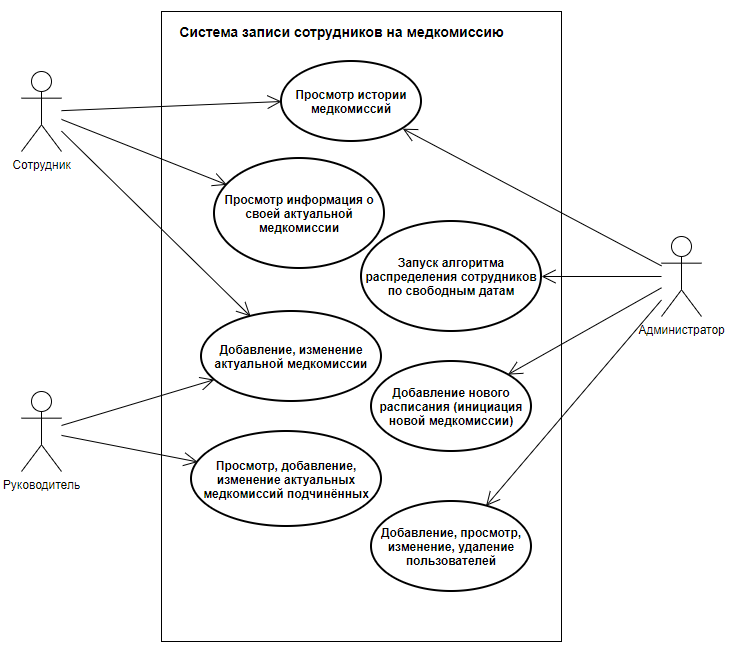


Рисунок 1.1 – Диаграмма прецедентов взаимодействия пользователей

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.

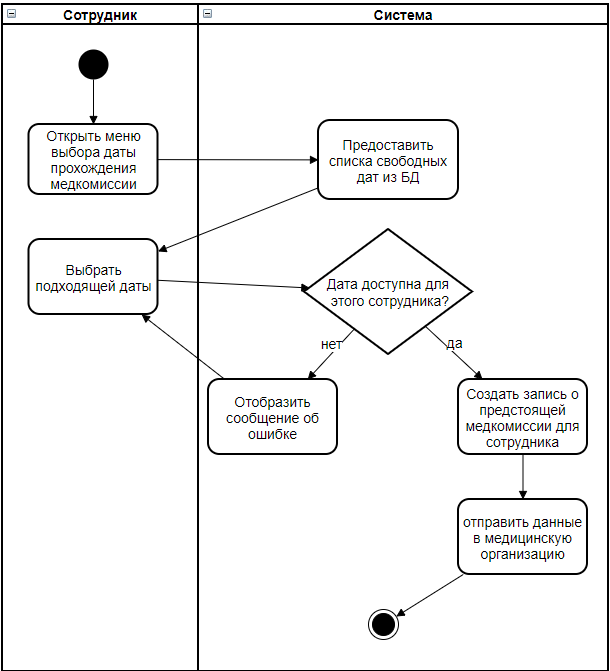


Рисунок 1.2 – Диаграмма видов деятельности «Запись на медкомиссию»

Условие доступности даты для сотрудника проверяет:

* кол-во 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.

## Разработка алгоритма распределения сотрудников на медкомиссию

Уникальной возможностью разрабатываемой системы является возможность автоматического равномерно распределения сотрудников, не записавшихся на конкретную дату, 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.

**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.

# Логическое проектирование модуля записи сотрудников на медкомиссию

## Выбор технологии логического моделирования

Как было описано в предыдущей главе, методология UML является достаточно универсальной и ее можно использовать на всех этапах моделирования будущей системы. Она довольно часто используется для описания структуры данных, отдельных сущностей, связей между ними. Соответственно для логического проектирования была также использована эта методология и подробный анализ других подобных решений не требуется. Это способствует сохранению общей концепции проектирования и пониманию системы в целом, так как использование одной технологии требует меньших усилий на понимание взаимосвязи разных моделей.

В то же время в качестве методологии для моделирования данных была использована IDEF1X, как наиболее современная и популярная методология, которая дает представление, независимое от конкретной реализации СУБД. Хотя данный стандарт и относится к семейству языков моделирования IDEF, концептуально он достаточно схож с подходом UML и не выбивается из общей концепции, представленной в работе. В IDEF1X таким же образом, как и в UML идет оперирование сущностями, их атрибутами и связями между сущностями.

В качестве инструмента для реализации диаграмм данных была выбрана программа Toad Data Modeller. Данная программа ориентирована на разработку моделей баз данных. Она позволяет производить логическое моделирование данных и физическое для конкретной СУБД. В данной программе в качестве основного языка моделирования используется непосредственно IDEF1X.

## Описание логической модели системы

Основным источником данных для любой работающей на текущий момент системы является база данных университета. Соответственно вся доступная информация ограничена данными, содержащимися в ней. Основная база данных построена на основе объектно-реляционной системы управления базами данных Oracle Database. В ходе работы был проведен анализ данных, содержащихся в базе, и определены основные таблицы и структура данных, хранящихся в них [4]. Визуально структура базы данных представлена на рисунке 1.

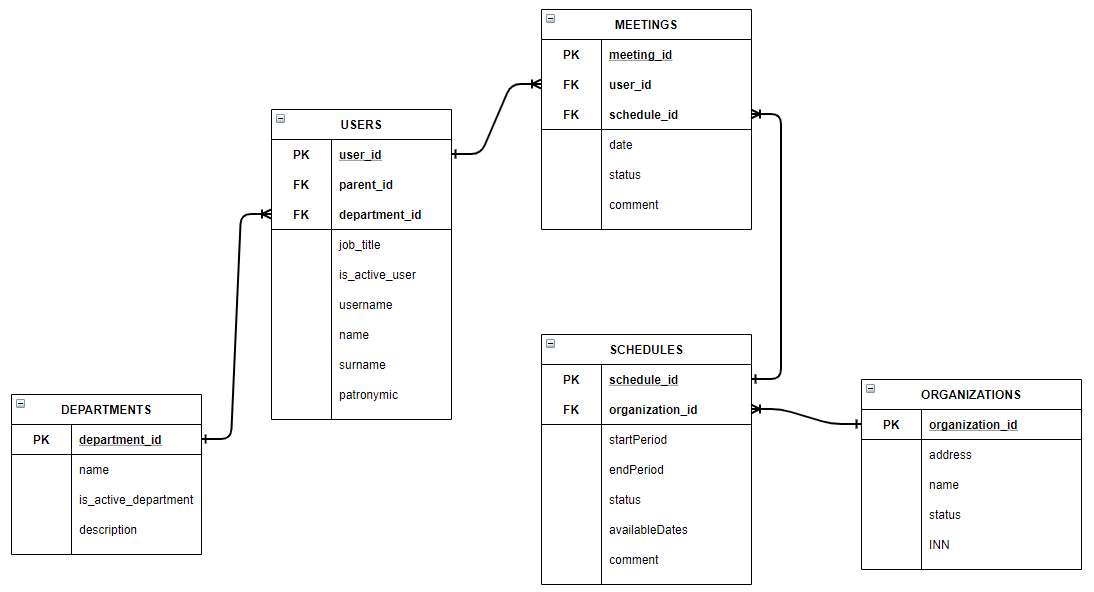


Рисунок 1 - Модель базы данных

В базе данных выделены 7 таблиц: преподаватели, группы, институты, кафедры, пары, аудитории, дисциплины. При анализе модели было выявлено несколько проблем организации базы данных:

* запись для пары имеет сложный составной ключ.
* одно занятие может проводиться для нескольких групп одновременно, что не отражено в базе данных университета, и, следовательно, одна пара может отражаться несколькими записями.
* аудитории привязаны к кафедрам, что затрудняет их поиск, так как логика распределения аудиторий по кафедрам достаточно запутана;
* у групп отсутствуют данные о курсе.

Соответственно существует необходимость преобразования этих данных в новую форму, что и будет рассмотрено далее.

Таким образом, в общем виде работа с данными в системе происходит следующим образом:

* производится чтение из базы данных ТГУ;
* считанные данные преобразовываются в формат, необходимый для дальнейшей работы;
* преобразованные данные сохраняются в новой базе согласно определенной модели. При этом для некоторых сущностей проставляется метка, показывающая необходимость отображения;
* данные из новой базы по запросу отправляются пользователю.

На основе этой схемы была создана диаграмма классов (см. приложение К) для более четкого представления потока данных. На этой диаграмме изображены основные сущности, которыми оперирует система. Объекты с пометкой Ext оперируют данными из внешней системы. При этом на ней отображено лишь концептуальное представление компонентов в виде абстрактных классов, для понимания общей структуры. Более детальная проработка производилась на этапе реализации, так как существует зависимость от общей программной архитектуры.

На диаграмме отмечен управляющий Application класс, который обеспечивает разделение логики на две части. Первая работает с внешними данными, вторая - со внутренними. При этом у них присутствует общая часть – Repository. Классы, относящиеся к данному типу, обеспечивают работу с внутренней базой данных, для одной части на ввод, для другой на вывод.

Классы типа ExtService содержат методы для чтения и преобразования данных из внешней базы. Как видно на диаграмме, внешние сущности почти полностью соответствуют сущностям, хранящимся в базе данных. После преобразования данные записываются во внутреннюю базу с использованием класса типа Repository.

Классы типа Service отвечают за чтение данных из внутренней базы и передачи их для дальнейшей работы остальным компонентам. При этом структура связей между сущностями, которыми оперируют данные классы, является более сложной, так как на выход отдаются полные блоки данных. Также у данных классов появляется функционал для работы с пользователями и скрытыми сущностями. Все сущности были определены согласно требованиям, а также согласно тому, как данные должны быть отображены.

Все внутренние сущности связаны между собой отношением ассоциации. Остальные сущности связаны отношением зависимости.

Таким образом, в плане организации потока данных система имеет один вход для получения данных и один выход. При этом производится буферизация данных, что позволяет двум этим модулям работать независимо.

## Создание логической модели данных

После описания взаимодействия компонентов и существующих в системе сущностей было произведено создание логической модели данных, которая затем легла в основу физической модели.

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

На текущий момент в мире технологий баз данных существует два основных направления: SQL и NoSQL (реляционные и нереляционные) базы данных. Различия между ними заключаются в том, как они спроектированы, какие типы данных поддерживают, как хранят информацию.

Реляционные БД хранят структурированные данные, которые обычно представляют объекты реального мира. В них хранятся данные, сгруппированные в таблицах, формат которых задан на этапе проектирования хранилища.

Нереляционные БД устроены иначе. Например, документо-ориентированные базы хранят информацию в виде иерархических структур данных. Речь может идти об объектах с произвольным набором атрибутов. То, что в реляционной БД будет разбито на несколько взаимосвязанных таблиц, в нереляционной может храниться в виде целостной сущности.

SQL базы данных применяются в случае необходимости соответствия базы данных требованиям ACID (атомарность, непротиворечивость, изолированность, долговечность), что позволяет уменьшить вероятность неожиданного поведения системы и обеспечить целостность базы данных. Либо если данные, с которыми работает система, структурированы и при этом структура не подвержена частым изменением.

NoSQL базы данных применяются при необходимости хранения больших объемов неструктурированной информации. Подобные базы не накладывают ограничений на типы хранимых данных. Более того, при необходимости в процессе работы можно добавлять новые типы данных. Также они обеспечивают лучшую скорость разработки [10].

Согласно определенной выше модели данные в проекте имеют высокую степень структурированности. Из общего потока достаточно просто выделить отдельные сущности, связи между ними понятны. Соответственно наилучшим выбором в данном случае являются реляционные базы данных.

Соответственно согласно выбранному типу базы данных были выделены сущности и связи между ними (рисунок 2). Всего модель содержит 10 сущностей, 2 из которых являются техническими и не связаны с остальными. При этом все связи кроме одной представляют тип «один-ко-многим». Связь между группами и занятиями представлена в виде многие-ко-многим, так как одно занятие может проводиться для нескольких групп одновременно.

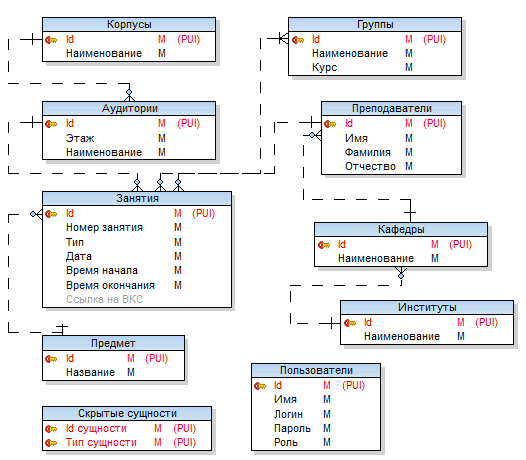


Рисунок 2 - Логическая модель данных

Если сравнивать модель с той, что представляла базу данных университета, можно отметить несколько отличий:

* добавлена таблица «Корпусы», для группировки аудиторий не по кафедрам, а непосредственно по зданиям, в которых они находятся;
* переработана модель занятия, которая теперь содержит единый идентификатор для лучшей организации связей, а также реализации объединения дубликатов занятий в одну сущность;
* для групп добавлен курс, к которому они относятся;
* были добавлены две технические таблицы: «Скрытые сущности» и «Пользователи».

**Выводы к главе 2**

В данной главе была рассмотрена логическая модель системы поиска и отображения расписания учебных занятий.

Первым делом был произведен выбор технологий моделирования. Для моделирования компонентов системы и потоков данных использовался UML. Для реализации непосредственно логической модели данных был выбран IDEF1X – современный инструмент моделирования, заточенный непосредственно на работу с базами данных.

После выбора технологий моделирования было произведено моделирование работы с данными в системе. Была выявлена структура базы данных на стороне ТГУ, из которой непосредственно система и получает всю информацию. Были выявлены основные модули системы, общий поток данных и основные сущности, которые в ней используются. Был сделан вывод, что в системе существует 2 независимых потока данных с общим буфером. Один поток производит чтение из внешней базы данных в буферную. Второй осуществляет чтение из буферной базы и передачу пользователю.

На основе полученной информации была разработана логическая модель данных. Были выделены основные сущности и связи между ними. Полученная модель ляжет в основу физической модели данных для конкретной СУБД.

# Программная реализация модели управления процессом записи сотрудников на медкомиссию

## Выбор архитектуры системы

За годы развития ПО разработчикам удалось придумать надежные подходы, чтобы устранить недостатки проектирования без архитектуры. Ниже представлены некоторые из самых известных:

* многослойная архитектура (Layered Architecture),
* многоуровневая архитектура (Tiered Architecture),
* сервис-ориентированная архитектура (Service Oriented Architecture — SOA),
* микросервисная архитектура (Microservice Architecture).

Многослойная архитектура работает по принципу разделения ответственностей. ПО разделено на слои, лежащие друг на друге, и каждый из них выполняет определенную обязанность. Архитектура подразумевает наличие трех базовых слоев: представления, бизнес-логики и передачи данных. Данные и элементы управления проходят через каждый слой и передаются от одного к другому. Эта система также повышает уровень абстракции и в некоторой степени даже стабильность ПО. Подобная архитектура подходит для небольших монолитных приложений.

Многоуровневая архитектура разделяет комплекс ПО на уровни по принципу взаимодействия “клиент-сервер”. Архитектура может иметь один, два и больше уровней, разделяющих ответственности между поставщиком данных и потребителем. Этот подход использует шаблон «Request Response» (запрос-ответ) для связи между уровнями. В отличие от многослойной архитектуры, он предлагает масштабируемость. Основным преимуществом данной архитектуры является возможность реализации распределенных систем, в которых необходима слабая зависимость интерфейса и бизнес логики.

Сервис-ориентированная архитектура состоит из компонентов и приложений, которые связываются друг с другом с помощью четко определенных сервисов. Клиент отправляет запрос с использованием стандартного протокола и формата данных по сети. Этот запрос обрабатывается ESB (enterprise service bus - сервисная шина предприятия), которая считается сердцем сервис-ориентированной архитектуры. Данная архитектура позволяет создавать системы, компоненты которой находятся в достаточной степени изоляции. Преимуществом подобного подхода является обеспечение высокой степени безопасности системы. В то же время структура системы в таком случае является наиболее сложной и запутанной.

Наконец микросервисная архитектура подразумевает реализацию приложения в виде набора небольших сервисов, каждый из которых работает в собственном процессе и связывается с легковесными механизмами. Эти сервисы основываются на бизнес-возможностях и могут развертываться независимо друг от друга с помощью полностью автоматизированного механизма. Централизованное управление между сервисами минимально. Они могут быть написаны на разных языках, использовать разные технологии хранения данных. Преимуществами архитектуры является слабая связанность компонент, модульность, высокая гибкость и масштабируемость. Микросервисная архитектура наиболее сложна для реализации. При разработке ПО необходимо досконально продумать взаимодействие микросервисов, способ обмена данными, учесть задержки сети [18].

Согласно информации, полученной в предыдущих главах, разрабатываемая система является распределенной, состоящей из нескольких приложений. Соответственно наиболее верным будет решение разделить ее на серверные и клиентские модули. Это позволит исключить дублирование функциональной логики, сохранить единообразие приложений. В то же время система состоит из ограниченного числа компонент. Также для системы не требуется обеспечение высокого уровня безопасности из-за отсутствия в ней доступа к приватным данным. Из всего выше перечисленного следует, что наиболее оптимальной будет многоуровневая архитектура. Данная архитектура позволит реализовать систему с общим сервером для обработки логики и множеством клиентских приложений, которые будут получать от него данные.

Далее согласно многоуровневой архитектуре было необходимо определить компоненты системы. В процессе анализа разрабатывались диаграммы последовательности, для документации компонент и взаимодействия между ними. Наиболее показательными являются диаграммы, описывающие процесс поиска (см. приложение Л) и процесс входа в систему (см. приложение М). На них приведены модели взаимодействия пользователя с web-приложением.

В рамках данных прецедентов взаимодействие пользователя с системой производится посредством браузера, который инициирует события, обрабатываемые web-приложением. Web-приложение реализует логику взаимодействия с сервером, в нем производится конвертация данных, производится отрисовка представления. Следующим компонентом является REST-сервер, реализующий основную логику, организующий взаимодействие с базами данных. Последним компонентом системы является непосредственно база данных.

## Выбор СУБД для внутренней базы данных

Первым делом были выделены основные требования к СУБД. Ранее было определено, что в системе должна использоваться реляционная база данных в связи с организацией структуры самих данных. Соответственно было необходимо производить поиск среди реляционных систем управления базами данных (РСУБД).

Также в связи с тем, что проект является некоммерческим, все компоненты в нем, за исключением используемых непосредственно в информационной системе университета, должны поставляться со свободной лицензией либо бесплатной лицензией для некоммерческого использования.

Помимо этого, к дополнительным требованиям можно отнести скорость и простоту разворачивания, постоянное развитие, поддержка современных стандартов для дальнейшей поддержки и расширения.

По состоянию на 2020 год самыми популярными РСУБД являются Oracle, MySQL, MS SQL Server, PostgreSQL, MongoDB, IBM Db2, SQLite. Из них некоммерческую лицензию имеют MySQL, PostgreSQL и SQLite, соответственно выбор производился среди них.

MySQL, распространяемая компанией Oracle, является одной из самых простых в освоении и управлении СУБД. Отличительными ее особенностями являются простая установка и подключение, мощность, большое количество функций, высокая скорость, достигаемая пренебрежением некоторыми стандартами. Основными недостатками являются относительно низкая надежность, застой в разработке.

PostgreSQL является open-source (СУБД с открытым исходным кодом) СУБД, то есть разрабатывается силами сообщества. Является наиболее продвинутой СУБД из списка представленных. Основной ее отличительной особенностью является полное соответствие стандартам, что способствует значительной надежности и определяет высокую степень совместимости со многими другими решениями, хотя и влияет на скорость работы. Также для данной СУБД постоянно выпускаются обновления с исправлением ошибок и добавлением функционала.

SQLite также является open-source СУБД. SQLite является встраиваемой СУБД, что означает, что она разворачивается вместе с приложением, которое ее использует. SQLite является наиболее простой в установке, управлении и обслуживании среди всех представленных. Однако она обладает очень скромным функционалом и не может применяться для хранения больших объемов данных [19].

В связи со скромными возможностями SQLite выбор стоял между MySQL и PostgreSQL. В процессе анализа преимуществ и недостатков данных СУБД предпочтение было отдано следованию стандартам и поддержке, то есть PostgreSQL.

## Создание физической модели данных

На основе логической модели была разработана физическая модель данных. Реализация модели так же производилась с использованием Toad data modeler. Итоговая модель представлена на рисунке 3.

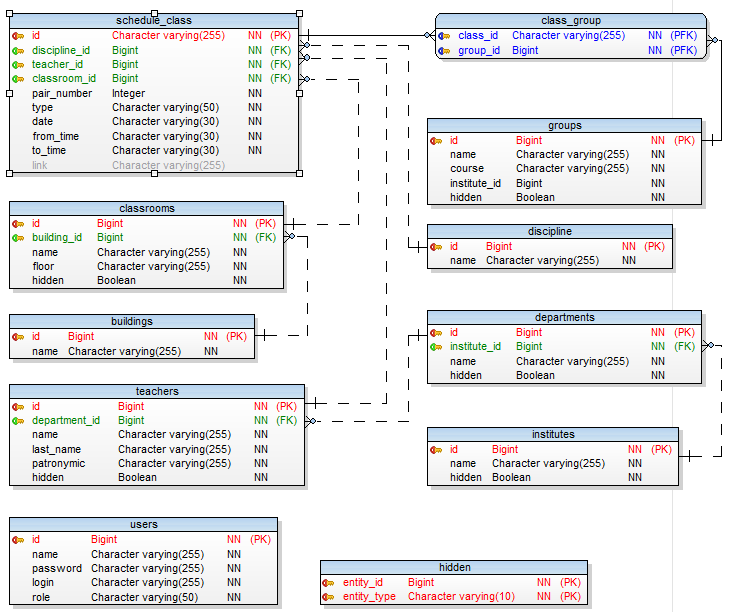


Рисунок 3 - Физическая модель данных

В первую очередь сущности логической модели были преобразованы в сущности физической модели. При этом модель постоянно перерабатывалась до момента ее приведения к третьей нормальной форме. Для каждого атрибута был выделен тип данных, выбранный среди возможных значений в СУБД. Каждый идентификатор, за исключением ключа сущности schedule\_class, имеет числовой тип Bigint и, либо генерируется автоматически, либо используется значение из родительской базы данных. Идентификатор для schedule\_class генерируется непосредственно в процессе работы системы.

Также в модели были предусмотрены дополнительные системные атрибуты. Например, флаги hidden булевого типа, для маркировки сущности как скрытой.

После создания всех сущностей были определены связи между ними. В логической модели присутствовала одна связь типа один-ко-многим между парами и группами. Она была преобразована посредством внедрения дополнительной таблицы, предназначенной для организации данной связи, которая хранит идентификатор пары и группы. Все остальные связи представлены типом один-ко-многим. При этом для реализации каждой связи сущностям были добавлены дополнительные атрибуты, представляющие ссылки на связанные сущности.

В заключение стоит также отметить, что в большинстве сущностей они представлены отдельным полем. В то же время в модели существуют две сущности, представленные составным идентификатором: связующая таблица пары и группы и сущность hidden, хранящая идентификатор скрытой сущности и ее тип.

## Реализация серверной части системы

### Выбор инструментов для реализации

Серверная часть является ядром системы. Она отвечает за реализацию всей основной логики, подключается к базам данных, отправляет данные клиентам.

Ранее уже было определено, что из-за отсутствия аналогов все компоненты системы должны быть разработаны с нуля. Соответственно выбор технологий разработки был достаточно свободный. В то же время следовало задать базовые требования для более эффективного выбора. В первую очередь платформа, на которой базируется разработка приложения должна поддерживать REST подход, что, впрочем, реализуют практически все современные технологии [6]. Также необходима удобная работа с базой данных, маршрутизацией запросов. В то же время выбранный инструментарий не должен быть перегружен лишним функционалом, быть эффективным по оперативной памяти и процессорному времени, что, однако, не должно сильно отражаться на удобстве разработки. То есть необходимо найти баланс удобства разработки, поддержки и конечной производительности.

В качестве основных платформ рассматривались самые популярные серверные платформы, а именно JVM, .NET, Go, Python, Node js. В первую очередь был отброшен Python. Причиной этому служил нестандартный синтаксис самого языка, а также его скриптовая природа. Несмотря на его простоту и популярность фреймворка Django, в проекте хотелось сохранить более-менее единообразную структуру. Следующим произошел отказ от Go. Несмотря на то, что этот язык с каждым годом набирает все большую популярность, он предназначен для достаточно специализированных вещей и обычно используется при реализации микросервисной архитектуры. Оставшиеся три платформы требовали более детального изучения.

Node js – платформа на основе Chromium, предназначенная для выполнения JavaScript-кода вне браузера. На текущий момент является одной из самых популярных скриптовых серверных платформ. Отличительными особенностями являются низкий порог входа, поддержка асинхронного выполнения кода, большое количество библиотек. В то же время основными недостатками платформы является высокий уровень потребления оперативной памяти, отсутствие строгой типизации, относительно низкая скорость вычислений, если рассматривать в сравнении с компилируемыми языками программирования. Данная платформа подходит для реализации небольших приложений, либо отдельных сервисов при реализации микросервисной архитектуры. Отличается высоким уровнем масштабируемости [14].

JVM (Java virtual machine) является платформой для исполнения Java байт кода. Помимо основного языка платформы Java на ней могут исполняться также программы, написанные на других языках программирования, относящихся к данному семейству, например, Kotlin, Groovy, Scala и прочие. Вокруг платформы образовалось мощное сообщество, состоящее как из отдельных разработчиков, так и из крупных компаний. JVM задумывалась как мультиплатформенный инструмент. Языки, что исполняются на ее основе, теряют небольшой процент производительности из-за промежуточной компоненты в системе, поэтому обычно в системах, где делается особая ставка на производительность используются другие инструменты [5].

.NET является аналогом JVM, выпущенным компанией Microsoft. Данная платформа имеет те же особенности, что и ее аналог, а именно кроссплатформенность, исполнение программ, написанных на разных языках программирования. Основными языками программирования, используемыми в ней, являются C#, Visual Basic .NET, F#. Если брать основной язык платформы C#, то от конкурентного языка Java он отличается большим количеством современных функций, более частыми обновлениями [13].

При детальном анализе можно сделать вывод, что при реализации текущей системы использование Node js в центральной компоненте системы не является оптимальным. В то же время выбор между платформами JVM и .NET был более сложным из-за сильной схожести данных платформ. С одной стороны, C# на текущий момент выглядит более привлекательным, чем Java за счет поддержки самых современных функций. С другой стороны, у обеих платформ есть более продвинутые языки. Kotlin и F# выглядят более перспективными. Однако статус F# на текущий момент не определен, у него достаточно слабая поддержка. В то же время Kotlin является одним из самых быстро набирающих популярность языков программирования. А полная совместимость с Java позволяет использовать инструментарий обоих языков, что делает Kotlin более универсальным.

По итогам анализа выбор был остановлен на платформе JVM и языке программирования Kotlin. Данный язык обладает всеми преимуществами Java, и в то же время использует лучшие практики, реализованные в других популярных языках программирования.

Для большего удобства разработки было решено применить серверный фреймворк. На данный момент для JVM разработано большое количество фреймворков. Наиболее известными фреймворками являются Java EE, Spring Boot, Micronaut, Quarkus. Также существует большое количество микрофреймворков, известность которых уже ниже, однако среди их представителей можно отметить Helidon SE и Ktor. Стоит отметить, что за исключением Java EE и Spring Boot, все фреймворки были выведены в релиз в 2018-2019 годах. Это значит, что большинство из них лишь набирает популярность. В то же время Java EE постепенно заменяется другими инструментами.

Если производить прямое сравнение стандартных фреймворков с микрофреймворками, то можно выделить несколько основных отличий:

* микрофреймворки обладают значительно меньшим функционалом в сравнении со стандартными фреймворками, что компенсируется большими возможностями их настройки;
* полноценные фреймворки активно используют внедрение зависимостей в разном исполнении.
* микрофреймворки за счет урезанного функционала обладают значительно лучшим быстродействием.

Для более точного сравнения характеристик фреймворков был произведён анализ технических характеристик, которые выдают одинаковые сервисы, реализованные с их использованием. Характеристики представлены в таблице 1. В ней отражены размеры артефактов, время первого запуска сервиса, минимальный объём heap-памяти (-Xmx), необходимый для запуска работоспособного (отвечающего на запросы) сервиса, минимальный объём heap-памяти, необходимый для прохождения нагрузочного теста 50 пользователей \* 1000 запросов и минимальный объём heap-памяти, необходимый для прохождения нагрузочного теста 500 пользователей \* 1000 запросов [8].

Таблица 1 - Характеристики java фреймворков

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Фреймворк | Размер артефакта, Мбайт | Время запуска, секунды | Минимальный объём heap-памяти, Мбайт | | |
| Запуск | 5 \* 1000 | 500 \* 1000 |
| Helidon SE | 16,6 | 2,2 | 9 | 9 | 11 |
| Ktor | 20,9 | 1,4 | 11 | 11 | 13 |
| Micronaut | 16,5 | 4,0 | 13 | 13 | 17 |
| Spring Boot | 42,7 | 10,2 | 22 | 23 | 25 |

При анализе данных представленных в таблице можно прийти к закономерному выводу, что полноценные фреймворки показывают не лучшую производительность. В то же время Helidon SE и Ktor показывают примерно одинаковые результаты.

Решающим при выборе фреймворка был тот факт, что в отличие от Helidon Ktor заточен непосредственно на работу с Kotlin. Соответственно перевес оказался на его стороне.

Ktor разработан компанией JetBrains, которая также является родоначальницей языка Kotlin. Соответственно фреймворк использует все возможности языка. Помимо всего прочего Ktor содержит модули не только для организации серверного приложения, но может выступать и как клиентская библиотека, за что и получил признание в сообществе разработчиков мобильных приложений [16]. Также компанией был разработан еще один инструмент – exposed. Exposed является реализацией JDBC паттерна и используется для более удобной работой с базами данных. Данный инструмент позволяет работать с сущностями баз данных как с объектами, поля которых строго типизированы. Он облегчает реализацию запросов, конвертацию данных, контроль за потоками данных [10].

### Определение структуры проекта

Корнем Ktor-приложения является функция расширения базового класса Application. В ней производится инициализация всех зависимостей, запуск базовых функций и инициализация маршрутизации. За пределами данной функции самим фреймворком не накладывается никаких ограничений, соответственно можно реализовать любую структуру.

Было решено использовать базовый подход с разделением на различные базовые компоненты: Route, Service, Repository, Model, Util. Компоненты типа Route отвечают непосредственно за маршрутизацию. В них производится реализация соответствия определенному адресу нужной функции. Компоненты типа Service отвечают непосредственно за реализацию основной логики приложения. Компоненты типа Repository отвечают за работу с базами данных. Таким образом происходит разделение ответственности различных компонентов сервиса. Компоненты типа Model являются классами, которые отображают модели данных. И компоненты Util содержат дополнительные функции.

Помимо этого, проект содержит различные конфигурационные файлы, константы, перечисления и т.д.

### Программная реализация компонентов сервиса

В первую очередь был описан непосредственно корневой Application модуль, подключены необходимые зависимости.

Далее были определены таблицы базы данных в объектном виде. Как было отмечено ранее, Exposed позволяет работать с сущностями базы данных как с обычными объектами. Соответственно были определённые объекты, представляющие сущности, для каждого поля было задано наименование и тип данных.

Также были определены модели сущностей, которые приходят из базы данных университета. Они были представлены в виде стандартных data-классов.

После создания сущностей можно было переходить непосредственно к организации подключения к базе данных. Для этого был создан отдельный вспомогательный класс DBWorker. В нем производится подключение к базе данных по заданным параметрам, инициализация таблиц. При этом при подключении к базе данных используется драйвер, который позволяет реализовывать универсальное подключение для использования Exposed.

Далее были реализованы так называемые ext-сервисы. Данные сервисы осуществляют работу с внешней системой, чтение из внешней базы данных и преобразование полученных сущностей в требуемые. Для конвертации в каждом классе были реализованы отдельные методы с определёнными правилами преобразования сущностей. Так как по большей части имена и прочие характеристики сущностей в системе университета стандартизированы, были выделены правила, по которым составляются характеристики сущностей. Например, для аудиторий было выделено, что первым в наименовании аудитории идет название корпуса, затем в числовом индексе идет первая цифра – номер этажа, остальные две цифры – номер аудитории. Однако в процессе работы системы было выявлено, что не все сущности соответствуют подобным правилам, поэтому для них была выделена отдельная подгруппа. Подобные правила были выделены для всех типов сущностей.

Также для работы с внутренней базой данных были выделены repository-классы. Данные классы реализовывали методы взаимодействия с базой данных. В приложении Н в качестве примера представлен метод получения кафедр по идентификатору института. Также не нем можно увидеть, как в Exposed реализована работа с запросами. Запрос представлен стандартным SQL-синтаксисом, обернутым в функциональный стиль. Внутри функций производится проверка типов, преобразование в SQL-функции с экранированием входных параметров для защиты от инъекций. Результатом подобного запроса является строго типизированный словарь, из которого по ключам можно получить значения.

Далее были реализованы service-классы, отвечающие за основную логику. В них реализованы методы, вызываемые при переходе на определенный URL-адрес. При этом производится преобразование входных параметров, вызов методов repository-классов.

По завершении реализации всех сервисов была организована маршрутизация. Было смоделировано дерево маршрутов, определены функции для каждого маршрута.

Далее была реализована логика для администратора. Работа производилась по той же схеме. Однако дополнительно была реализована аутентификация и авторизация. Авторизация реализовывалась посредством JWT (JSON Web Token). При вызове функции логина производится проверка существования пользователя с переданными данными в системе. Если пользователь существует, производится генерация токена на основе данных о пользователе. Данный токен передаётся во всех защищённых запросов для проверки доступа для пользователя. Для исключения дублирования логики проверки токена для каждого запроса была разработана функция-обертка для защищенных маршрутов. Данная функция принимает на вход список ролей, которым доступен маршрут, а также список маршрутов. Она извлекает из заголовков запроса токен, проверяет его на корректность, выделяет роль пользователя и сравнивает ее с поступившим списком. Если роль отсутствует во входных параметрах, выбрасывается исключение. В противном случае управление передаются методу, соответствующему маршруту.

В завершении все реализованные модули были связаны воедино. Реализовано обновление базы данных по расписанию. За заданный промежуток времени производится вызов методов обновления. При этом производится проверка на доступность внешней базы данных. Если подключение недоступно, выбрасывается исключение и система продолжает работать в автономном режиме.

## Реализация клиентской части web-приложения

### Выбор инструментов для реализации

На данный момент существует два основных подхода для реализации web-приложения. Первый подход заключается в создании шаблонов, которые заполняются данными на серверной части и отправляются клиенту для отображения в виде статичных данных. Второй подход подразумевает формирование статики на стороне клиента. Первый подход снимает нагрузку с клиентской машины на рендер. В то же время повышается нагрузка на передачу данных в сети. Второй подход требует большей вычислительной мощности от клиентской машины, но в то же время позволяет увеличить быстродействие самого приложения.

Быстродействие современных устройств позволяет нормально работать с клиентскими web-приложениями. В свою очередь сетевое соединение в кампусе университета является нестабильным, особенно при больших нагрузках, соответственно вариант с клиентским рендерингом является более предпочтительным.

Современные технологии реализации web-приложений базируются на языке JavaScript и основанных на нем инструментах. Данные приложения можно реализовывать непосредственно на чистом JavaScript, но в большинстве проектов используются специализированные фреймворки. Это происходит из-за проблем с поддержкой различными браузерами разных версий языка. Фреймворки облегчают управление состоянием элементов приложения, снимают часть работы по отрисовке.

На данный момент существует так называемая большая тройка фреймворков: Angular 2+, React и Vue. Данные фреймворки являются самыми популярными в web-сообществе.

Angular 2+ - фреймворк, разработанный компанией Microsoft. Его отличительной особенностью является наличие инъекций зависимостей, что приближает его к серверным фреймворкам, таким как Spring Boot. Angular 2+ содержит большое количество различных инструментов для управления состоянием, организации маршрутизации, управления элементами интерфейса и т.д. Это делает его хорошим выбором для больших приложений, в которых делается упор на организацию разработки, а конечная производительность не сильно важна.

React - фреймворк разработанный компанией Facebook. Он содержит базовые функции управления состоянием и отображением элементов. Все остальные функции реализуются посредством интеграции различных библиотек. Это делает его более универсальным инструментом, так как для каждого проекта можно реализовать индивидуальную конфигурацию. Это является одновременно и преимуществом, и недостатком, так как для настройки эффективного проекта необходимы определенные навыки.

Vue является продуктом, реализованным сообществом разработчиков. Данный фреймворк создавался как более легковесная альтернатива Angular. Он также имеет в базовой конфигурации различные инструменты для управления состоянием, отображением, маршрутизацией. В то же время в нем было решено отказаться от реализации инъекций зависимостей. Обычно он применяется в проектах средней величины, где нужно соблюсти баланс скорости разработки и конечной производительности.

Таким образом, основной выбор стоял между Vue и React из-за их легковесности. Окончательным выбором стал React, так как его конфигурацией достаточно просто управлять, что способствует легкости модернизации конечного продукта.

Также было решено использовать в проекте в качестве основного языка TypeScript. TypeScript - это расширенная версия JavaScript, вводящая в разработку принципы строгой типизации. Данный выбор обусловлен, во-первых, тем, что это позволяет писать более качественный код, избежать ошибок, которые связаны с некорректными типами, способствует более четкому пониманию кода. Во-вторых, использование строго типизированного языка программирования при разработке клиентского приложения способствует общему единообразию программной системы, позволяет провести аналогии с сущностями, которыми оперирует сервер.

### Определение структуры проекта

Как и в предыдущем примере в проекте существует корневой компонент, к которому идет подключение остальных модулей. Корневыми компонентами по части представления являются отдельные View или страницы.

Страницы компонуются из других модулей – компонент. Компоненты содержат в себе файлы с реализованным представлением и стилями. При этом компоненты разделены на общие, управляемые и компоненты-представления. Общие компоненты используются в разных частях проекта. Компоненты-представления не содержат в себе никакой логики и являются стилизованными элементами. Управляемые компоненты содержат помимо представления бизнес-логику.

Связь с сервером организована в API-модуле. В нем определена конфигурация подключения и непосредственно функции запросов к серверу. Также в проекте присутствуют дополнительные модули с утилитами, моделями, константами и т.п.

Также для разворачивания приложения был добавлен небольшой серверный модуль, который возвращает статику приложения для загрузки в браузере.

### Программная реализация компонентов приложения

Для начала были реализованы компоненты-представления. Все страницы и компоненты представляют обычные функции, в которых используются React hooks для управления состоянием. Для реализации глобального состояния приложения был использован React context [17]. Дополнительные библиотеки не применялись за ненадобностью.

Маршрутизация была реализована с помощью библиотеки React Router. Данная библиотека позволяет сопоставить компоненты и URL адреса, которым они соответствуют. При этом разным маршрутам могут соответствовать одни компоненты. Например, список институтов и список корпусов являются одной страницей, для которой делаются разные запросы на сервер. Для этого при сопоставлении маршрутов в компоненты пробрасываются определенные флаги, показывающие принадлежность к той или иной группе.

При стандартной маршрутизации все необходимые компоненты уже загружены браузером и переключение между страницами происходит моментально. Однако в общий код web-приложения также входит панель администратора, доступ к которой необходим ограниченному числу пользователей. Соответственно для нее была предусмотренная так называемая «ленивая загрузка». То есть модули панели администратора загружаются только при прямом переходе к ней, что благополучно сказывается на использовании трафика остальных пользователей.

Для того чтобы приложение можно было развернуть на терминалах, были внесены дополнительные модификации. Версия для терминала отображается по адресу /terminal. Соответственно была введена проверка для данного адреса и, если она возвращает истину, производится блокировка некоторых частей интерфейса. Например, скрываются все ссылки на внешние ресурсы.

Для реализации http-запросов была использована библиотека axios. Данная библиотека является оберткой над XMLHttpRequest, стандартным инструментом JavaScript. Ее преимуществом является поддержка большинства браузеров, удобство реализации запросов. Все запросы были разделены на две части: публичные и приватные. Приватные запросы – те, которые требуют авторизации пользователя. Для них был реализован перехват и обработка ошибок. Для публичных запросов была реализована обработка случая отмены запроса. Для приватных запросов была также добавлена проверка на возврат ответа с кодом статуса 401, что означает, что пользователь не является авторизованным. В этом случае удаляется токен и производится перенаправление на главную страницу. Все запросы выполняются в асинхронном режиме, то есть в процессе их выполнения главный поток не блокируется, и в этот момент система может выполнять другие функции. По завершении запроса полученный результат возвращается в главный поток.

Аутентификация пользователя производится с использованием формы входа. Введенные пользователем логин и пароль отправляются на сервер. Сервер возвращает токен авторизации, который сохраняется в браузере и впоследствии отправляется в заголовках для каждого приватного запроса.

Серверная компонента основана на node js. Небольшой сервер слушает все запросы, на каждый из них возвращает всю статику и возвращает index.html файл. Это необходимо в связи с особенностями работы React-приложения. При переходе по какой-либо ссылке браузер пытается найти в каталоге сервера соответствующий html-файл. Однако в подобных приложениях URL адрес не отражает файловую структуру, а используется как маркер для отрисовки соответствующего представления.

## Настройка сборки и разворачивания компонентов системы

Для организации развертывания системы была реализована диаграмма развертывания (см. приложение Р). На диаграмме представлена структура взаимодействия всех компонент за исключением мобильного приложения. Разворачивание идет на общем физическом сервере. При этом, как было определено ранее, не должно существовать зависимости от операционной системы и прочих характеристик. Система должна быть переносимой. В связи с этим для разворачивания компонентов было решено использовать Docker.

Docker - программное обеспечение с открытым исходным кодом, применяемое для разработки, тестирования, доставки и запуска веб-приложений в средах с поддержкой контейнеризации. Он нужен для более эффективного использования системы и ресурсов, быстрого развертывания готовых программных продуктов, а также для их масштабирования и переноса в другие среды с гарантированным сохранением стабильной работы. Основной принцип работы Docker - контейнеризация приложений. Этот тип виртуализации позволяет упаковывать программное обеспечение по изолированным средам — контейнерам. Каждый из этих виртуальных блоков содержит все нужные элементы для работы приложения. Это дает возможность одновременного запуска большого количества контейнеров на одном хосте [9].

Для разворачивания системы с помощью Docker прежде всего было необходимо подготовить образы созданных компонент. Для этого в каждом проекте был добавлен Dockerfile с конфигурацией процесса сборки образа. Образы backend и frontend частей собираются из заранее собранных проектов и необходимых им базовых образов.

Сборка образа коннектора к базе данных oracle является более трудоемкой. Образ построен на основе облегченной версии Oracle Linux седьмой версии. В образ копируются файлы самого коннектора, построенного на основе node js сервера. Далее производится установка всех необходимых зависимостей, включая Oracle Client, который предназначен непосредственно для подключения к базе данных. После завершения установки всех необходимых пакетов производится запуск коннектора.

По завершении сборки каждого образа они публикуются в репозиториях Docker Hub. Docker Hub – это публичный регистр образов, предназначенный для публикации, хранения и скачивания образов. Применение Docker Hub позволяет использовать созданные образы на любом устройстве с доступом к сети интернет.

Непосредственно для разворачивания приложения были использован Docker Compose. Docker Compose - это инструментальное средство, входящее в состав Docker. Оно используется для одновременного управления несколькими контейнерами, входящими в состав приложения. Этот инструмент предлагает те же возможности, что и Docker, но позволяет работать с более сложными приложениями [15].

Основная работа с Docker Compose заключается в создании конфигурационного yml-файла, в котором описаны правила запуска контейнеров. В нем описывается последовательность запуска контейнеров, порты запуска и прочие настройки.

Непосредственно разрабатываемая система включает в себя четыре контейнера: schedule-adapter, postgres, schedule-backend, schedule-frontend (см. приложение Т). Контейнер postgres представляет внутреннюю базу данных и основан на образе postgres:10-alpine, то есть десятой версии СУБД. Также были определены конфигурационные файлы, в которых содержатся переменные окружения. Данные файлы подключаются в разделе env\_file. Для каждого контейнера было задано правило restart: unless-stopped. Это означает, что при непредвиденной остановке контейнера он должен будет в автоматическом режиме совершить попытку перезапуска.

Сам yml файл и прочие файлы конфигурации были опубликованы в отдельном репозитории. Для запуска системы достаточно скачать содержимое репозитория и запустить все с использованием docker-compose. Сам запуск при этом условии производится всего одной командой. В то же время каждым запущенным контейнером можно управлять по отдельности.

## Описание функциональности системы

Итоговая функциональность системы полностью соответствует определенным в начале работы требованиям. Сценарии работы разделяются на два основных потока: работа от имени пользователя и работа от имени администратора.

Пользователь может производить поиск и просмотр интересующих его сущностей. Для этого он может воспользоваться web приложением, перейдя к нему из браузера; терминалами в кампусе университета, на которых развернута урезанная версия web-приложения, в то же время сохранившая основной функционал; мобильным приложением. Пользователь осуществляет поиск интересующей его сущности одним из предусмотренных в системе способов. Либо же в мобильном приложении он может перейти напрямую к одной из последних посещенных сущностей. Перейдя к расписанию сущности, пользователь может производить корректировку своего запроса, изменять отображение, используя элементы управления.

Что касается администратора, то для доступа в систему он должен быть зарегистрирован в ней. В системе всегда зарегистрирован как минимум один суперадминистратор, который может регистрировать других пользователей и удалять их из системы. Суперадминистратор выдает пользователям логины и пароли, под которыми они могут осуществлять вход. Для входа в панель администратора пользователь должен перейти по ссылке /admin основного домена. Ему откроется форма входа, в которую необходимо ввести данные. Если они верны, пользователь получает доступ к компонентам системы соответственно его роли. При получении доступа пользователь с максимальными правами может просматривать список всех сущностей и изменять их видимость, запускать обновление базы данных, управлять другими пользователями. Пользователь же с минимальными администраторскими правами может лишь изменять видимость сущностей.

## Тестирование системы

Тестирование системы проводилось в несколько этапов. Первый этап проводился непосредственно в процессе разработки системы. В первую очередь производилось тестирование работы серверной части, а именно соответствие данных, которые отправляет сервер ожидаемым моделям. Для отправки запросов был использован сервис Postman. Postman - это программа для тестирования работы различных API. Он предназначен для проверки запросов с клиента на сервер и получения ответа от сервера. В процессе тестирования на каждый реализуемый адрес отправлялись запросы с различными возможными входными параметрами. При этом отслеживалось общее поведение системы, ее реакция на ошибки, соответствие полученных данных ожидаемым. По результатам тестирования в работу системы вносились корректировки.

Также производилась проверка корректности обновления данных в базе данных. Было обнаружено, что в базе данных университета содержится большое количество неиспользуемых либо ошибочных сущностей. Соответственно была произведена их группировка и разработана функция фильтрации.

Второй этап тестирования системы производился по окончании работ, когда система была развернута на локальном сервере. В процессе данного этапа производилось ручное тестирование пользовательского интерфейса. Производилась проверка соответствия его дизайну, корректности его отображения на различных устройствах, тестировалась отзывчивость элементов управления.

Третий, заключительный, этап был самым продолжительным. В его рамках система была развернута на тестовой машине в сети университета. В первую очередь работа системы была проверена сотрудниками университета, которые отвечают за работу с расписанием. Они указали на недочеты системы и после их устранения была произведена публикация для использования студентами в тестовом режиме. В раках данного этапа производилась стабилизация работы системы, вносился новый функционал согласно пожеланиям пользователей.

**Выводы к главе 3**

В данной главе были рассмотрены современные методы и технологии построения прикладных программных продуктов. Были исследованы основные архитектурные принципы, применяемые при разработке программных систем, произведен их анализ, произведен выбор архитектуры, которая подходит для реализации системы, исследуемой в работе. Для реализации была выбрана многоуровневая архитектура, так как она позволяет разделять элементы системы на различные независимые компоненты, что значительно повышает ее понятность, потенциал для модернизации и расширения.

Далее были изучены современные системы управления базами данных, представленные на рынке программного обеспечения. Среди данных систем было необходимо выбрать подходящую для реализации проекта и разработать для нее физическую модель данных, для чего были выделены основные требования. Из них следовало, что система должна быть реляционной, иметь некоммерческую лицензию, стабильной, надежной, для нее должны выпускаться частые обновления. Среди всех вариантов выбор был остановлен на PostgreSQL.

По завершении всех подготовительных этапов была осуществлена непосредственно программная разработка компонентов системы. Для каждого из компонентов были выбраны наиболее оптимальные технологии реализации. При выборе был сделан упор на использование строгой типизации, легковесную основу, высокую производительность в сравнении с аналогами. При реализации разных компонентов были разработаны общие модели данных, благодаря чему сохранялось общее единообразие компонент.

По окончании разработки все компоненты были развернуты с использованием Docker. С его помощью была создана конфигурация, позволяющая запускать систему при помощи одной команды. По окончании работ было проведено трехэтапное тестирование.

# Заключение

В данной работе была произведена реализация модуля поиска и отображения расписания учебных занятий в Тольяттинском государственном университете.

В рамках работы над проектом был произведен анализ предметной области, определены основные проблемы существующих решений. На основе полученной информации, а также пожеланий пользователей и представителей администрации университета были выявлены требования к реализации и произведено проектирование различных аспектов реализуемой системы. Для проектирования была выбрана одна общая методология UML, а также в процессе по необходимости использовались дополнительные инструменты, такие как IDEF1X. При проектировании было изучено взаимодействие пользователя с системой, потоки и структуры данных. В результате была поставлена задача на реализацию web-приложения, приложения для работы с использованием стационарных терминалов и мобильного приложения.

В рамках процесса реализации была определена архитектура будущей системы. Выбор пал на многоуровневую архитектуру из-за удобного взаимодействия отдельных компонентов системы, а также потенциала дня расширения и модификации.

С использованием полученной информации был определен стек технологий для реализации программного обеспечения, который отвечал основным требованиям, архитектуре, требованиям к удобству разработки и поддержки. Далее с их использованием была произведена реализация отдельных компонентов системы.

По завершении реализации было произведено разворачивание и тестирование системы, по итогам которого была выпущена web версия приложения и версия для терминалов. Мобильная версия ожидает решения организационных вопросов и работает в тестовом режиме.

На данный момент система находится в рабочем состоянии, она эксплуатируется большим числом пользователей, что подтверждают различные встроенные метрики.

При всем вышеописанном система продолжает свое развитие. Готовится публикация мобильного приложения, постоянно вносится дополнительный функционал, производится исправление ошибок, оптимизация процессов. Следующим этапом развития должна стать переработанная версия с новым интерфейсом, с более продвинутым пользовательским опытом, а также значительными оптимизациями в работе сервера.

Подводя итоги, можно утверждать, что работа была проведена в полном объёме, все поставленные задачи выполнены. Система введена в эксплуатацию, активно используется и развивается.

# Список используемой литературы

1. Гради Буч. Введение в UML от создателей языка / Гради Буч, Джеймс Рамбо, Ивар Якобсон. - ДМК Пресс, 2015. – 496 с.
2. Репин В.В., Елиферов В.Г. Процессный подход к управлению. Моделирование бизнес-процессов / Владимир Репин, Виталий Елиферов. - М.: Манн, Иванов и Фербер, 2016. - С. 80 – 108.
3. Figma — простое решение для дизайнера, сложное решение для верстальщика / [Электронный ресурс] URL: https://habr.com/ru/post/463181
4. Oracle PL/SQL учебник / [Электронный ресурс]. URL: https://oracleplsql.ru/contents-oracle-plsql.html, (дата обращения: 03.04.2021).
5. Архитектура JVM / [Электронный ресурс]. URL: https://coderlessons.com/articles/java/arkhitektura-jvm-obzor-arkhitektury-jvm-i-jvm, (дата обращения: 23.03.2021).
6. Архитектура REST / [Электронный ресурс] URL: https://habr.com/ru/post/38730/, (дата обращения: 29.03.2021).
7. Василий Дичак. Варианты кроссплатформенной разработки мобильных приложений / [Электронный ресурс] URL: https://dou.ua/lenta/ articles/cross-platform-mobile-development, (дата обращения: 15.04.2021).
8. Не Spring Boot’ом единым: обзор альтернатив / [Электронный ресурс] URL: https://habr.com/ru/company/raiffeisenbank/blog/456376/, (дата обращения: 25.04.2021).
9. Что такое Docker и как его использовать в разработке / [Электронный ресурс] URL: https://eternalhost.net/blog/razrabotka/chto-takoe-docker, (дата обращения: 03.05.2021).
10. Alon Brody. SQL Vs NoSQL: The Differences Explained / [Электронный ресурс] URL: https://blog.panoply.io/sql-or-nosql-that-is-the-question, (дата обращения: 02.04.2021).
11. BPMN Specification - Business Process Model and Notation / [Электронный ресурс] URL: https://www.bpmn.org/, (дата обращения: 06.05.2021).
12. Dirk Strauss, Jas Rademeyer. C# 7 and .NET Core 2.0 Blueprints. - Packt Publishing Ltd, 2018
13. Flavio Copes. The definitive Node.js handbook / [Электронный ресурс] URL: https://www.freecodecamp.org/news/the-definitive-node-js-handbook-6912378afc6e/, (дата обращения: 24.03.2021).
14. Gaël Thomas. A beginner’s guide to Docker - how to create a client/server side with docker-compose / [Электронный ресурс] URL: https://www.freecodecamp.org/news/a-beginners-guide-to-docker-how-to-create-a-client-server-side-with-docker-compose-12c8cf0ae0aa/, (дата обращения: 04.05.2021).
15. Ktor official documentation / [Электронный ресурс] URL: https://ktor.io/docs, (дата обращения: 05.04.2021).
16. Mark Tielens Thomas. React in Action. - Manning Publications, 2019
17. Mohit Malhotra. Everything About Software Architecture / [Электронный ресурс] URL: https://medium.com/ swlh/everything-aboutsoftware-architecture-dfd2b9351ef4, (дата обращения: 27.03.2021).
18. 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: 03.04.2021).

# Приложение К

**Общая диаграмма классов**

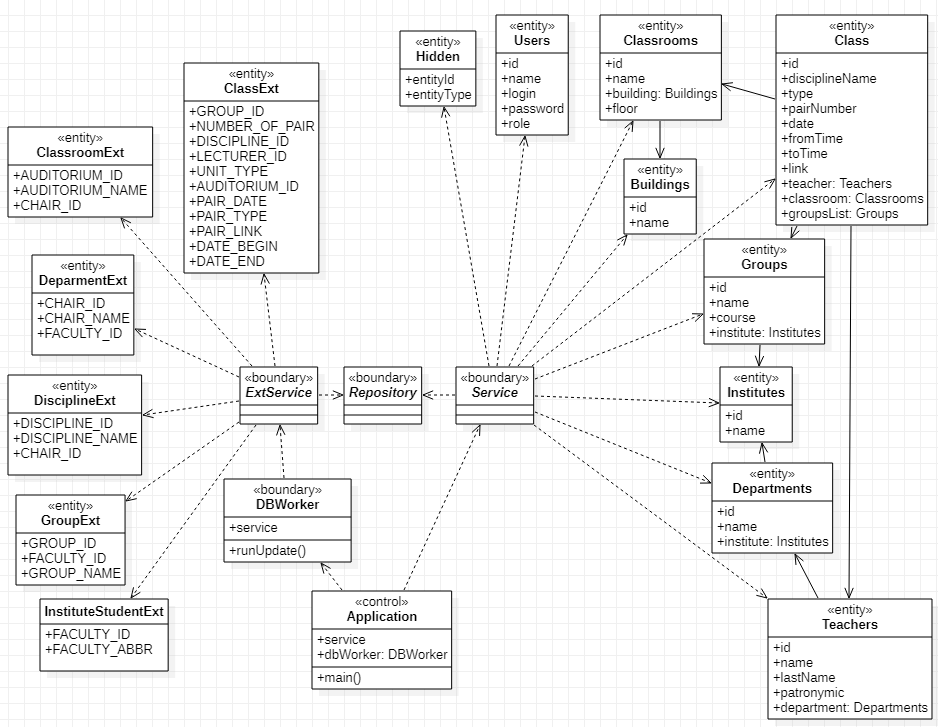


Рисунок К.1 – Абстрактная диаграмма классов

# Приложение Л

**Диаграмма последовательности для прецедента «Поиск расписания»**

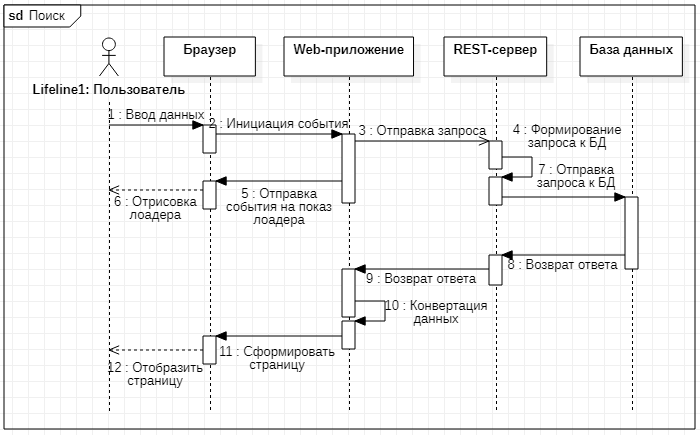


Рисунок Л.1 – Диаграмма последовательности для прецедента «Поиск расписания»

# Приложение М

**Диаграмма последовательности для прецедента «Вход в систему»**

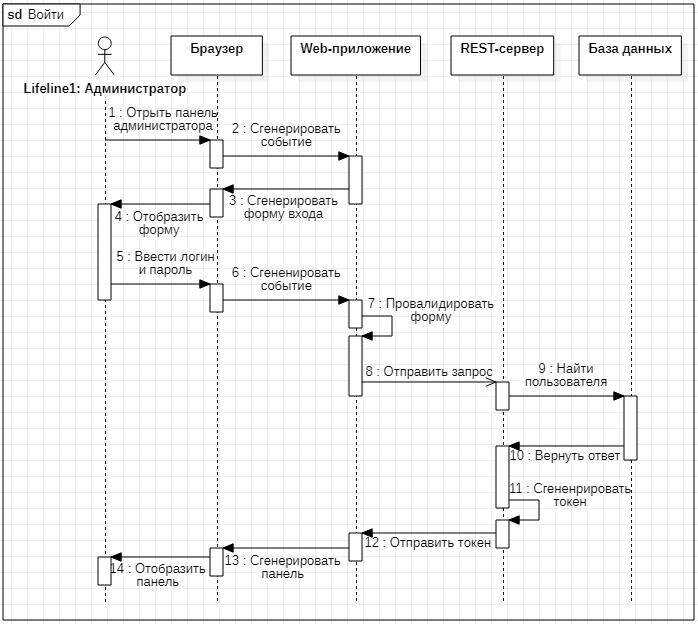


Рисунок М.1 – Диаграмма последовательности для прецедента «Вход в систему»

# Приложение Н

**Метод получения кафедр по идентификатору института**

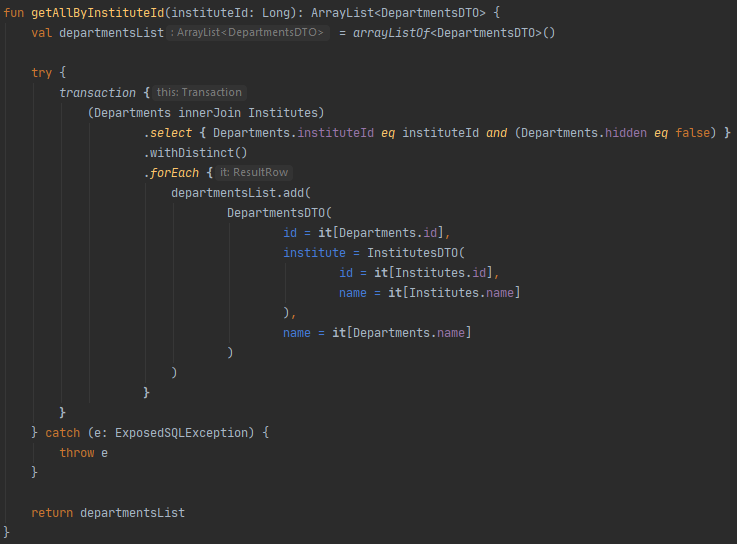


Рисунок Н.1 – Программный код метода получения кафедр по идентификатору института

# Приложение П

**Процент поисковых запросов на форуме StackOverflow для различных кроссплатформенных инструментов от общего числа запросов**

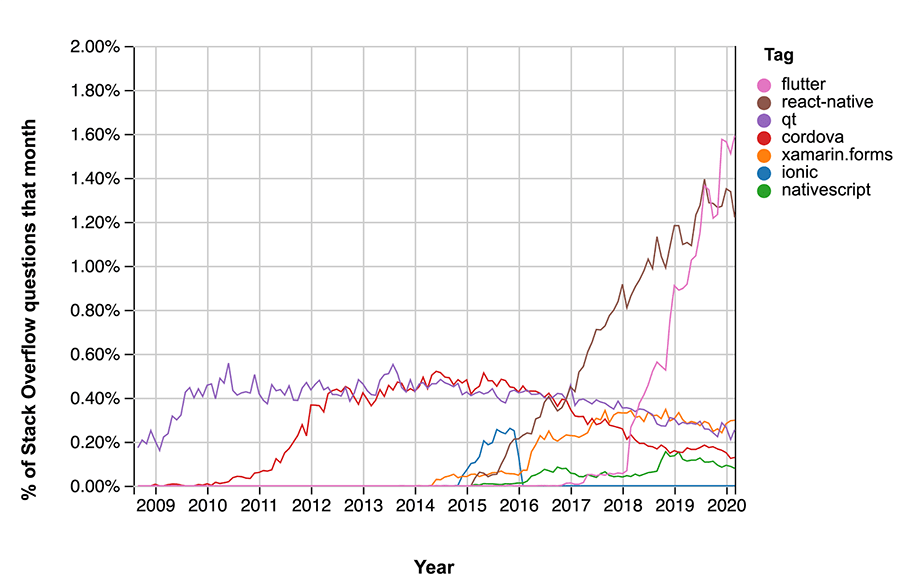


Рисунок П.1 – График статистики поисковых запросов

# Приложение Р

**Использование FutureBuilder**



Рисунок Р.1 – Программный код с примером использования FutureBuilder

# Приложение С

**Диаграмма развертывания системы**

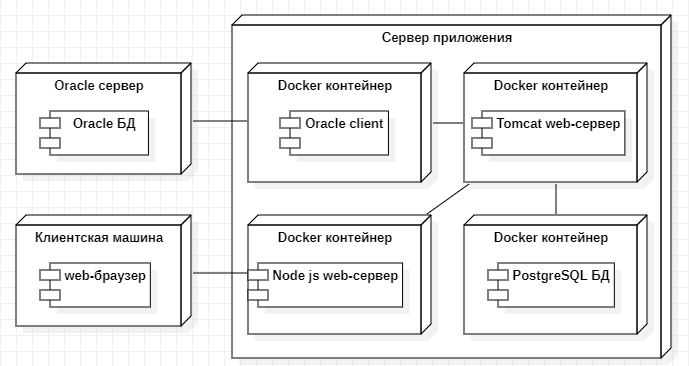


Рисунок С.1 – Диаграмма развертывания системы

# Приложение Т

**Docker Compose конфигурация**

version: '3'  
services:  
 schedule-adapter:  
 image: naiu0819/schedule-adapter:latest  
 container\_name: schedule-adapter  
 restart: unless-stopped  
 networks:  
 - internal  
 - external  
 ports:  
 - "8000:80"  
 env\_file:  
 - ./env/adapter.env  
  
 postgres:  
 image: postgres:10-alpine  
 container\_name: postgres  
 restart: unless-stopped  
 ports:  
 - '5432:5432'  
 networks:  
 - internal  
 - external  
 env\_file:  
 - ./env/postgres.env  
  
 schedule-backend:  
 image: naiu0819/schedule-backend:latest  
 container\_name: schedule-backend  
 restart: unless-stopped  
 depends\_on:  
 - schedule-adapter  
 - postgres  
 networks:  
 - internal  
 - external  
 ports:  
 - "8080:8080"  
 env\_file:  
 - ./env/backend.env  
  
 schedule-frontend:  
 image: naiu0819/schedule-frontend:latest  
 container\_name: schedule-frontend  
 restart: unless-stopped  
 depends\_on:  
 - schedule-backend  
 networks:  
 - internal  
 - external  
 ports:  
 - "3000:3000"  
  
networks:  
 internal:  
 driver: bridge  
 internal: true  
 external:  
 driver: bridge