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)

09.03.03 «Прикладная информатика»
(код и наименование направления подготовки, специальности)

Корпоративные информационные системы
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка программного модуля для поиска и просмотра расписания учебных занятий в ТГУ»

Студент Н.А. Юсупов
(И.О. Фамилия) (личная подпись)

Руководитель А.П. Тонких
(ученая степень, звание, И.О. Фамилия)

Консультант А.В. Москалюк
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

Аннотация

С. 56, рис. 18, табл. 1, лит. 20 источников

РАСПИСАНИЕ УЧЕБНЫХ ЗАНЯТИЙ, ПОИСКОВАЯ СИСТЕМА, UML, IDEF1X, REST-СЕРВЕР, WEB-ПРИЛОЖЕНИЕ, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ.

Разработана система поиска и отображения расписания учебных занятий для Тольяттинского государственного университета.

Выполнен анализ процесса работы с данными о студентах, преподавателях и расписании их учебных занятий в университете, выявлены основные проблемы процесса поиска и просмотра расписания. Построены модели «как есть» с использованием различных видов UML-диаграмм. Сформулированы цели и задачи разработки и требования к реализации системы.

Выполнено проектирование процессов «как должно быть», разработана логическая и физическая модель. Произведен выбор архитектуры программной реализации системы, выполнен анализ инструментов, с использованием которых должна производиться разработка, и на его основе произведен отбор. С использованием выбранных инструментов были реализованы программные компоненты системы, настроено ее разворачивание.

Выполнено тестирование готовых программных продуктов, произведен сбор информации о проблемах, произведено устранение дефектов.

Работа находится на стадии активного использования, производится доработка компонент системы.

Abstract

This graduation work is about development of a software module for searching and displaying class schedules for Togliatti State University.

The author dwells on the topic of software design and development.

The graduation work consists of an explanatory note on 56 pages, introduction, including 18 figures, 1 table, the list of 20 references including 11 foreign sources.

The aim of this work is research of modern approaches to software development and their implementation for existing information systems.

The object of the graduation work is a university information management system.

The subject of the graduation work is implementation of classes schedule data obtaining, converting and storing.

The graduation work is divided into three logically connected parts: analysis, design and implementation.

The work begins with the analysis of the existing university information system. In the course of the analysis, the main problems are identified and ways to solve them are proposed. Also processes are documented by UML diagrams. Finally, in the first part of the graduation work the main basic requirements for the new system are documented.

The second part is devoted to logical analysis. It includes analysis of the data presented in university data base and data flow analysis.

The final part of the graduation work contains description of physical implementation of the system. The created program is based on multilevel architecture. It contains web-server, web-application and cross platform mobile application. The system is deployed on the university server using Docker and tested in several stages.

The result of the work is finished software module for searching and displaying class schedules for Togliatti State University. This service is successfully

integrated into the university information system. For now, it is actively used by a students and university employees.

For future releases, user interface redesign and final release of the mobile application are planned.

Оглавление

Введение

В выпускной квалификационной работе будет рассмотрен процесс разработки модуля поиска и отображения расписания учебных занятия Тольяттинского государственного университета.

Актуальность работы обусловлена отсутствием полноценных систем данного типа. На момент начала работ в инфраструктуре университета для решения задачи поиска п просмотра расписания использовались программные продукты ограниченной функциональности, которые в основном являлись лишь модулями других более крупных систем. В то же время у студентов университета существовала потребность в современном продукте, который был бы удобен для использования.

Объектом исследования является система управления данными о работе университета.

Предметом исследования является реализация получения, преобразования, хранения данных о расписании учебных занятий, а также реализация системы поиска и отображения этих данных в удобном для пользователей формате.

Целью работы является исследование современных подходов к реализации современных программных систем, а также их применение для реализации системы поиска и отображения расписания учебных занятий в университете.

Для достижения поставленных целей были выделены следующие задачи:

- выявление требований к системе,
- выбор технологий проектирования,
- разработка концептуальной модели системы,
- разработка логической модели системы,
- выбор технологий программной реализации,

– программная реализация системы,
– тестирование системы.

Результатом работы является список реализованных программных продуктов, составляющих единую систему. К ним относится серверная часть, реализующая основную логику системы, web-приложение и мобильное приложение.

В первой главе работы было проведено исследование предметной области, определены основные требования к разрабатываемой системе. На основе полученных данных была разработана функциональная модель различных аспектов системы, произведено моделирование основных прецедентов. В завершении были поставлены задачи на разработку.

Во второй главе было произведено моделирование потоков данных, взаимодействие отдельных компонент системы, моделирование отдельных сущностей данных. На основе полученной информации была разработана логическая модель данных.

В третьей главе было проведено изучение современных архитектурных решений для реализации прикладных программных продуктов, произведен выбор технологий организации хранения данных и реализации программного обеспечения. На основе выбранной системы управления базой данных была разработана физическая модель данных. По завершении всех приготовлений была непосредственно произведена программная реализация компонент системы на основе выбранной архитектуры. По завершении реализации было произведено разворачивание и тестирование готового продукта.

Глава 1 Функциональное моделирование модуля поиска и отображения расписания учебных занятий

1.1 Описание процесса работы с расписанием

На текущий момент работа с расписанием в Тольяттинском государственном университете включена в общую систему управления предприятием. Управление осуществляется с использованием системы «Галактика», в которой производится работа с большей частью организационных процессов, таких как управление данными о преподавателях, студентах, учет объектов университета и т.д. В процессе работы вся информация сохраняется в базах данных и впоследствии может быть использована как непосредственно внутри системы, так и в каких-либо внешних компонентах. При этом оба типа компонентов могут иметь как узкую направленность и использоваться для решения какой-то конкретной задачи, так и сочетать в себе перечень функций. Примером узконаправленных систем могут служить различные боты, предоставляющие студентам информацию о расписании, преподавателях и т.д. Примером же системы широкого направления является образовательный портал, который позволяет решать различные задачи.

Модули, позволяющие просматривать расписание занятий в университете, непосредственно являются составляющими систем широкого направления. Все они строятся примерно на одних принципах: берется набор таблиц из основной базы данных, относящихся непосредственно к занятиям, проводимым в университете, на сервере производится обработка данных в необходимом формате, данные передаются клиенту по запросу. При этом данные могут быть выданы пользователю персонально, например, в случае его аутентификации в подсистеме, либо могут выдаваться в зависимости от заданных параметров анонимному пользователю.

1.2 Определение требований к разрабатываемой системе

Для выявления требований к программному модулю были проанализированы пожелания представителей университета, проведен анализ существующей системы, произведено ее моделирование.

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

– отображение в приложении расписания занятия для групп, преподавателей и аудиторий,

– наличие web-версии приложения,

– наличие мобильной версии приложения,

– наличие версии для стационарных терминалов,

– наличие функционала для скрытия или отображения сущностей, представленных в базе данных системы «Галактика».

На основе общих требований было 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 [1]. 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 [**Ошибка! Источник ссылки не найден.**].

При использовании представленных инструментов используются разные подходы к моделированию и предназначены они для решения разных задач. Соответственно, чтобы определиться с выбором был определен круг задач, решаемых при моделировании.

В рамках выполнения работы был заранее известен модуль, который было необходимо разработать, соответственно основной упор должен был производиться именно на его проектирование и реализацию, в то время как бизнес-процессы и предметная область в целом не требуют детального анализа. Из этого можно сделать вывод, что наиболее подходящим инструментом для решения данной задачи является UML из-за его направленности на разработку программного обеспечения, универсальности и

полноты. С использованием выбранной методологии и было произведено дальнейшее моделирование системы.

На момент начала разработки проекта официальная система отображения расписания учебных занятий ТГУ базировалась на использовании таблиц формата xls. Данные таблицы формируются с использованием системы «Галактика» и публикуются на официальном сайте университета. То есть для просмотра расписания пользователю необходимо найти нужную таблицу на сайте, скачать ее и найти интересующий раздел. Процесс поиска отображен на диаграмме видов деятельности (см. приложение А). По ней можно понять, что процесс поиска требует большого количества действий от пользователя. При этом каждое действие в реальной системе сопровождается продолжительными загрузками. Пользователю необходимо производить поиск и группировку информации вручную. При этом если необходимо изменить параметры поиска, то необходимо проходить весь цикл заново. Помимо этого, подобный функционал слабо оптимизирован для мобильных устройств из-за необходимости приложения для чтения xls файлов. Дальнейшее моделирование процессов «Как есть» не представляется возможным, так как существующая система достаточно примитивна. Соответственно разрабатываемая система должна быть значительно шире и охватывать большее количество аспектов.

Также еще одним существенным недостатком системы является отсутствие возможности редактирования сущностей, так как компонент работы с расписанием является частью общей системы и изменение сущностей для нее приведет к изменениям во всех остальных компонентах, что является нежелательным.

На основе полученной информации можно сформировать окончательные требования к разрабатываемой системе. Требования к системе определяются по классификации требований FURPS+, то есть делятся по группам: функциональность, удобство, надежность, производительность, поддерживаемость и проектные ограничения.

Функционально систему можно разделить на 3 модуля: модуль обработки данных, модуль выдачи данных и модуль редактирования данных. Модуль обработки данных отвечает за работу с данными, поступающими из базы данных ТГУ. Данные должны преобразовываться в формат, удобный для работы системы и последующей выдачи. При этом подобные действия производятся циклично для поддержания актуального состояния данных. Модуль выдачи данных включает как интерфейсную, так и функциональную часть и предоставляет пользователю инструменты для поиска и просмотра интересующей его информации в удобном формате. При этом запросы должны быть гибкими, то есть пользователь может в любой момент скорректировать критерии поиска и получить обновленную информацию. Модуль редактирования данных включает в себя функционал по администрированию системы: скрытие и отображение сущностей: групп, преподавателей, аудиторий и т.д. Также он включает в себя функционал по работе с базой данных: запуск ручного обновления, очистка. Помимо этого, должен быть предусмотрен функционал управления пользователями-администраторами.

Модуль выдачи данных должен давать доступ на просмотр информации любому анонимному пользователю. Модуль редактирования данных должен иметь аутентификацию и авторизацию с доступом по ролям. Должно присутствовать как минимум две роли: администратор, управляющий сущностями и супер-администратор, имеющий доступ к работе с базой данных и к управлению доступом для других пользователей с более низкими по иерархии ролями. Это должно обеспечить защиту от несанкционированного доступа к функционалу системы, что должно увеличить общую надежность.

Также для обеспечения большей надежности система должна иметь возможность какое-то время работать автономно в случае потери соединения с серверами ТГУ. То есть необходимо предусмотреть копирование информации, чтобы система могла продолжить свою работу до восстановления соединения. Помимо этого, в случае падения системы она

должна в автоматическом режиме совершить перезапуск вышедших из строя модулей или, если такая возможность отсутствует, должен быть реализован удобный инструмент для быстрого перезапуска в ручном режиме.

Что касается удобства использования, то приложение должно иметь понятный и современный интерфейс, что подразумевает удобное расположение управляющих элементов, смысловое выделение различных элементов интерфейса. Должно быть достаточное количество элементов управления для работы с системой, но в то же время функционал не должен быть перегружен. Оно должно поддерживаться разными платформами.

Интерфейс web-приложения должен поддерживаться во всех популярных браузерах с актуальной поддержкой. Версия для терминалов должна корректно отображаться в браузере Mozilla Firefox последних версий, а также не иметь ссылок на внешние ресурсы. Мобильное приложение должно корректно работать на смартфонах с версией операционной системы Android 6 и выше, а также на смартфонах с версией iOS 7 и выше. Серверная часть должна исполняться на операционных системах семейства UNIX, а именно Ubuntu Server версии 16.04 и выше, а также CentOS версии 7 и выше.

Поддерживаемость должна быть обеспечена возможностью быстрого развертывания модулей. Помимо этого, необходимо реализовать модульную систему, чтобы можно было в любой момент добавить или заменить какой-либо модуль. Также в разработке самих модулей стоит использовать компонентный подход и минимизировать связность компонент.

Дополнительных требований к архитектуре приложений, используемым технологиям не предъявлялось.

1.3 Разбор аналогов системы

Официальным аналогом от ТГУ является уже упомянутая система, использующая документы xls. В целом принцип ее работы и недостатки были

рассмотрены ранее. Среди студентов данная система пользуется наименьшей популярностью из-за большого времени, которое тратится на поиск нужной информации.

Помимо этого, в качестве официального сервиса можно отметить расписание, которое отображается на образовательном портале университета. Оно находится в одном из разделов личного кабинета пользователя. Соответственно для просмотра требуется аутентификация. Основным недостатком данной системы является невозможность задания критериев поиска. Расписание отображается только для текущего пользователя.

На данный момент наиболее популярным среди студентов аналогом является бот для социальной сети «Вконтакте» и мессенджера «Телеграмм» «МИСА». Данный бот разработан сотрудниками ЦНИТ. По запросу он предоставляет информацию о расписании, а также различную информацию из образовательного портала. Данный бот обладает обширным функционалом, предоставляет различную справочную информацию, для чего реализована аутентификация через образовательный портал. Основным преимуществом является быстрый доступ к информации в процессе нахождения в социальной сети. Однако недостатками, как и у любого тестового бота, являются ограничение для формата информации, привязка к родительскому приложению. В целом данный бот и разрабатываемое приложение являются дополнением друг к другу.

Также стоит упомянуть web-приложение «Бинарус», разработанное студентом ИМФиИТ, идейным продолжателем которого является проект, разрабатываемый в рамках работы. Данное приложение впервые было размещено на терминалах в кампусе университета и пользовалось большой популярностью, однако в определенный момент перестало поддерживаться.

Таким образом, на текущий момент не существует полноценной системы для работы с расписанием, которая бы удовлетворяла всем выявленным требованиям и полностью бы покрывала потребности пользователей.

1.4 Постановка задачи на разработку

В рамках работы было необходимо реализовать справочную систему, которая бы позволяла просматривать расписание учебных занятий ТГУ с любых устройств, включая терминалы, установленные в зданиях университета.

Общие функциональные возможности системы:

- поиск групп, преподавателей, аудиторий по поисковому запросу;
- поиск групп, преподавателей, аудиторий по объектам, к котором они привязаны: кафедры, институты, корпусы и т.д.;
- просмотр расписания учебных занятия на неделю;
- просмотр расписания учебных занятий на день;
- корректировка критериев поиска;
- регулярное обновление информации о занятиях.

Специфичный функционал для администратора:

- просмотр списка скрытых сущностей;
- поиск сущностей
- управление видимостью сущностей.

Специфичный функционал для супер-администратора:

- добавление/удаление пользователей;
- запуск обновления базы данных;

Необходимо реализовать адаптивное web-приложение, версию для терминала на его базе, мобильное приложение, панель администратора и реализовать серверную часть для обеспечения всех этих компонентов данными.

Соответственно в первую очередь необходимо проанализировать функционал будущей системы, взаимодействие пользователей с ней, произвести моделирование системы и процессов, протекающих в ней.

Произвести анализ связей с системой университета. Изучить входные данные, с которыми система будет работать.

Также необходимо построить модель данных, определить связи, основные сущности. Определиться с типом, структурой и физической реализацией системы хранения данных.

В конечном итоге необходимо определиться с архитектурой системы, взаимодействием компонент. Стоит изучить технологии, применяемые для подобных проектов, выбрать наиболее подходящие для реализации и непосредственно перейти к этапу разработки. В процессе разработки вносить необходимые корректировки, промежуточное тестирование.

По завершении разработки необходимо определиться с моделью развертывания на тестовом и продуктовом сервере, развернуть систему на тестовом окружении и произвести полноценное тестирование.

По итогам тестирования необходимо внести необходимые правки, создать окончательный план развертывания и развернуть систему на продуктовом окружении. В процессе работы производить мониторинг системы и в случае необходимости вносить корректировки.

1.5 Разработка модели «Как должно быть»

В первую очередь были выделены области ответственности для разных пользователей. На диаграмме прецедентов (см. приложение Б) представлены основные действующие лица и их основные действия. При этом стоит отметить, что супер-администратор наследует доступ ко всем функциям администратора и также имеет дополнительный доступ. Пользователь же никак не пересекается с управляющей частью.

Далее для большего понимания взаимодействия пользователя с системой для процессов, отображенных на диаграмме деятельности, были составлены диаграммы видов деятельности.

Первым делом была разработана диаграмма деятельности для прецедента «Поиск группы, преподавателя, аудитории» (см. приложение В). На диаграмме видно, что в сравнении с предыдущим вариантом было уменьшено количество шагов. Помимо этого, при просмотре расписания была добавлена возможность редактирования параметров, что избавляет пользователя от необходимости прохождения всего потока заново. При этом модель прецедента поиска была объединена с моделью поиска прецедента для просмотра расписания.

В целом, процесс взаимодействия пользователя с системой является достаточно тривиальным, поэтому далее было проведено моделирование работы администраторов.

Для начала было произведено моделирование процесса «Вход в систему» (см. приложение Г). Данный процесс является общим для всех прецедентов, в которых принимают участие администраторы. Процесс инициализируется пользователем, который переходит к панели администратора. Система делает проверку на факт того, является ли пользователь аутентифицированным. Если является, то пользователь сразу получает доступ к разделам системы, согласно правам, предусмотренным его ролью. В противном случае отображается форма входа. Пользователь вводит данные в форму и отправляет ее. В случае если пользователь с такими данными существует, аутентификация считается успешной, и пользователь получает доступ к разделам панели.

Далее было произведено моделирование процесса управления сущностями базы данных (см приложение Д). В рамках этого процесса пользователь входит в систему согласно логике, описанной ранее, и переходит в раздел управления сущностями. В отображенном окне ему необходимо найти сущность, к которой нужно применить настройки видимости. После нахождения искомой сущности выбирается опция и подтверждается действие. Далее система запускает логику отображения или скрытия сущности.

Следующим было произведено моделирование процессов для супер-администратора. Первым делом была разработана диаграмма для процесса работы с базой данных (см. приложение Е). Данный процесс также инициализируется входом в систему. Затем пользователь переходит в нужный раздел и выбирает опцию обновления. Система производит запуск алгоритма для выбранной опции и уведомляет пользователя о завершении работы.

Последними оставшимися прецедентами являются процессы, связанные с управлением пользователями, а именно добавление и удаление. Для инициации обоих процессов пользователю необходимо войти в систему, выбрать нужный раздел. Для добавления нового администратора необходимо отрыть форму, ввести данные о пользователе и инициализировать процесс оправки формы (см. приложение Ж). Система сохранит данные о новом пользователе в случае отсутствия в базе данных идентичной записи.

Для удаления записи необходимо перейти к списку существующих пользователей, выбрать нужную запись и инициализировать удаление. Система производит поиск записи и удаляет ее из базы данных (см. приложение И).

Для каждого из прецедентов должно быть предусмотрено отображение сообщений о возникающих в процессе работы ошибках для более продуктивного взаимодействия с пользователем.

Выводы к главе 1

В рамках данного этапа было произведено функциональное моделирование предметной области для изучения текущего состояния системы, выявления ее проблем и способов их устранения. Было выявлено, что текущая система не содержит полноценных компонентов для поиска информации о расписании учебных занятий университета. Соответственно основной задачей была поставлена разработка подобного функционала с нуля.

Для моделирования текущей и разрабатываемой систем была выбрана методология UML, так как она является универсальной, заточена под разработку программного обеспечения, что является одним из основных критериев, так как в рамках работы производилась разработка программного продукта с нуля, а не на базе существующей системы.

С использованием выбранной методологии было произведено моделирование процесса поиска информации в существующей системе. Было выявлено, что процесс состоит из большого количества шагов, не является адаптируемым к запросам пользователей, использует устаревшие технологии, не обладает поддержкой большинства устройств.

На основе полученной информации были выявлены требования к разрабатываемой системе, среди которых стоит отметить интуитивно понятный процесс поиска искомой информации, современный понятный интерфейс, поддержка различных устройств.

Также был произведен анализ аналогов системы, используемых в университете. Было выявлено, что, несмотря на наличие подобных, они являются либо нишевыми, и поставляются в виде дополнений к другим системам, либо же не обладают всей полнотой функционала. В то же время в процессе анализа аналогов было получено дополнительное представление о возможных вариантах реализации.

На основе анализа системы были получены данные для постановки задачи. Были выявлены функциональные требования для разных пользователей, функционал на реализацию.

На основе полученной информации для всей системы были смоделированы прецеденты взаимодействия пользователей с ней, которые легли в основу дальнейшего проектирования и реализации.

Глава 2 Логическое проектирование модуля поиска и отображения расписания учебных занятий

2.1 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.

2.2 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.

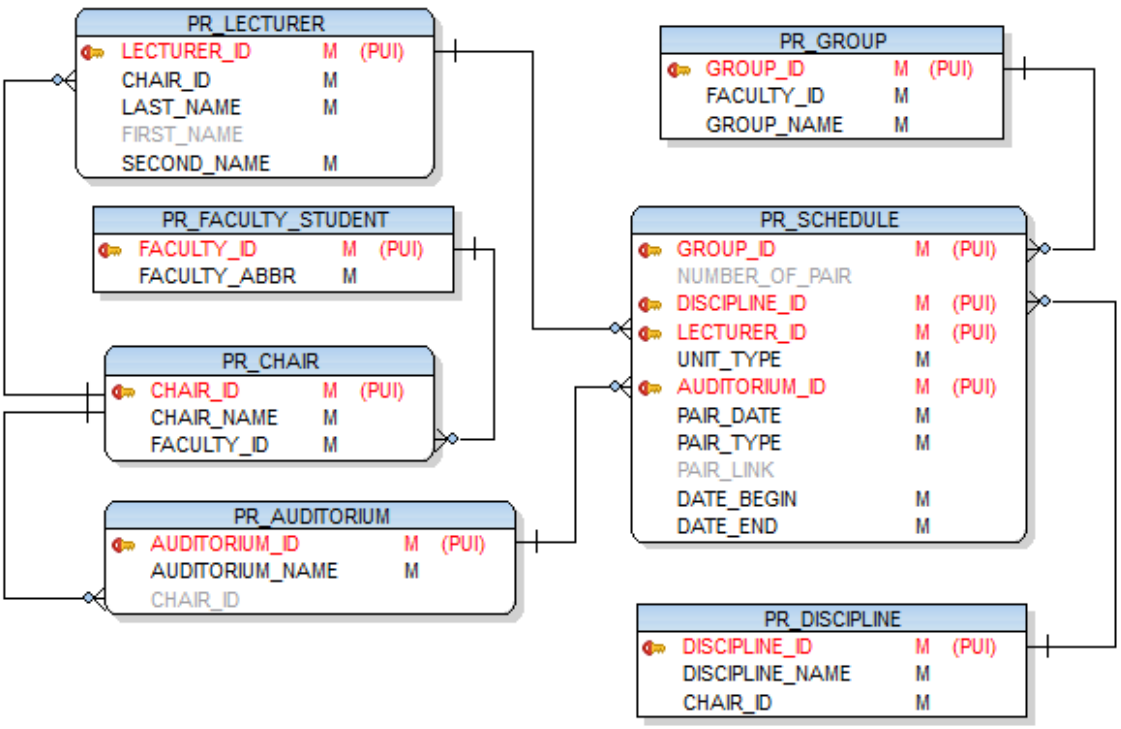

Рисунок 1 - Модель базы данных ТГУ

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.

2.3 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.

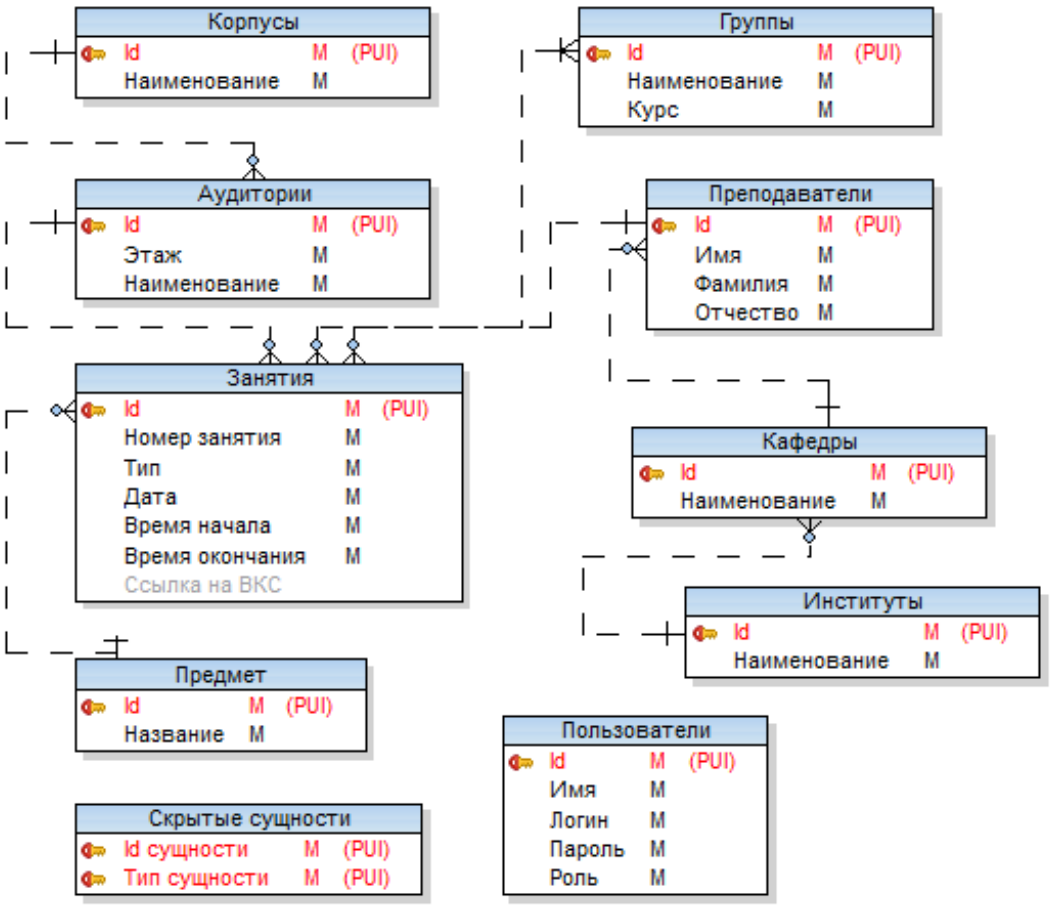

Рисунок 2 - Логическая модель данных

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.

Глава 3 Функциональное проектирование модуля поиска и отображения расписания учебных занятий

3.1 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.

3.2 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.

3.3 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.

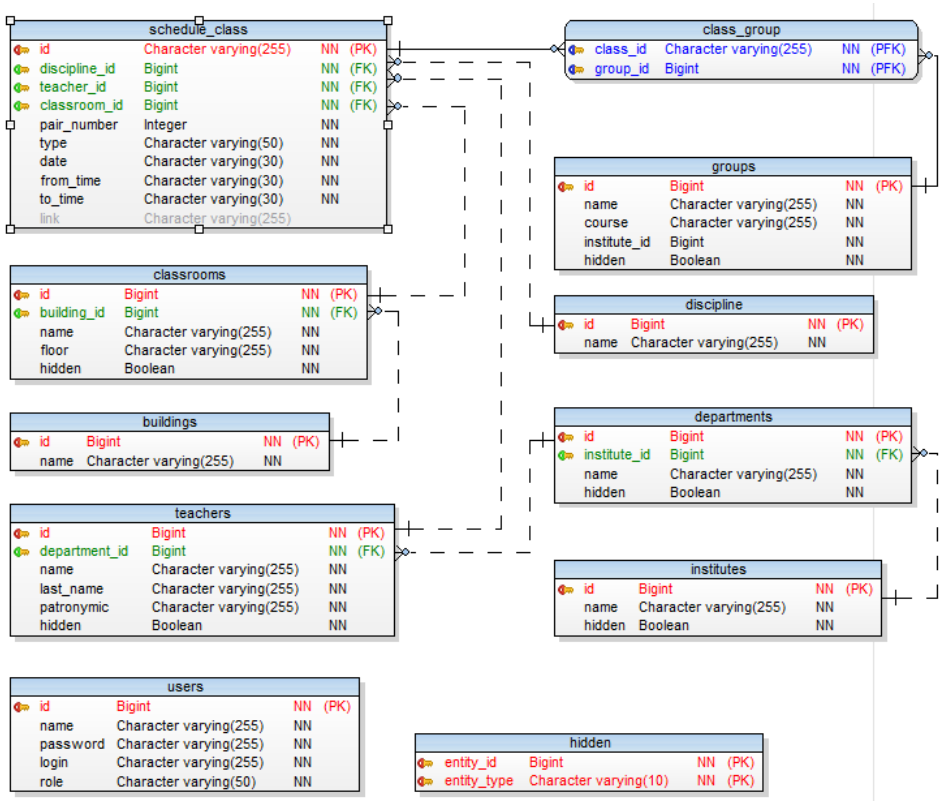

Рисунок 3 - Физическая модель данных

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.

3.4 Реализация серверной части системы

3.4.1 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].

3.4.2 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.

3.4.3 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.

3.5 Реализация web-приложения

3.5.1 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.

3.5.2 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.

3.5.1 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.

3.6 Реализация мобильного приложения

3.6.1 Выбор инструментов для реализации

На данный момент существует три основных подхода к реализации мобильных приложений: реализация нативных приложений отдельно для двух платформ, использование кроссплатформенных инструментов, оборачивание web-приложения в WebView.

При реализации нативных приложений используются платформозависимые языки программирования, а именно Java/Kotlin для Android и Objective-C/Swift для iOS. Главным преимуществом реализации нативных приложений является быстродействие. В них происходит взаимодействие непосредственно с самой операционной системой. Также функционал для реализации ограничен только самой платформой. Основным же недостатком подобного подхода является трудоемкость и дороговизна разработки. Необходимо разрабатывать и поддерживать одновременно два приложения. К тому же для реализации приложений для операционной системы iOS необходимо дорогостоящее оборудование в постоянном доступе.

Использование кроссплатформенных инструментов подразумевает создание одного приложения, которое способно исполняться на разных платформах. Для этого обычно используется программная прослойка между приложением и операционной системой. Преимуществами данного подходя являются скорость и дешевизна разработки. Однако подобные приложения имеют меньшую скорость работы в связи с тем, что обращение идет не напрямую к ОС.

Оборачивание web-приложения в WebView является простейшим способом создания мобильного приложения при условии, что уже реализовано web приложение. Однако данный способ накладывает ряд ограничений. Функционал web- и мобильного приложения будет идентичен, недоступны многие функции, которые можно реализовать непосредственно для мобильной платформы. Помимо всего прочего данные приложения отличаются очень низкой скоростью работы.

В рамках работы над реализацией мобильного приложения было решено использовать компромиссный вариант с использованием кроссплатформенных инструментов. Они обладают производительностью лучше, чем WebView, однако требуют меньших затрат в сравнении с нативными приложениями.

На данный момент существует большое количество кроссплатформенных инструментов разработки: Flutter, React Native, QT, Cordova, Xamarin, Ionic, NativeScript. В большинстве случаев они имеют схожую архитектуру: существует сам инструмент/платформа, а также прослойка, благодаря которой идет обращение к компонентам операционной системы. Большинство подобных инструментов предполагают разработку на JavaScript, так как он является самым простым и в то же время продвинутым языком для реализации пользовательских интерфейсов. В то же время существуют инструменты, применяющие другие подходы. По состоянию на 2020 год согласно статистике поисковых запросов, на форуме StackOverflow лидирующие позиции занимают два инструмента с принципиально разными подходами: Flutter и React Native (см. приложение П).

React Native - это фреймворк от Facebook для разработки кроссплатформенных приложений. В React Native не используются HTML и CSS, но есть некоторые компоненты платформы в JSX и CSS-подобные полифилы. В React Native каждый элемент является представлением нативного элемента, то есть является элементом более высокого уровня абстракции. Это позволяет добиться уровня представления нативных

мобильных приложений. В то же время подобный подход приводит к значительным задержкам в процессе работы приложения. Данный инструмент применяется в случаях, когда конечная производительность не так важна, как скорость разработки, а также если в компании делается ставка на React JS как на основной фреймворк.

Flutter - SDK от компании Google с открытым исходным кодом для создания кроссплатформенных мобильных приложений, который предоставляет пользователям как Android, так и iOS по-настоящему нативный дизайн и опыт. Особенностью архитектуры Flutter является то, что он сам рисует каждый пиксель, контролирует жесты и анимацию. Он не использует стандартные виджеты, как это делает React Native. Вместо этого были разработаны два набора виджетов для основных мобильных платформ: Material для Android и Cupertino для iOS. Непосредственно с мобильной платформой (геолокация, звук, Bluetooth) взаимодействие происходит через шину, называемую Platform Channels [7]. Данный инструмент является отличным решением, если необходимо разработать кроссплатформенное приложение с производительностью, близкой к нативным приложениям.

По завершении изучения современных инструментов кроссплатформенной разработки выбор для реализации приложения был остановлен на Flutter, как на самом эффективном инструменте такого типа.

3.6.1 Определение структуры проекта

По большей части структура проекта достаточно похожа на ту, что была применена для web-приложения. Однако были введены определенные изменения, связанные с особенностями разработки для мобильных устройств.

Во-первых, проект содержит дополнительные директории с конфигурациями для Android и iOS платформ. В данных директориях находятся платформозависимые модули, а также различные конфигурационные файлы.

Еще одним отличием является введение сервисов. Сервисы – модули, предназначенные для обработки данных, используемых приложением. При этом были выделены две группы сервисов: online и offline. Соответственно online-сервисы работают непосредственно с серверным API. Offline-сервисы работают с файловой системой устройства.

Корневым элементом является файл, содержащий функцию main, в которой производится запуск корневого класса приложения.

3.6.1 Программная реализация компонентов приложения

Весь процесс разработки приложения строился по тому же принципу, что и при разработке web-приложения, то есть сначала создавался View, потом он заполнялся компонентами. Каждый View представляет StatelessWidget, то есть не содержит в себе никакой бизнес логики, а отвечает только за представление и компоновку. При этом также учитывались типы View, к которым они относятся: группы, преподаватели, аудитории [20].

После инициализации View-компонентов была реализована маршрутизация, для навигации в приложении и отображения этих компонент. В Flutter существует два основных вида маршрутизации: прямая и именованная. Прямая маршрутизация предполагает переход непосредственно к компонентам. То есть в ссылках указываются сами компоненты. Именованная же предполагает задание компонентам псевдонимов и обращение непосредственно по ним. В проекте было решено использовать второй тип маршрутизации, так как это позволяет в случае необходимости легко заменить компоненту без изменения ссылок на нее.

Далее было необходимо реализовать получение данных от сервера. Первым делом были определены основные модели, которые используются в приложении. По большей части они соответствуют тем, что были определены на серверной части. При этом, так как обмен данными между сервером и клиентом производится в формате JSON, было необходимо для каждой модели

предусмотреть возможность конвертации. В базовом виде JSON-формат представляет словарь данных с ключами строкового типа, соответственно для его преобразования достаточно выделить значения, соответствующие нужным ключам.

Непосредственно получение данных производилось при помощи запросов с использованием библиотеки http. Функционал данной библиотеки аналогичен axios, который использовался для реализации web-приложения. При этом Dart поддерживает асинхронность, что позволяет не блокировать основной поток при выполнении запроса и отображать лоадер.

Отрисовка данных, получаемых от сервера, была реализована с помощью так называемых FutureBuilder. FutureBuilder – это компоненты, которые получают на вход асинхронный объект и отслеживают его состояние. Они позволяют отображать компонент по умолчанию на время выполнения асинхронной функции и перехватывать событие выполнения, после чего отображать нужный компонент. В приложении Р представлен пример использования FutureBuilder. На нем можно увидеть, что внутри компонента производится проверка на содержание данных и на наличие ошибки. При этом если отсутствуют и данные и ошибка, это значит, что функция находится в процессе выполнения. При изменении состояния функции производится повторная проверка.

По завершении отрисовки компонент, работающих с данными, приходящими от сервера, оставалось реализовать работу с файловой системой. В приложении на главной странице присутствует компонент со списком кнопок для перехода к последним посещенным элементам. Логика его работы следующая:

– пользователь переходит к расписанию для группы, аудитории или преподавателя;

– система получает список последних элементов, сохраненных в файловой системе;

– производится поиск искомого элемента, и в случае его отсутствия он добавляется в начало списка, в противном случае его позиция в списке изменяется на начальную;

– при следующем заходе в программа читает список из файловой системы и отображает элементы в том порядке, в котором они находятся в списке.

Для реализации этой логики был создан класс FileService. Данный класс содержит конструктор, в который передается имя файла. Поиск искомого файла ведется в директории приложения. Также класс содержит методы для чтения и записи данных в файл. Все операции производятся асинхронно. В остальном вся работа с файловой системой производится тем же образом, что и с сервером.

В завершении реализации приложения были изменены конфигурации для последующей сборки. Например, были добавлены иконки рабочего стола, добавлены списки разрешений, необходимых приложению, а именно доступ к интернету и файловой системе.

3.7 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.

3.8 Описание функциональности системы поиска и отображения расписания учебных занятий

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.

3.9 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. Alessio Stalla. Guide to the Kotlin Exposed Framework / [Электронный ресурс] URL: https://www.baeldung.com/kotlin/exposed-persistence, (дата обращения: 09.04.2021).
11. 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).

12. BPMN Specification - Business Process Model and Notation / [Электронный ресурс] URL: https://www.bpmn.org/, (дата обращения: 06.05.2021).

13. Dirk Strauss, Jas Rademeyer. C# 7 and .NET Core 2.0 Blueprints. - Packt Publishing Ltd, 2018

14. 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).

15. 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).

16. Ktor official documentation / [Электронный ресурс] URL: https://ktor.io/docs, (дата обращения: 05.04.2021).

17. Mark Tielens Thomas. React in Action. - Manning Publications, 2019

18. Mohit Malhotra. Everything About Software Architecture / [Электронный ресурс] URL: https://medium.com/ swlh/everything-aboutsoftware-architecture-dfd2b9351ef4, (дата обращения: 27.03.2021).

19. 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).

20. Rap Payne. Beginning App Development with Flutter: Create Cross-Platform Mobile Apps / Rap Payne - Apress 2019.

Приложение А

Диаграмма видов деятельности для прецедента «Поиск расписания» («Как есть»)

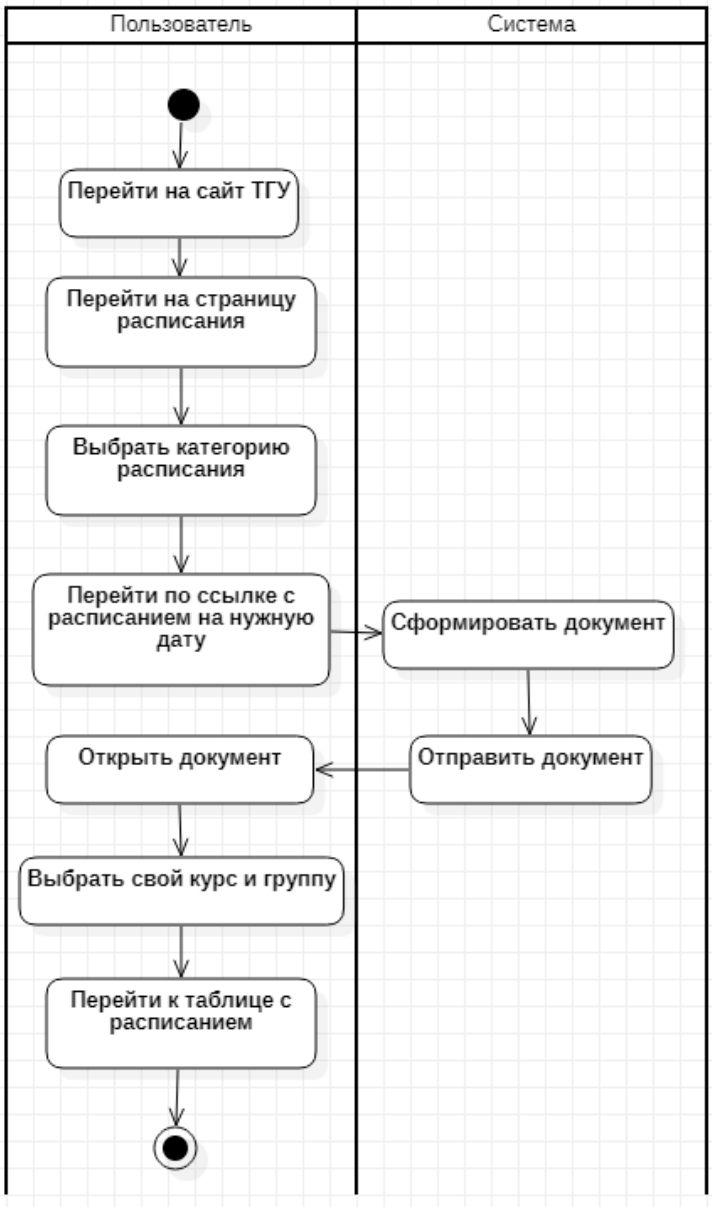

Рисунок А.1 - Диаграмма видов деятельности для прецедента «Поиск расписания» («Как есть»)

Приложение Б

Диаграмма прецедентов

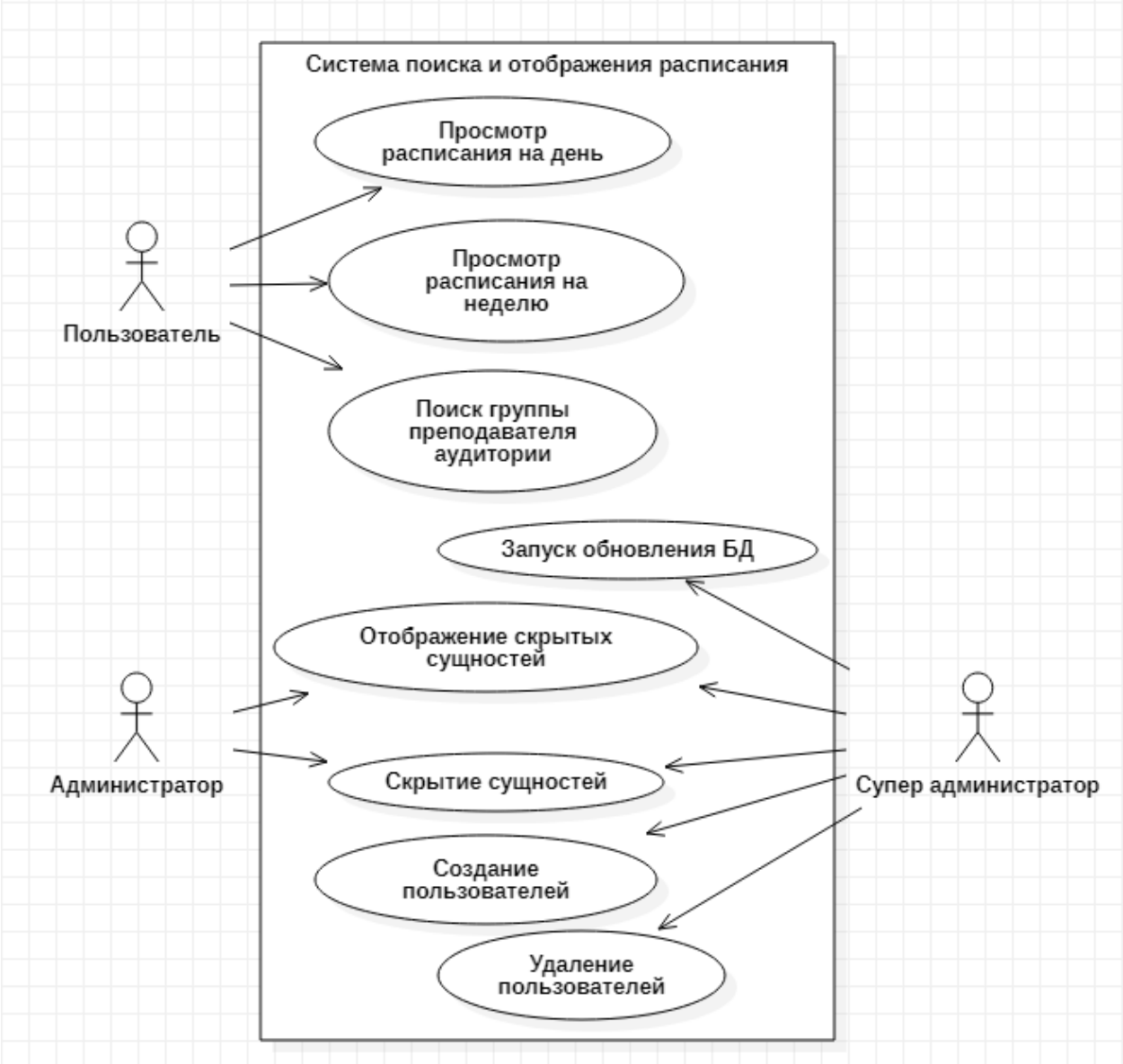

Рисунок Б.1 - Диаграмма прецедентов взаимодействия пользователей

Приложение В

Диаграмма видов деятельности для прецедента «Поиск расписания» («Как должно быть»)

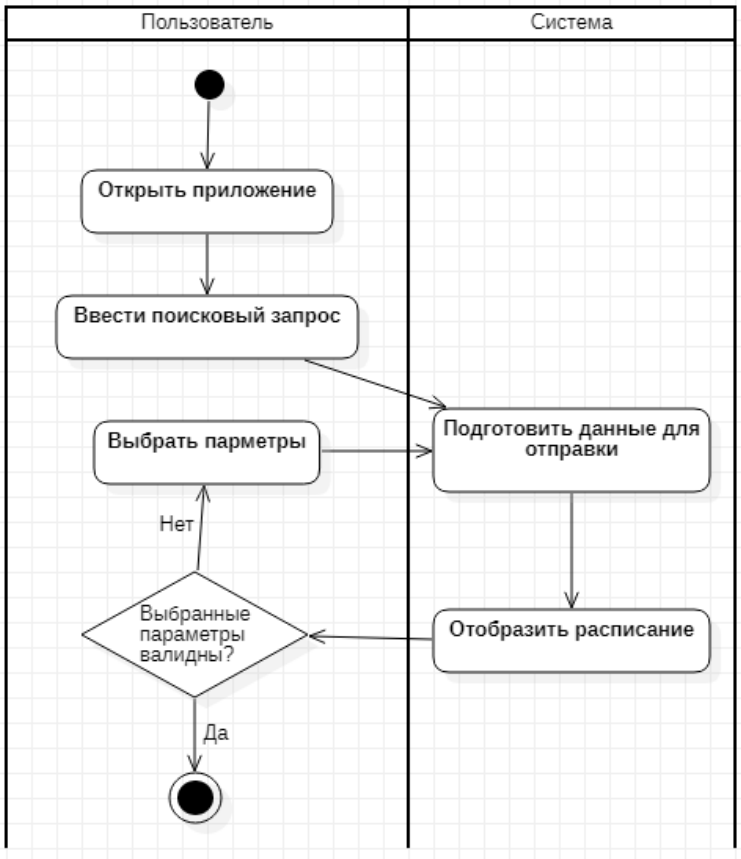

Рисунок В.1 - Диаграмма видов деятельности для прецедента «Поиск расписания» («Как должно быть»)

Приложение Г

Диаграмма видов деятельности прецедента «Вход в систему»

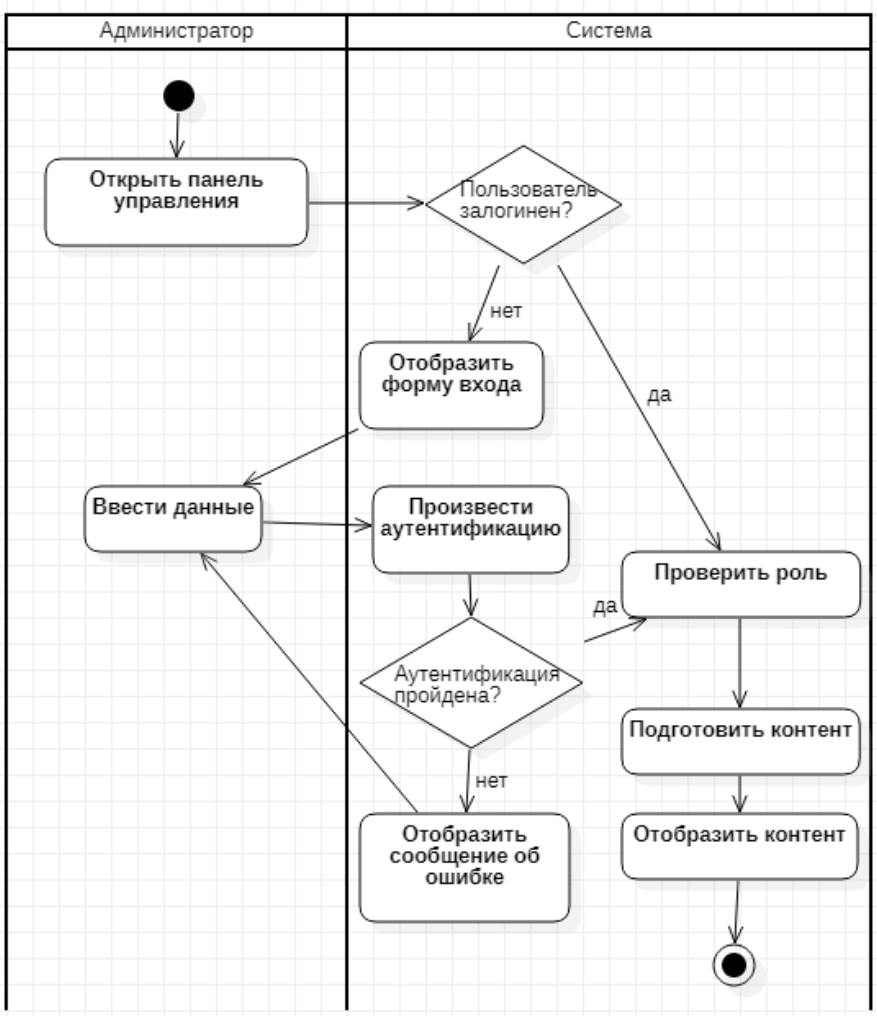

Рисунок Г.1 - Диаграмма видов деятельности прецедента «Вход в систему»

Приложение Д

Диаграмма видов деятельности для процесса «Управление сущностями»

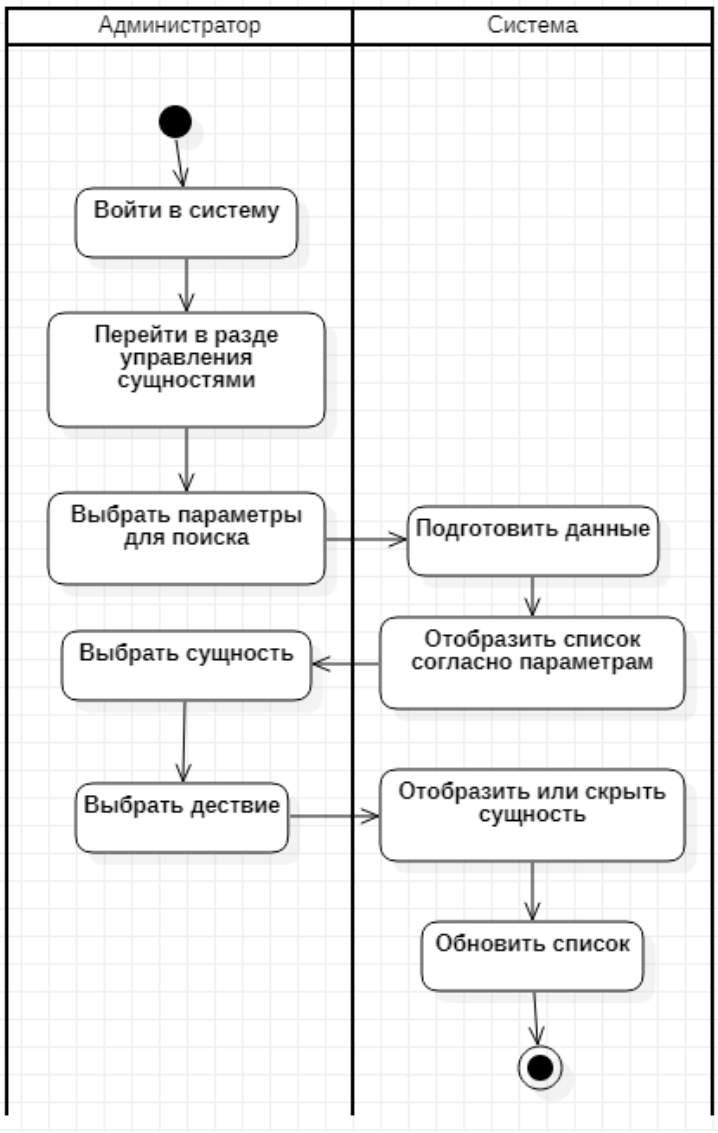

Рисунок Д.1 - Диаграмма видов деятельности для процесса «Управление сущностями»

Приложение Е

Диаграмма последовательности для прецедента «Обновление БД»

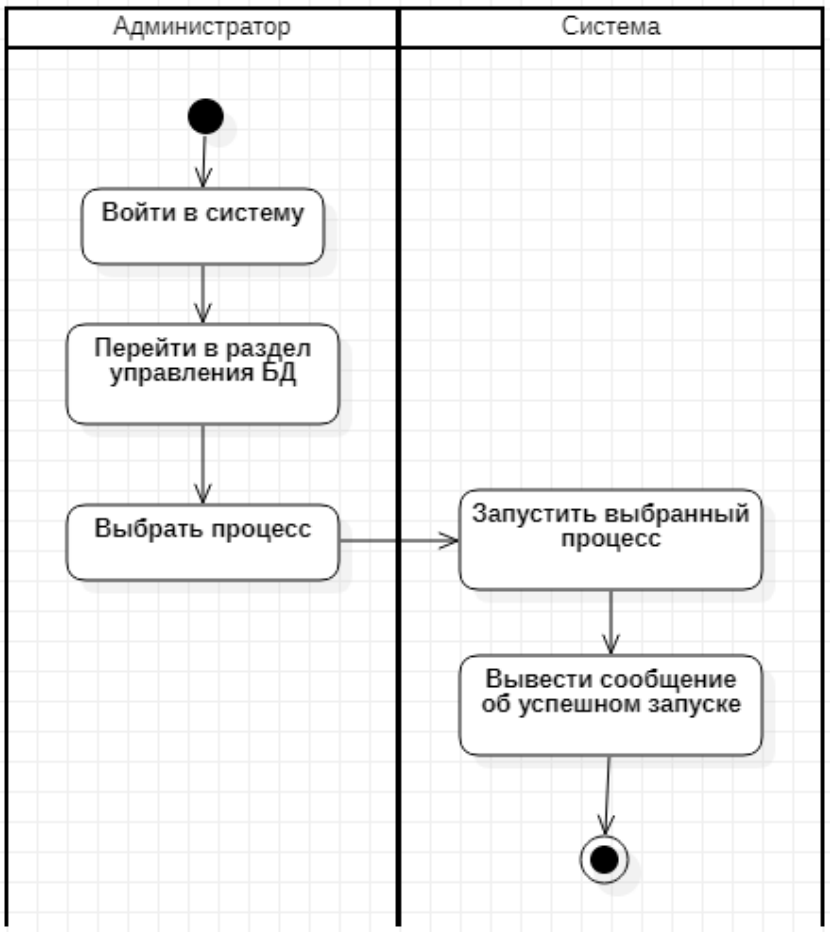

Рисунок Е.1 - Диаграмма последовательности для прецедента «Обновление базы данных»

Приложение Ж

Диаграмма последовательности для прецедента «Добавление пользователя»

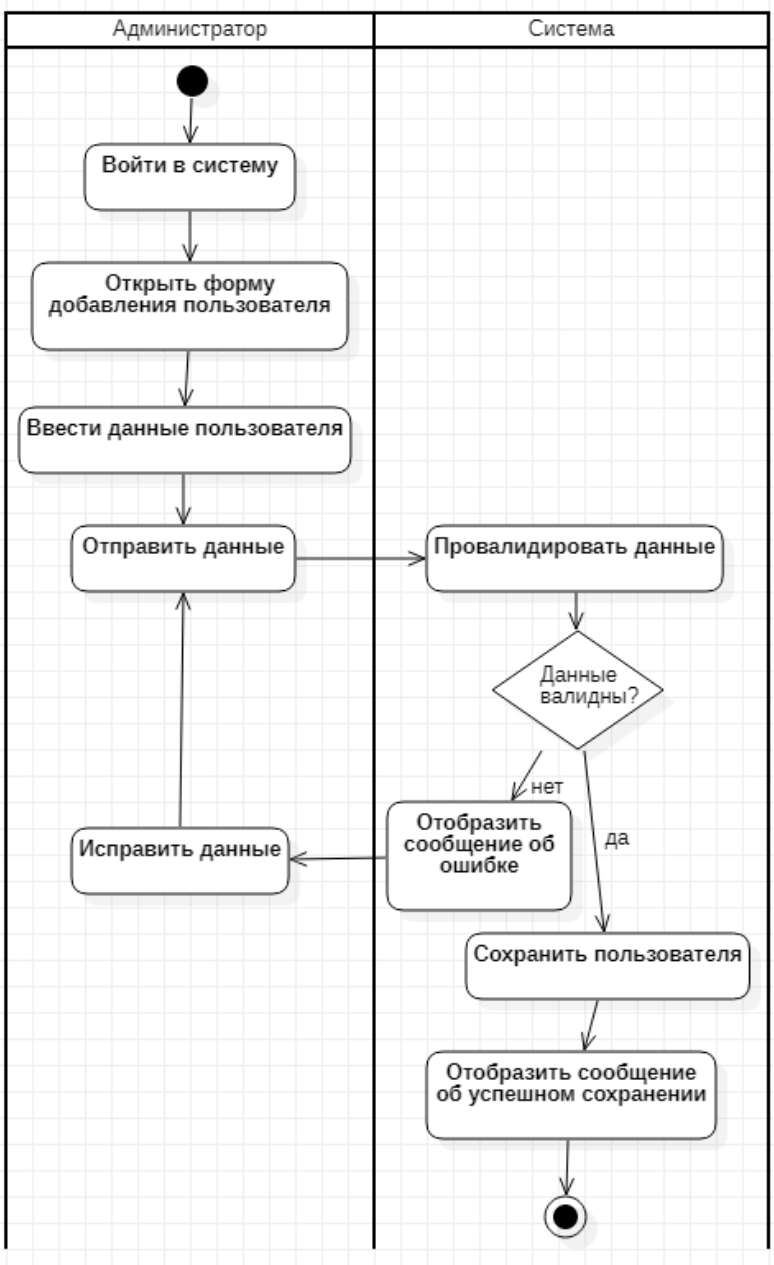

Рисунок Ж.1 - Диаграмма последовательности для прецедента «Добавление пользователя»

Приложение И

Диаграмма последовательности для прецедента «Удаление пользователя»

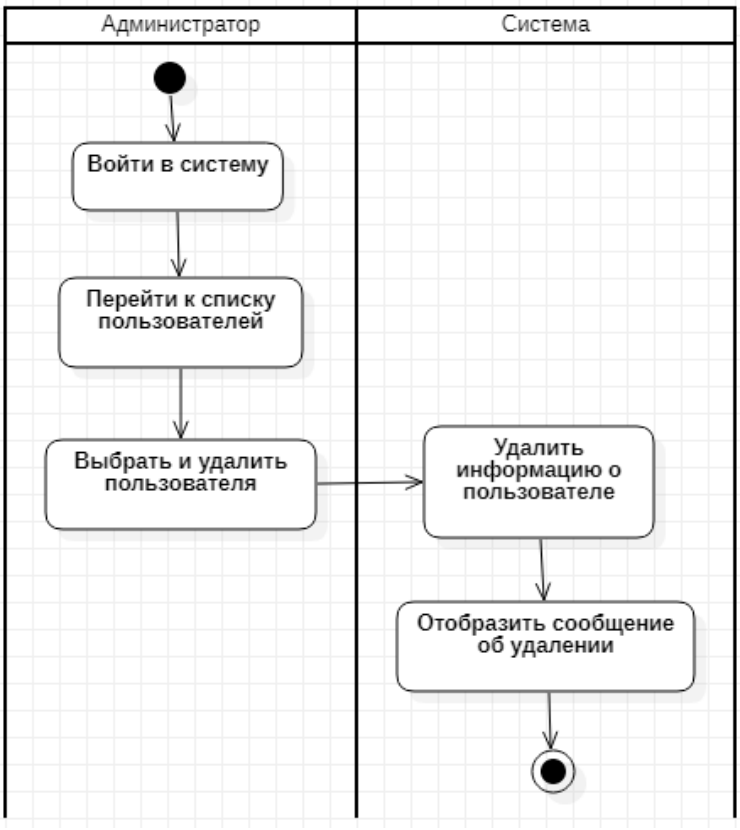

Рисунок И.1 - Диаграмма последовательности для прецедента «Удаление пользователя»

Приложение К

Общая диаграмма классов

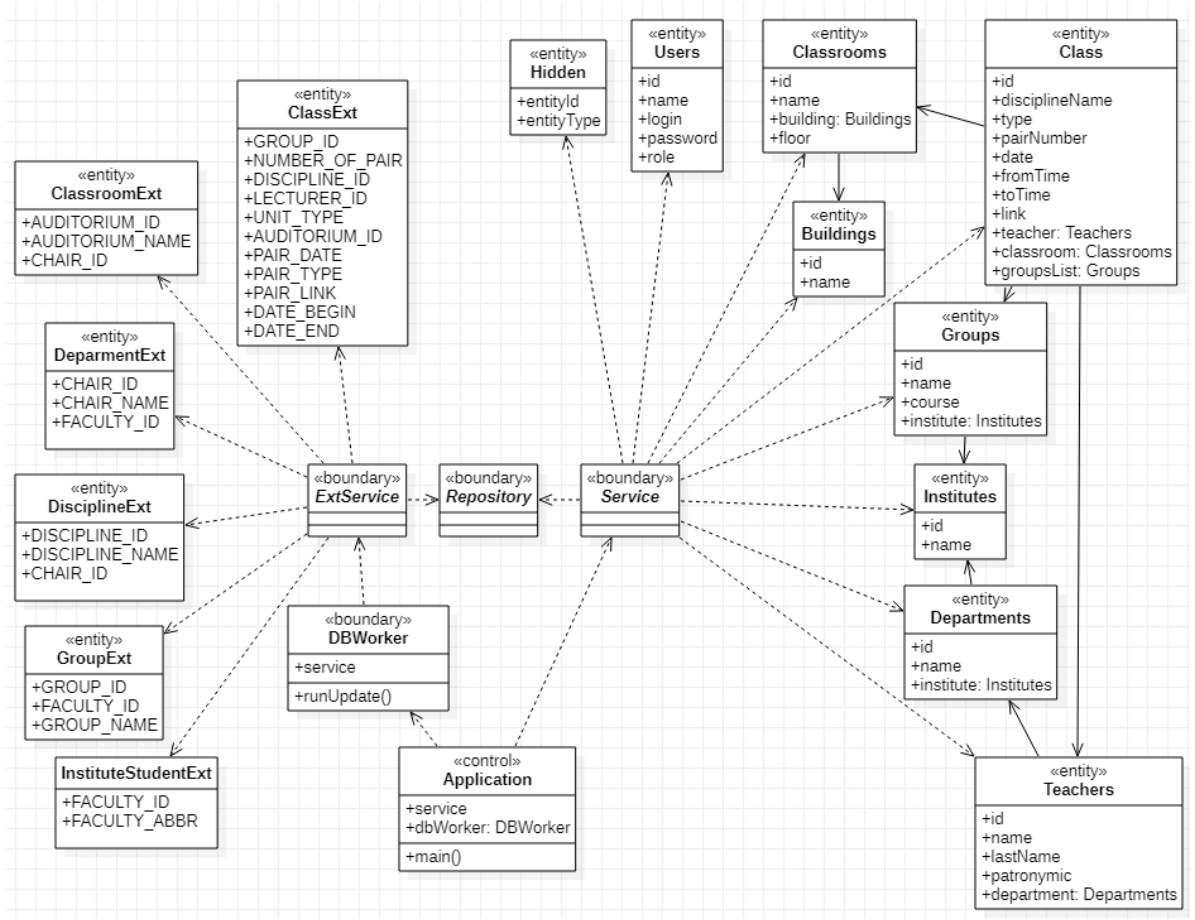

Рисунок К.1 – Абстрактная диаграмма классов

Приложение Л

Диаграмма последовательности для прецедента «Поиск расписания»

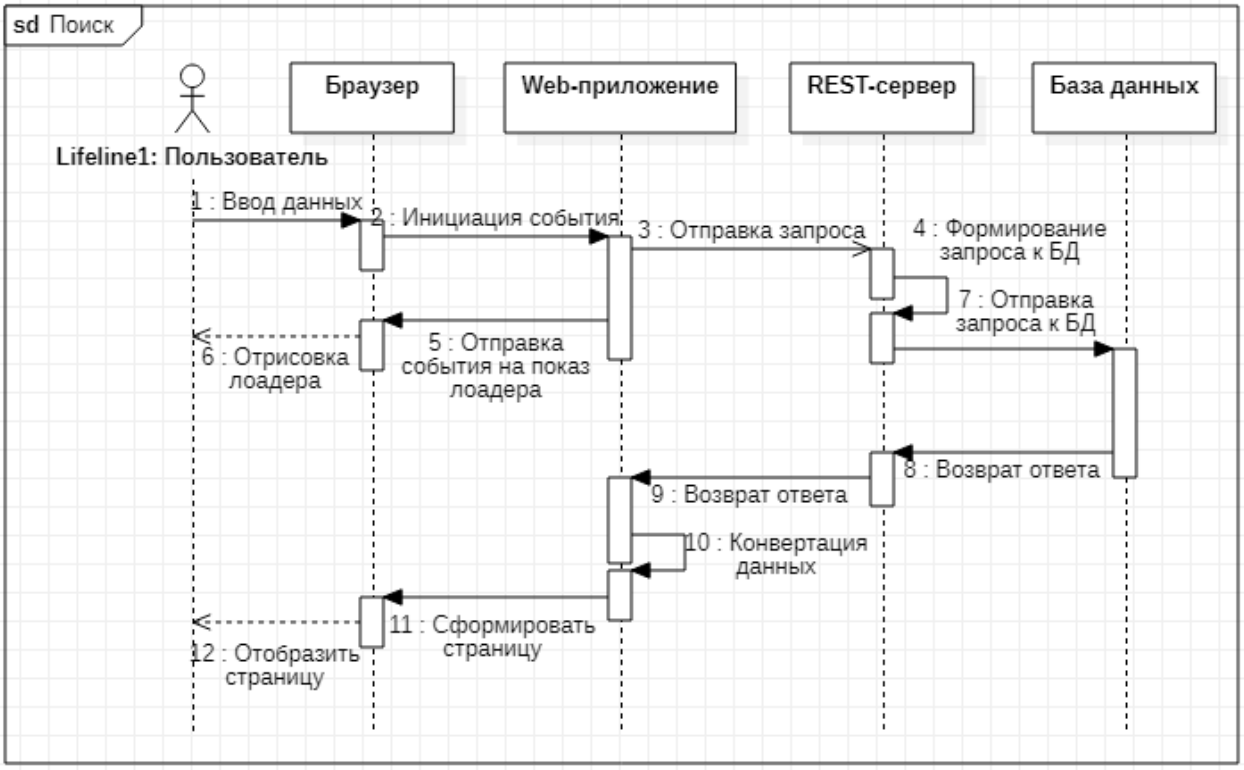

Рисунок Л.1 – Диаграмма последовательности для прецедента «Поиск расписания»

Приложение М

Диаграмма последовательности для прецедента «Вход в систему»

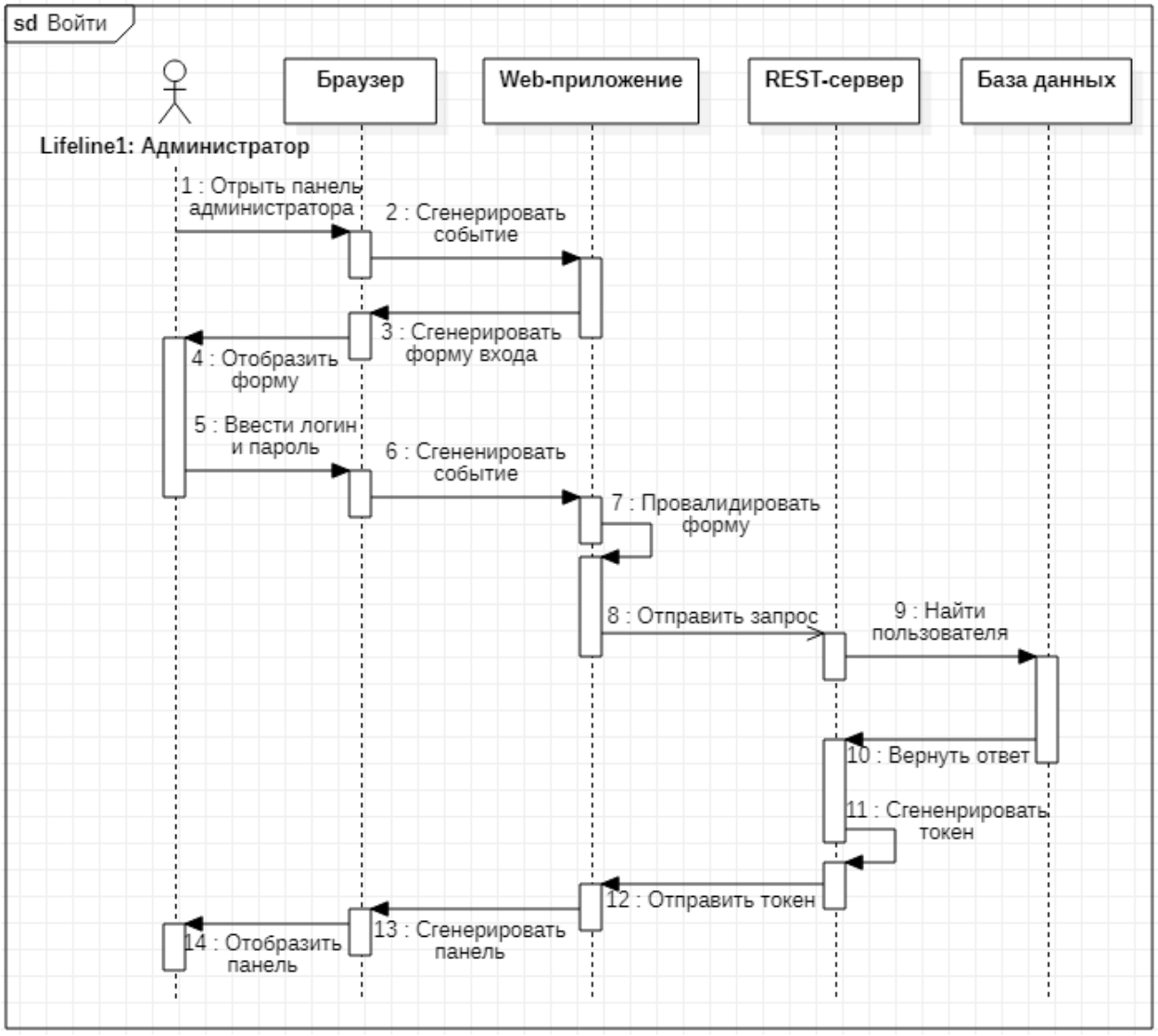

Рисунок М.1 – Диаграмма последовательности для прецедента «Вход в систему»

Приложение Н

Метод получения кафедр по идентификатору института

```
fun getAllByInstituteId(instituteId: Long): ArrayList<DepartmentsDTO> {
    val departmentsList : ArrayList<DepartmentsDTO>  = arrayListOf<DepartmentsDTO>()

    try {
        transaction { this: Transaction
            (Departments innerJoin Institutes)
                    .select { Departments.instituteId eq instituteId and (Departments.hidden eq false) }
                    .withDistinct()
                    .forEach { it: ResultRow
                        departmentsList.add(
                                DepartmentsDTO(
                                        id = it[Departments.id],
                                        institute = InstitutesDTO(
                                                id = it[Institutes.id],
                                                name = it[Institutes.name]
                                        ),
                                        name = it[Departments.name]
                                )
                        )
                    }
        }
    } catch (e: ExposedSQLException) {
        throw e
    }

    return departmentsList
}
```

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

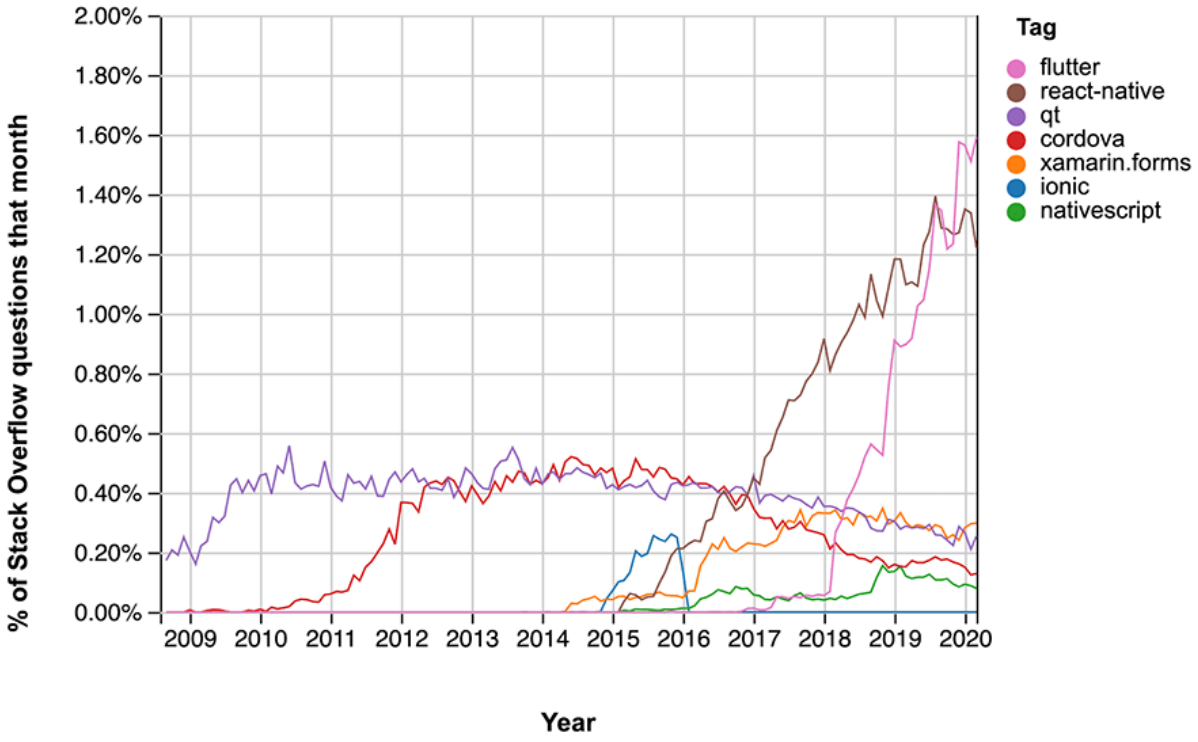

Рисунок П.1 – График статистики поисковых запросов

Приложение Р

Использование FutureBuilder

```
@override
void initState() {
  super.initState();
  futureList = this.groupItemsApiService.getGroups(this.rootItemId, type);
}

@override
Widget build(BuildContext context) {
  return FutureBuilder<List<GroupItems>>(
    future: this.futureList,
    builder: (context, snapshot) {
      if (snapshot.hasData) {
        return ListView.builder(
            padding: EdgeInsets.only(bottom: 60),
            itemCount: snapshot.data.length,
            itemBuilder: (context, index) {
              return Container(
                margin: EdgeInsets.only(top: 20),
                child: DropdownItem(
                    itemGroup: snapshot.data[index].group,
                    itemsList: snapshot.data[index].itemsList,
                    type: this.type
                ), // DropdownItem
              ); // Container
            }); // ListView.builder
      }

      if (snapshot.hasError) {
        return ConnectionError();
      }

      return Loader();
    },
  ); // FutureBuilder
}
```

Рисунок Р.1 – Программный код с примером использования FutureBuilder

Приложение С

Диаграмма развертывания системы

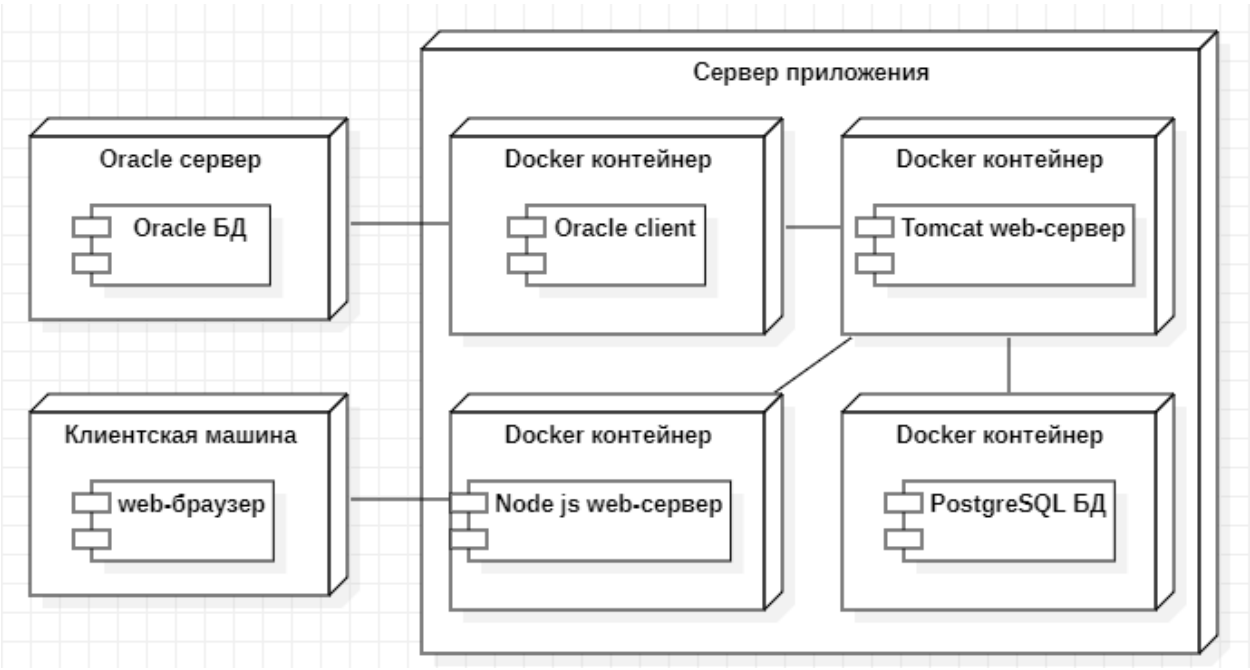

Рисунок С.1 – Диаграмма развертывания системы

Приложение Т

Docker Compose конфигурация

```yaml
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
```