Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет «Информационных систем и технологий»

Кафедра «Вычислительная техника»

Дисциплина «Сетевые технологии»

**КУРСОВОЙ ПРОЕКТ (РАБОТА)**

Тема «Автоматизация формирования индивидуальных планов преподавателей» »

Выполнил студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /П.С. Кондратьев /

подпись инициалы, фамилия

Курс 1 Группа ИВТАСмд-11

Направление/ специальность 09.04.01 «Информатика и вычислительная техника»

Руководитель ктн доцент кафедры ВТ Мартынов А.И.

Дата сдачи:

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

Дата защиты:

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ульяновск

2021 г.

Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет «Информационных систем и технологий»

Кафедра «Вычислительная техника»

Дисциплина «Сетевые технологии»

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)**

студенту ИВТАСмд-11 Кондратьев П.С.

группа фамилия, инициалы

Тема проекта (работы) «Автоматизация формирования индивидуальных планов преподавателей»

Срок сдачи законченного проекта (работы) «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

Исходные данные к проекту (работе) Индивидуальный план преподавателя.

Содержание пояснительной записки (перечень подлежащих разработке вопросов)

Пояснительная записка должна содержать 25 – более страниц текста, распределенного по разделам следующим образом:

1. Титульный лист

2. Введение

3. Расширенная постановка задачи

4. Проектирование приложения

5. Заключение

6. Список литературы

7. Приложение (листинг программы)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель ктн доцент кафедры ВТ / Мартынов А.И /

должность подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ П.С. Кондратьев /

подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г

Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**ОТЗЫВ  
руководителя на курсовой проект (работу)**

Студента Кондратьева Павла Сергеевича

фамилия, имя и отчество

Факультет ФИСТ группа ИВТАСмд-11 курс 1

Дисциплина «Сетевые технологии»

Тема проекта (работы) «Автоматизация формирования индивидуальных планов преподавателей»

Отмечаются следующие моменты: актуальность темы исследования; соответствие содержания и структуры курсовой работы ее теме; степень разработанности проблемы, наиболее интересно исследованные вопросы. Оценивается степень самостоятельности и инициативы студента; умение пользоваться различными источниками информации; уровень его теоретической подготовки; умение анализировать научные материалы, делать практические выводы; знание основных концепций, научной и специальной литературы по избранной теме. Содержится оценка проекта (работы) руководителем.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель ктн доцент кафедры ВТ \_\_\_\_\_\_\_\_\_\_\_\_\_ / Мартынов А.И /

должность, учёная степень, ученое звание подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_г.

**Оглавление**

[Введение 5](#_Toc63507173)

[Анализ методов и средств автоматизации проектирования 7](#_Toc63507174)

[Расширенная постановка задачи 8](#_Toc63507175)

[Анализ основных видов проектных решений и инструментальных средств 8](#_Toc63507176)

[Выбор модели базы данных 12](#_Toc63507177)

[Проектирование приложения 15](#_Toc63507178)

[Проектирование ER модели базы данных 15](#_Toc63507179)

[Разработка структурно-функциональной организации Автоматизация формирования индивидуальных планов 17](#_Toc63507180)

[Разработка диаграммы компонентов 17](#_Toc63507181)

[Разработка диаграммы классов 18](#_Toc63507182)

[Разработка диаграмм вариантов использования (Use case diagram) 24](#_Toc63507183)

[Разработка диаграммы деятельности 24](#_Toc63507184)

[Реализация проектных решений 25](#_Toc63507185)

[Создания одной точки вхождения для API запросов (единых, автоматических генерируемых, сущностей БД) с использованием GraphQL 25](#_Toc63507186)

[Создание Паспорт стратегии для управления состоянием проверки подлинности во время аутентификации пользователя в системе 32](#_Toc63507187)

[Модуль создания шаблона документа 36](#_Toc63507188)

[Заключение 39](#_Toc63507189)

[Список используемых источников 40](#_Toc63507190)

[Приложение 42](#_Toc63507191)

[Шаблон индивидуального плана 42](#_Toc63507192)

# Введение

Сегодня, нет ни одной сферы человеческой деятельности, которую так или иначе не коснулась бы автоматизация. Автоматизация – одно из направлений научно-технического прогресса, использующее саморегулирующие технические средства и математические методы, с целью освобождения человека от участия в процессах получения, преобразования, передачи и использования энергии, материалов, изделий или информации, либо существенного уменьшения степени этого участия или трудоёмкости выполняемых операций. С развитием науки человек всегда старался совершенствовать уже имеющиеся наработки, тем самым создавая новые и новые научно-технические решения. Одним из которых и стала автоматизация.

В настоящее время сотрудникам государственных учреждений необходимо работать с большим количеством документации. Речь идет не только о тех людях, у которых прямые обязанности заключаются в ведении документации, но и преподавателях в школах и высших учебных заведениях. Дополнительная психофизическая нагрузка, направленная на преподавателей, вызывает утомление, которое в свою очередь вызывает процесс понижения работоспособности, временный упадок сил, возникающий при выполнении определенной монотонной работы и, как следствие, это плохо сказывается на образовательном процессе.

Существуют различные формы отчетности, планы и другие документы, которые создаются на основе определенных стандартов. Как правило, сотрудники, чьи обязанности подразумевают создание таких документов, тратят на это большое количество времени и периодически допускают грамматические, орфографические, пунктуационные, фактические и другие ошибки.

Для ускорения создания типовых документов и уменьшения количества допускаемых ошибок, а именно рабочих программ для дисциплин, преподаваемых в рамках кафедры «Вычислительная техника» Ульяновского государственного технического университета, разрабатывается система автоматизированной поддержки рабочих программ.

**Целями и идеями проекта являются:**

* Разработка нового сценария взаимодействия ИП
* Развитие средств автоматизации организационного управления кафедры ВТ и УлГТУ
* Автоматизировать процесс создания рабочих программ

Улучшение UX редактора, за счёт категоризации файлов по определённым критериям

# Анализ методов и средств автоматизации проектирования

В рамках разработки автоматизированной системы поддержки рабочих программ необходимо создать автоматизированное рабочее место преподавателя, выполняющего определенную функцию – разработку рабочей программы дисциплин. Это приведёт к снижению нагрузки сотрудника, отвечающего за обработку информации и к уменьшению затрачиваемого времени на создание рабочей программы, что позволит преподавателю меньше заниматься рутинной обработкой документации и больше уделять внимания студентам и учебному процессу.

Автоматизированное рабочее место – это комплекс технических и программных средств, основным назначением которого является автоматизации профессионального труда специалиста и обеспечивающий подготовку, редактирование, поиск и выдачу на экран и печать необходимых ему документов и данных. А также обеспечивает оператора всеми средствами, необходимыми для выполнения определенных функций.

Всего можно выделить 4 задачи автоматизированной системы поддержки рабочих программ:

1. Автоматизированное рабочее место
2. Снижение нагрузки сотрудника
3. Снижение затрачиваемого времени на создание рабочей программы
4. Создание небольшой семантического разрыва между языками спецификациями и манипуляции, с одной стороны, и логической организации данных с другой стороны

# Расширенная постановка задачи

## Анализ основных видов проектных решений и инструментальных средств

Разрабатываемая подсистема автоматизации формирования индивидуальных планов является частью платформы, которая является частью автоматизированной системы поддержки рабочих программ.

Назначением данной подсистемы является работа с внешними данными, которые необходимы для создания «Рабочей программы». Внешние данные включают в себя рабочие планы дисциплин, расчеты штатов, нормы времени.

Являясь частью комплексной платформы, данная подсистема имеет общую цель системы – создание автоматизированной системы поддержки рабочих программ с возможностью использования готового шаблона и исходных данных по необходимому предмету. Главная задача данной системы – снижение нагрузки сотрудника, отвечающего за обработку информации, а также минимизация возможности совершения ошибки при оформлении рабочей программы, так как имеется стандартизованный шаблон.

Технологии проектирования и инструментальные средства, используемые во время проектирования и разработки приложения представлены в следующей таблице.

Таблица 1. Инструментальные средства

|  |  |
| --- | --- |
| https://miro.medium.com/max/1200/0*kS3GZAc7IGO3gvxx.png | NestJS Серверная часть приложения |
| https://upload.wikimedia.org/wikipedia/commons/thumb/c/cf/Angular_full_color_logo.svg/1024px-Angular_full_color_logo.svg.png | Angular9 Клиентская часть приложения |
|  | TypeScript Язык со строгой типизацией Система для работы с модулями и классами |
| https://www.ascendtraining.com/wp-content/uploads/2015/06/javascript.png | JavaScript Мультипарадигменный скриптовый язык программирования |
| https://hsto.org/webt/3l/ef/wk/3lefwko0funvsqwyvgcaqtf7ory.png | PostgresSQL Система управления базами данных |
| https://upload.wikimedia.org/wikipedia/commons/thumb/1/17/GraphQL_Logo.svg/1024px-GraphQL_Logo.svg.png | Graph QL  Облегчает агрегацию данных из нескольких источников |
| https://s3.amazonaws.com/media-p.slid.es/uploads/72196/images/1779635/pasted-from-clipboard.png | Git Система контроля версий |
| The JavaScript spreadsheet | jExcel Легкий плагин JavaScript для создания интерактивных веб-таблицы и электронных таблицы |
| https://lh3.googleusercontent.com/P8MfN2vqXmP90AGCy89Wuf_AGB40IctMdWLPBj7yxkxuwL1rscWp7V4st7vcf_IFZX0=w300 | docxtemplater Инструмент слияния почты, который используется программно и обрабатывает условия |

В современной разработке мало кто не использует готовых фреймворков, для реализации той или иной задачи. Фреймворки позволяют не реализовывать часть базового функционала и не развертывать структуру приложения из отдельных компонентов, а использовать встроенную проприетарную для фреймворка структуру.

Для текущего проекта были выбраны 2 JS фреймворка:

* Серверная часть (Back-end на Фреймворке NestJs), С использованием TypeScript
* Пользовательский интерфейс (Front-end на Фреймворке Angular) для автоматизированной системы поддержки рабочих программ, С использованием TypeScript

**Почему TypeScript?**

Размышляя о том, почему имеет смысл попробовать поработать на TypeScript, выделяются несколько основных пунктов:

1. Возможность жестко описать каждый элемент приложения – это, вероятно, исключает возможность неверной реализации или некорректного вызова методов; Заставляет разработчиков продумать логику (например, в методах) до самой реализации; Не дает возможность изменить один кусок приложения, сломав другой
2. Возможность описать область видимости свойств класса – еще один барьер, который ограничивает разработчика от совершения ошибок
3. Из-за жесткой архитектуры, возможно, необходимо писать меньше тестов – все параметры методов жестко описаны, и если код скомпилировался, тогда, вероятно, каждый вызов является валидным, и не требует дополнительной проверки
4. Многие конструкции в TypeScript имеют жесткий формат, поэтому многие из ошибок форматирования кода (в соответствии каким-то принятым в коллективе нормам) исключены. Можно считать, что это плюс, потому что процесс ревью кода часто выявляет ошибки форматирования, на которые тратится ценное время. Если это время можно потратить на другие более полезные активности — это уже плюс.

В течение времени прохождения удаленной практики в одной из IT компаний (front-end разработчиком), приходилось решать многие задачи на фреймворке Angular. По этой причине был выбран этот фреймворк, с учетом его плюсов и минусов, так же продолжилось изучения данного фреймворка в разработке ВКР.

**Пару слов о достоинствах Angular:**

* Большое количество разнообразных функций
* Функции взаимозависимы
* Информацию можно получать напрямую, а не через третьих лиц
* Представлена возможность работать отдельно в одном разделе программы, используя имеющиеся данные
* Минимальный риск ошибок

**При этом Angular имеет такие недостатки:**

* В основе сложный язык программирования
* Ошибки во время миграции между версиями

Теперь стоит рассказать о серверной часть более подробнее. Nest (NestJS. Под капотом Nest использует надежные http-серверные фреймворки, такие как Express (по умолчанию), это одно из причин использования Nest взамен всем известного использование фреймворка NodeJS – Express.

**Минусы Express:**

* гибкость и быстрое развитие порождает также и минусы т.к. надо постоянно следить за обновлениями, некоторые вещи выходят недостаточно протестированными
* был случай, когда разработчик удалил свою библиотеку из NPM и множество приложений, использующих ее перестали работать

Nest предоставляет готовую архитектуру приложений, которая позволяет разработчикам и командам создавать высоко тестируемые, масштабируемые, слабо связанные и легко ремонтопригодные приложения. Архитектура сильно вдохновлена Angular.

## Выбор модели базы данных

В качестве СУБД используется **PostgreSQL** 10.12 С использованием **Graph QL**. Это СУБД, которая использует реляционную модель для своих баз данных и поддерживает стандартный язык запросов SQL. PostgreSQL предоставляет множество различных возможностей, достаточно надежна и имеет хорошие характеристики по производительности. Кроме того, PostgreSQL свободно распространяется и имеет открытый исходный код. PostgreSQL выгодно отличается от многих других СУБД. Она обладает практически всеми возможностями, которые есть в других базах данных (коммерческих или Open Source), а также некоторыми дополнительными.

**Достоинства PostgreSQL:**

* Открытое ПО соответствующее стандарту SQL – PostgreSQL, бесплатное ПО с открытым исходным кодом
* Существует довольно большое сообщество, в котором можно найти ответы на интересующие вопросы или решить проблемы, связанные с процессом разработки
* Существует и постоянно дополняется большое количество дополнений, позволяющих разрабатывать данные для этой СУБД и управлять ими.
* Существует возможность расширения функционала за счет сохранения своих процедур
* Объектность – PostrgreSQL это не только реляционная СУБД, но также и объектно-ориентированная с поддержкой наследования и много другого
* Многомерные массивы, к примеру, MySQL, MariaDB, и Firebird так не умеют. Чтобы хранить такие массивы значений в традиционных реляционных базах данных, придется использовать обходной путь и создавать отдельную таблицу со строками для каждого из значений массива

**Недостатки PostgreSQL:**

* Производительность – при простых операциях чтения PostgreSQL может значительно замедлить сервер и медленнее чем MySQL
* Проблема с выбором хостинга для развертывания сервера PostgreSQL (требуется поиск)

Самое большое преимущество пребывания в Postgres заключается в том, что у нас есть лучшее из обоих миров. Мы можем хранить данные в JSONB с ограничениями, согласованностью и скоростью. С другой стороны, мы можем использовать все функции SQL для других типов данных.

И в заключении выбора БД и его составляющих, пару слов о **Graph QL**.

**Почему Graph QL?**

* Позволяет клиенту точно указать, какие данные ему нужны
* Облегчает агрегацию данных из нескольких источников
* Использует систему типов для описания данных
* Высокая степени готовности инструментальных средств использования в проекте, возможность избежать дублирования кода, а также имеющаяся одна точка входа
* Небольшой семантический разрыв между языками спецификациями и манипуляции, с одной стороны, и логической организации данных с другой стороны

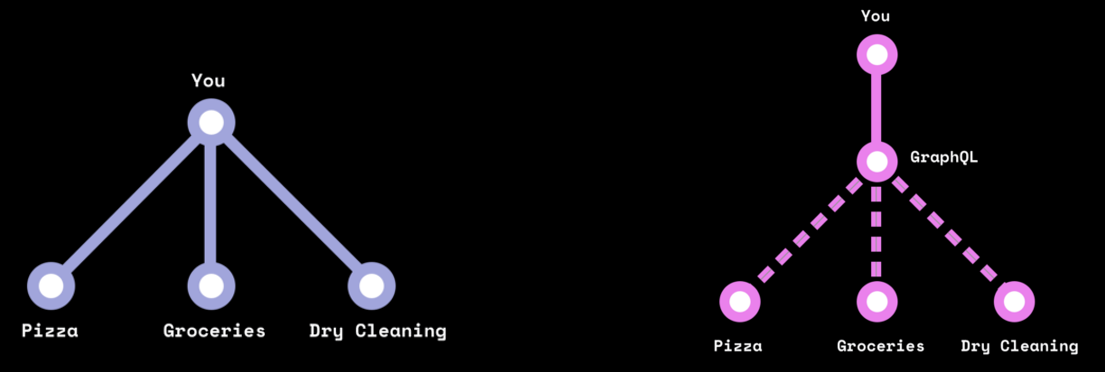


Рисунок 1 – REST-модель Graph QL и других БД

**Декларативный стиль БД**

* Фактически, слой GraphQL находится между клиентом и одним или несколькими источниками данных
* QL принимает запросы клиентов и возвращает необходимые данные в соответствии с переданными инструкциями

Пример текста **SQL**:

SELECT firstName, lastName, middleName, age FROM users WHERE id = userID

Пример текста **QL**:

query {

users(id: userID) {

firstName, lastName, middleName, age

}

}

# Проектирование приложения

## Проектирование ER модели базы данных

Инфологическая модель представляет собой описание будущей базы данных, представленное с помощью естественного языка, формул, графиков, диаграмм, таблиц и других средств, понятных как разработчикам БД, так и обычным пользователям.

По состоянию на июнь 20 года схема бд прототипа для системы поддержки формирования индивидуальных планов преподавателей представлена на рисунке 2:

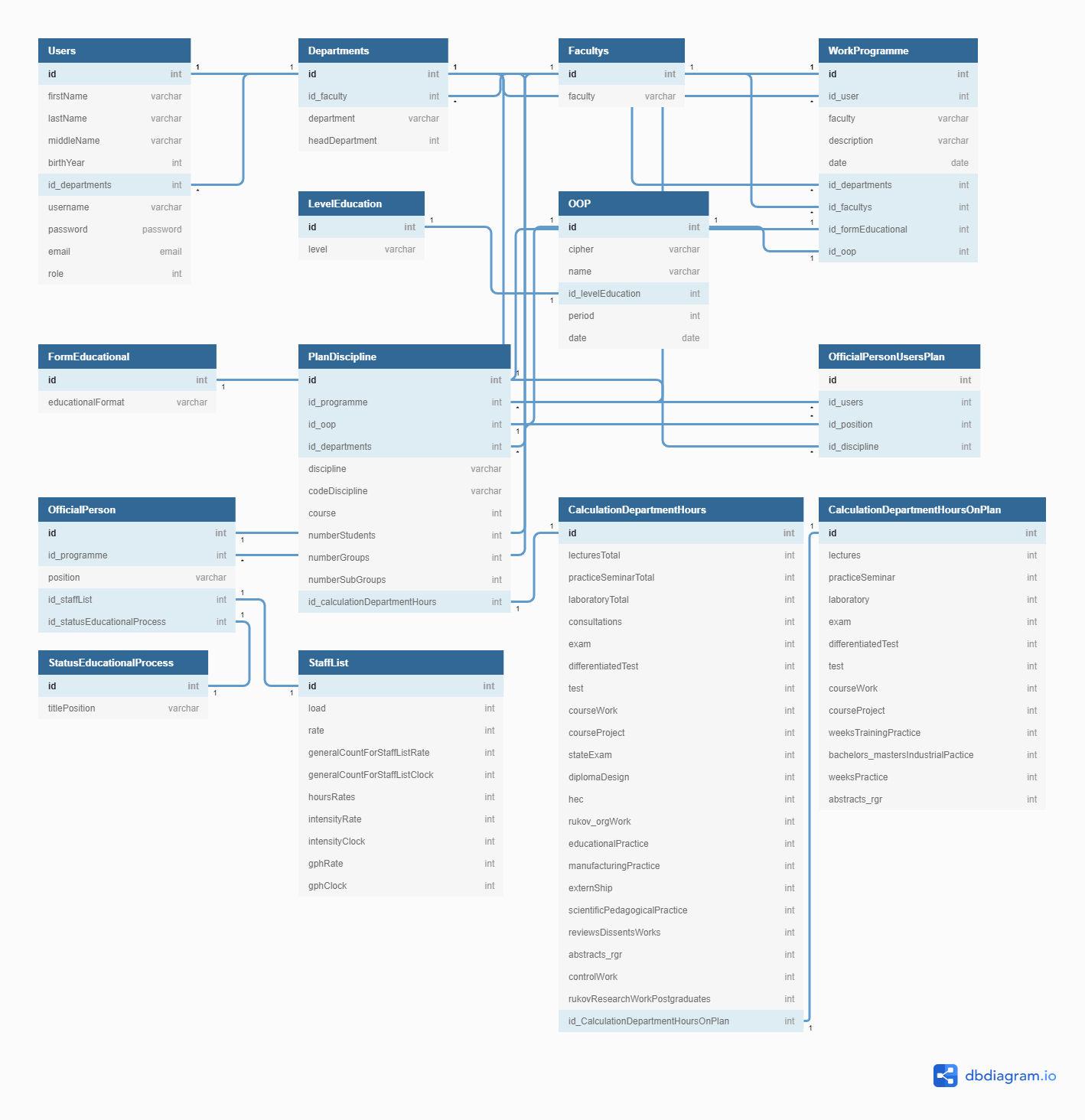


Рисунок 2 – ER диаграмма БД

Выше были показаны основные сущности разрабатываемой системы, подробнее о них рассмотрим ниже.

Users – это пользователь, зарегистрировавшийся в системе, который может просматривать или создавать свои шаблоны ИП. Предполагается, что пользователь, не авторизованный в системе не имеет доступ к системе.

Departments и Facultys – таблицы с названиями кафедр и факультетов, которые используются при заполнении титульного листа ИП.

PlanDiscipline – одна из главных таблиц описывающая дисциплины, которые соответствуют текущему пользователю (преподавателю). Данная таблица используется в связке с 3 таблицами – CalculationDepartmentHours, StaffList, OfficialPerson. За каждой дисциплиной закреплена рабочая нагрузка, которая описывается в отдельной таблице CalculationDepartmentHours. В ней указываются часы на проведение лекция, практик, лабораторный занятий, экзаменов, зачетов и тд.

У каждого преподавателя есть своя должность в вузе и к каждой должности закрепляется объем рабочей нагрузки, которая учитывается в штатном расписании кафедры, помимо должности у пользователя есть статус в учебном процессе:

* штатный
* внутр.совм.
* внешний совм.
* раб договор
* ГПХ отодатель

Все выше перечисленные должности описываются в таблице StaffList, которая в свою очередь связана с промежуточной таблицей OfficialPerson с помощью, которой каждый пользователь может получить данные о текущей программе для закрепленный за преподавателем дисциплин.

## Разработка структурно-функциональной организации Автоматизация формирования индивидуальных планов

### Разработка диаграммы компонентов

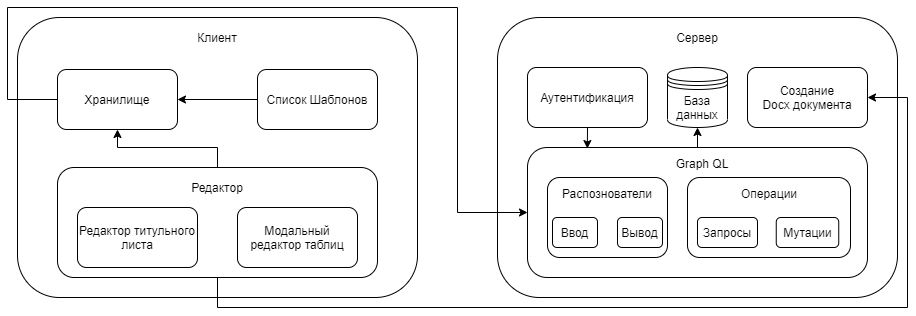


Рисунок 3 – Диаграммы компонентов

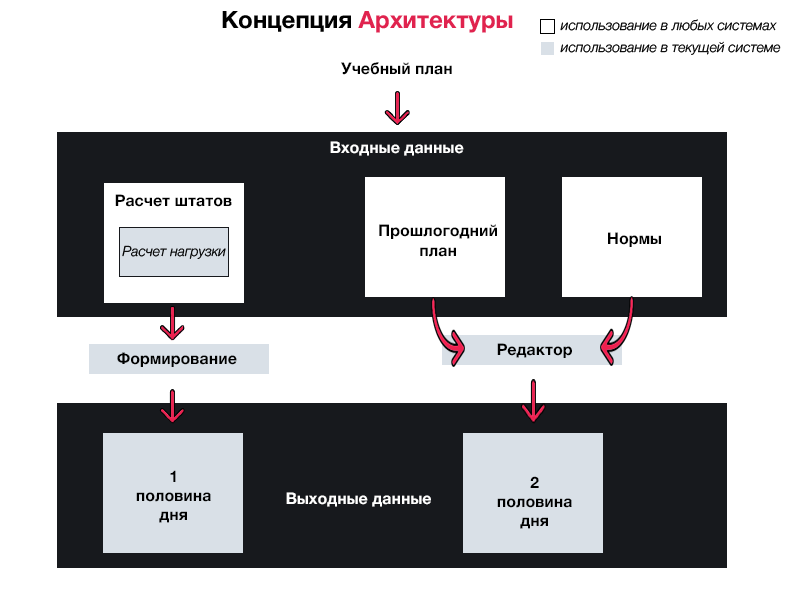


Рисунок 4 – Концепция автоматизированной системы создания индивидуальных планов преподавателя

## Разработка диаграммы классов

**Серверная часть:**

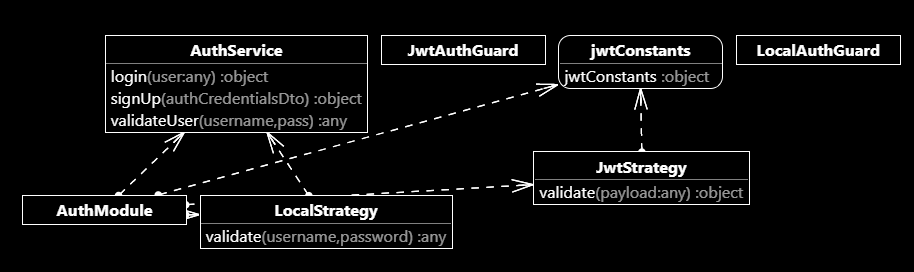


Рисунок 5 – Диаграмма класса аутентификации

Существует множество различных подходов и стратегий для обработки аутентификации. Подход, принятый для любого проекта, зависит от его конкретных прикладных требований.

Была реализована JWTStrategy с целью:

1. Проверки подлинности пользователя путем проверки его "учетных данных" (таких как имя пользователя / пароль, JSON Web Token (JWT) или Identity token от поставщика удостоверений)
2. Управления состоянием проверки подлинности (путем выдачи переносного маркера, например, JWT, или создания Экспресс-сеанса)
3. Прикрепления информации об аутентифицированном пользователе к Request объекту для дальнейшего использования в обработчиках маршрутов

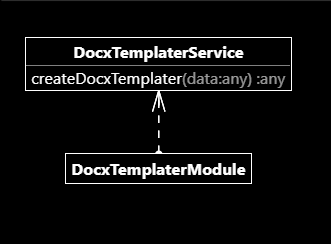


Рисунок 6 – Диаграмма класса генератора документа

Ниже описаны классовые сущности БД, из-за большого количества таблиц (похожих сущностей), связей между ними, были выделены основные классы по работе с БД.

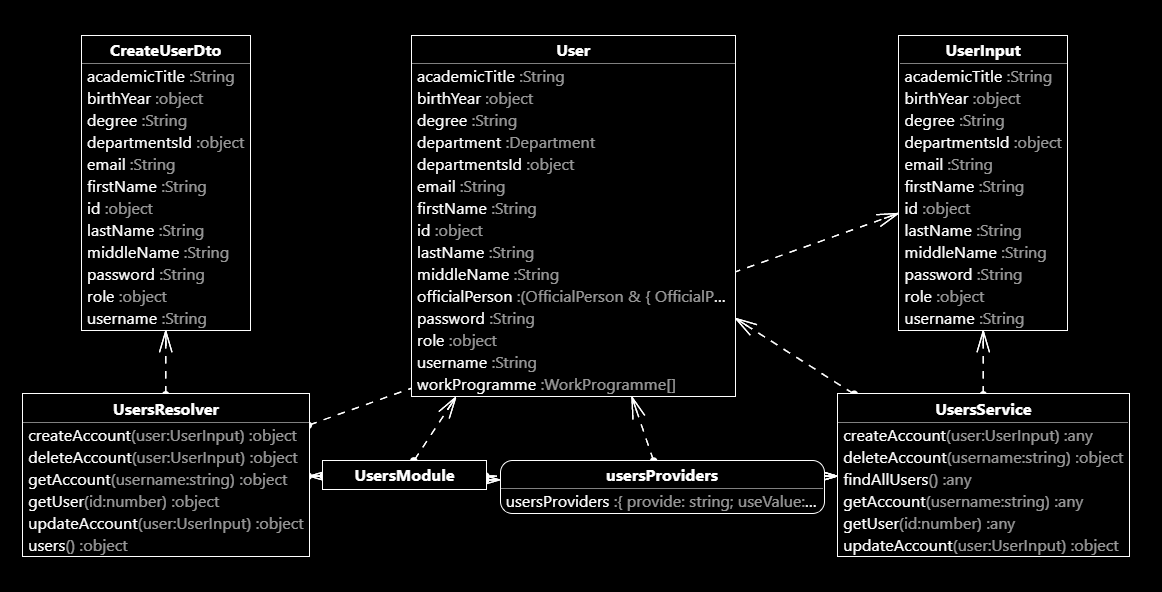


Рисунок 7 – Диаграмма класса пользователя

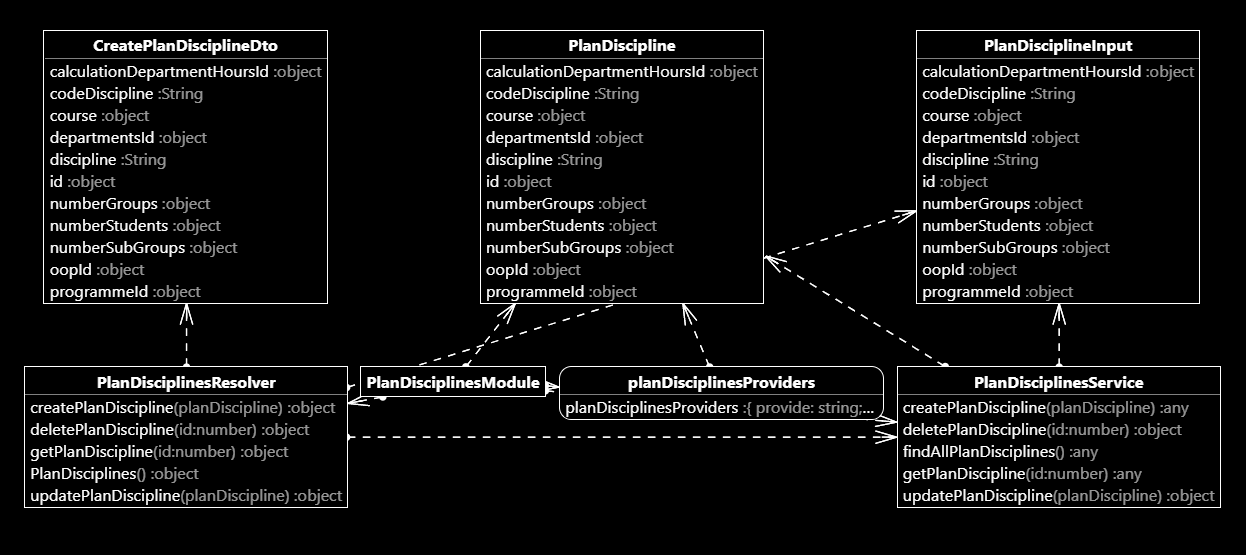


Рисунок 8 – Диаграмма класса дисциплины

Появление таких сущностей как Dto и Input, вызвано тем, что для работы c GraphQL запросами и мутациями необходимы экземпляры классов сущностей, которые разделяют ввод и вывод.

Это также особенно ценно в случае мутаций, когда вы, возможно, захотите передать весь создаваемый объект. В языке схем GraphQL входные типы выглядят точно так же, как обычные типы объектов, но с ключевым словом input вместо type. Также поля типа объекта ввода могут сами ссылаться на типы объектов ввода, но вы не можете смешивать типы ввода и вывода в своей схеме. Типы входных объектов также не могут иметь аргументов в своих полях.

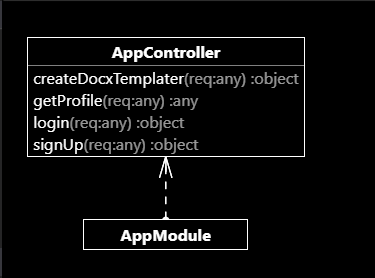


Рисунок 9 – Диаграмма класса основного контроллера

**Клиентская часть:**

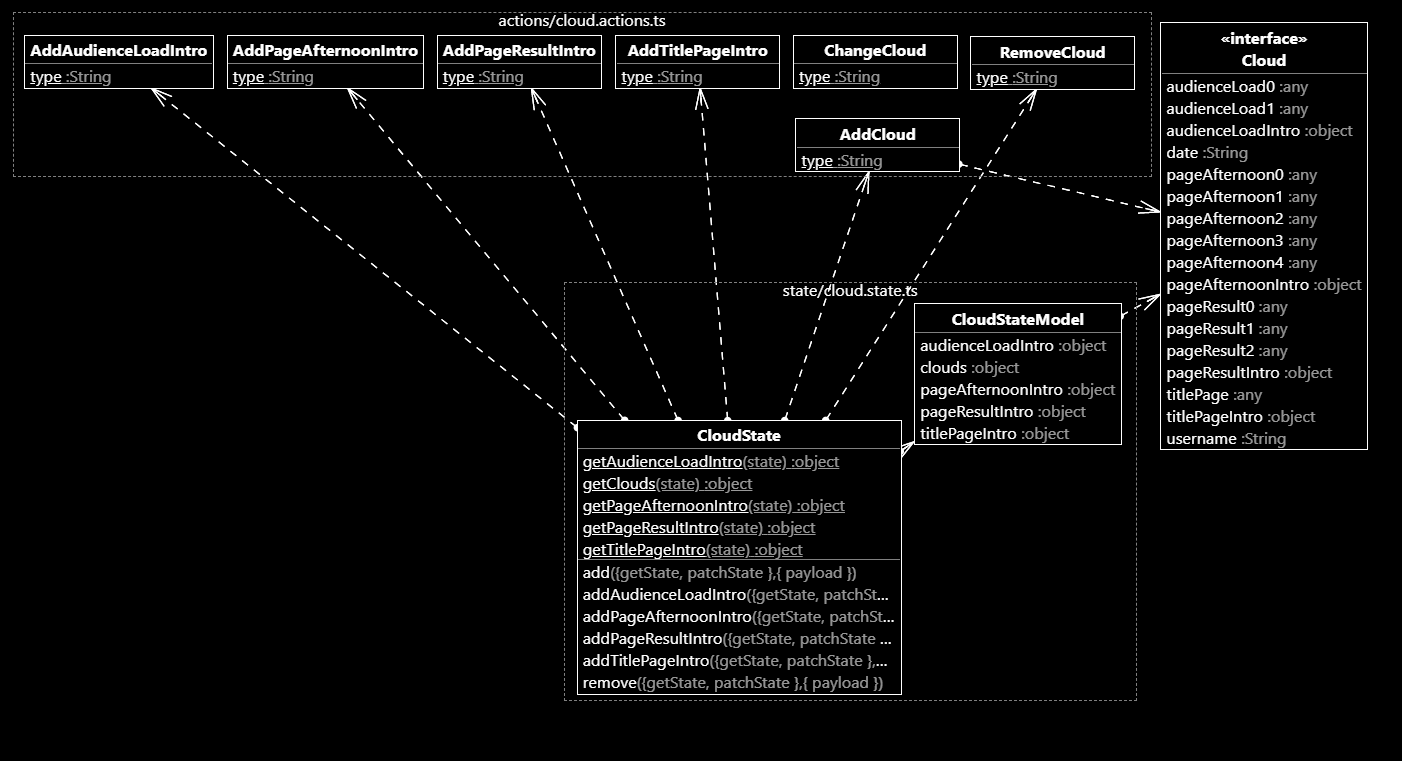


Рисунок 10 – Диаграмма класса внутреннего хранилища

NGXS – это шаблон управления состоянием + библиотека для Angular. Он действует как единственный источник истины для состояния вашего приложения, предоставляя простые правила для предсказуемых мутаций состояния.

NGXS моделируется по шаблону CQRS, обычно реализуемому в таких библиотеках, как Redux и NgRx, но уменьшает шаблонность за счет использования современных функций машинописи, таких как классы и декораторы. Благодаря такому разделению, при использовании данной модели хранения данных на клиенте имеется точное представление что именно происходит в хранилище.

Происходит это благодаря созданию единой модели хранилища представленной в виде ориентированного графа, в котором благодаря контроллерам Add\* можно следить за движениями данных в хранилище.



Рисунок 11 – Диаграмма класса модуля по работе с таблицами

В выше описанном классе по работе с таблицами был реализован единый стиль на все таблицы. Каждая из которых является частью большой карусели, в которой и происходит пред просмотр каждой из этих таблиц, которые в последующих действиях используются уже в модальном редакторе.

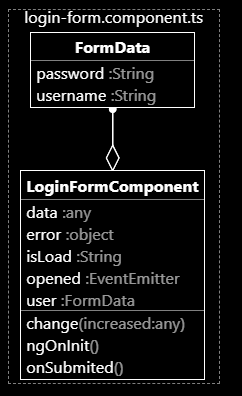


Рисунок 12 – Диаграмма класса аутентификации

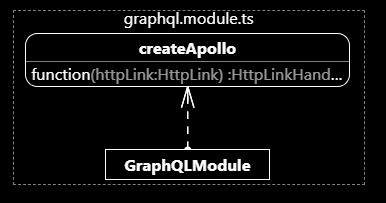


Рисунок 13 – Диаграмма класса контроллера Graph QL

Используя распознаватель, GraphQL понимает, как и где получить данные, соответствующие запрашиваемому полю. Конечно! GraphQL это только спецификация, вы можете использовать его с любой библиотекой на любой платформе, используя готовый клиент (к примеру, у Apollo есть клиенты для web, iOS, Angular и другие) или вручную отправляя запросы на сервер GraphQL.

### Разработка диаграмм вариантов использования (Use case diagram)

Ниже представлена диаграмма взаимодействия пользователя с интерфейсом, отражающую какие типы действий доступны ему.

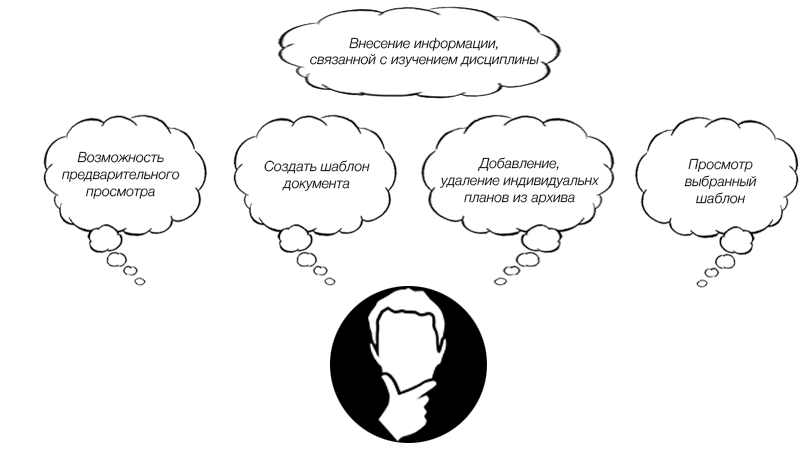


Рисунок 14 – Use case диаграмма

### Разработка диаграммы деятельности

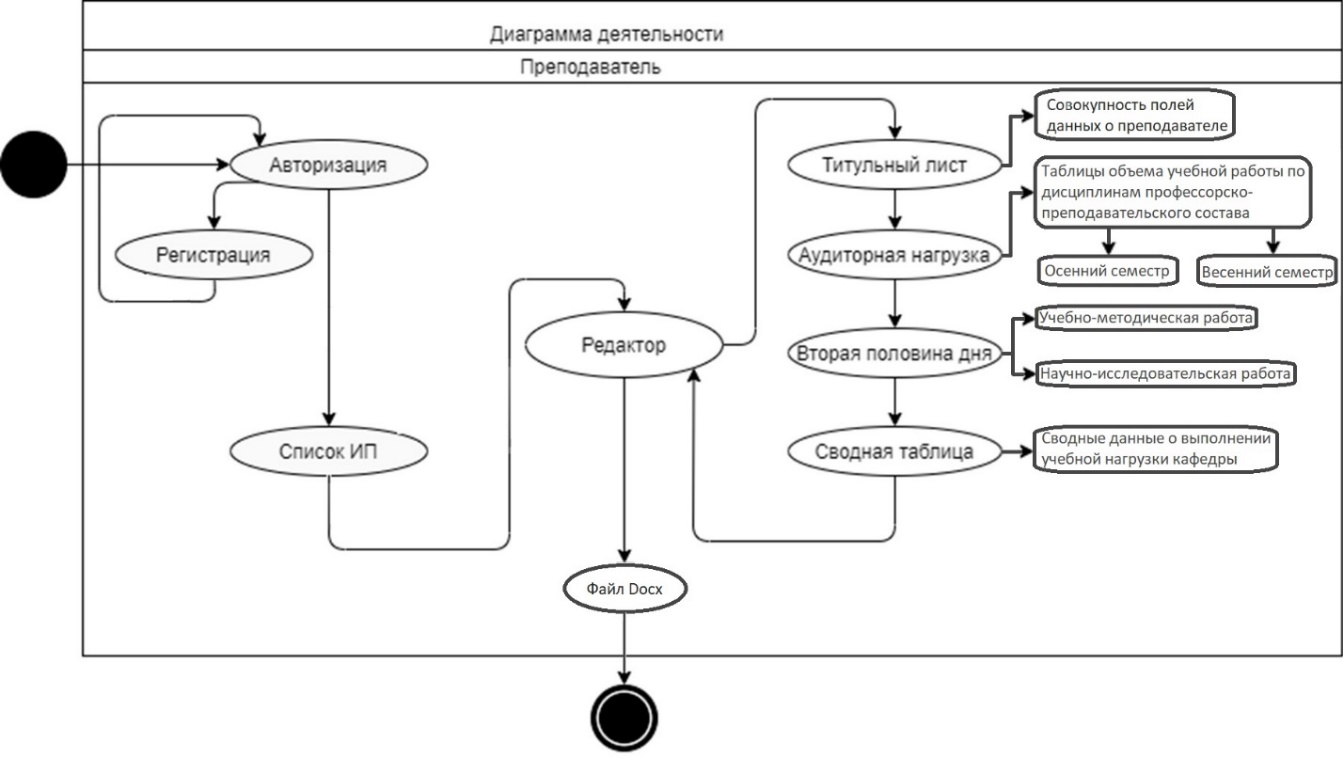


Рисунок 15 – Диаграмма деятельности прототипа автоматизированной системы поддержки рабочих программ

# Реализация проектных решений

Для проектирования системы автоматизация формирования индивидуальных планов необходимо, в первую очередь, разобраться с функциональными требованиями и той нагрузкой и задачами, которые стоят перед подсистемой в целом:

* Аутентификации пользователей с использованием технологией JWT
* Создание единой системы запросов к БД
* Модуль по просмотру выбранного Индивидуального плана
* Модуль по работе с электронными таблицами
* Создание отдельных шагов (модулей), с возможностью для преподавателя заполнить поля в ИП
* Возможность по заполненным данным создавать Индивидуальный план в формате Docx
* Аккаунты администраторов задаются с помощью должностного лица, по умолчанию, на стадии разработки, создан 1 глобальный аккаунт
* Данные из расчета штатов статичны – вбиваются вручную в тестовую БД
* Данные по Учебно-методической работе заполняются вручную по шаблону прошлого года
* Целостность данных – безопасность от перезаписи уже существующих шаблонов

### Создания одной точки вхождения для API запросов (единых, автоматических генерируемых, сущностей БД) с использованием GraphQL

Модуль создан для создания единого API для всего приложения, не ограничиваясь конкретным механизмом хранения данных. API GraphQL, которое используется приложением, доступно на многих языках. Вы предоставляете функции для каждого поля в системе типов, и GraphQL вызывает их с оптимальным параллелизмом. Для этого у GraphQL под капотом есть собственная площадка для тестирования запросов, мутаций, схем.

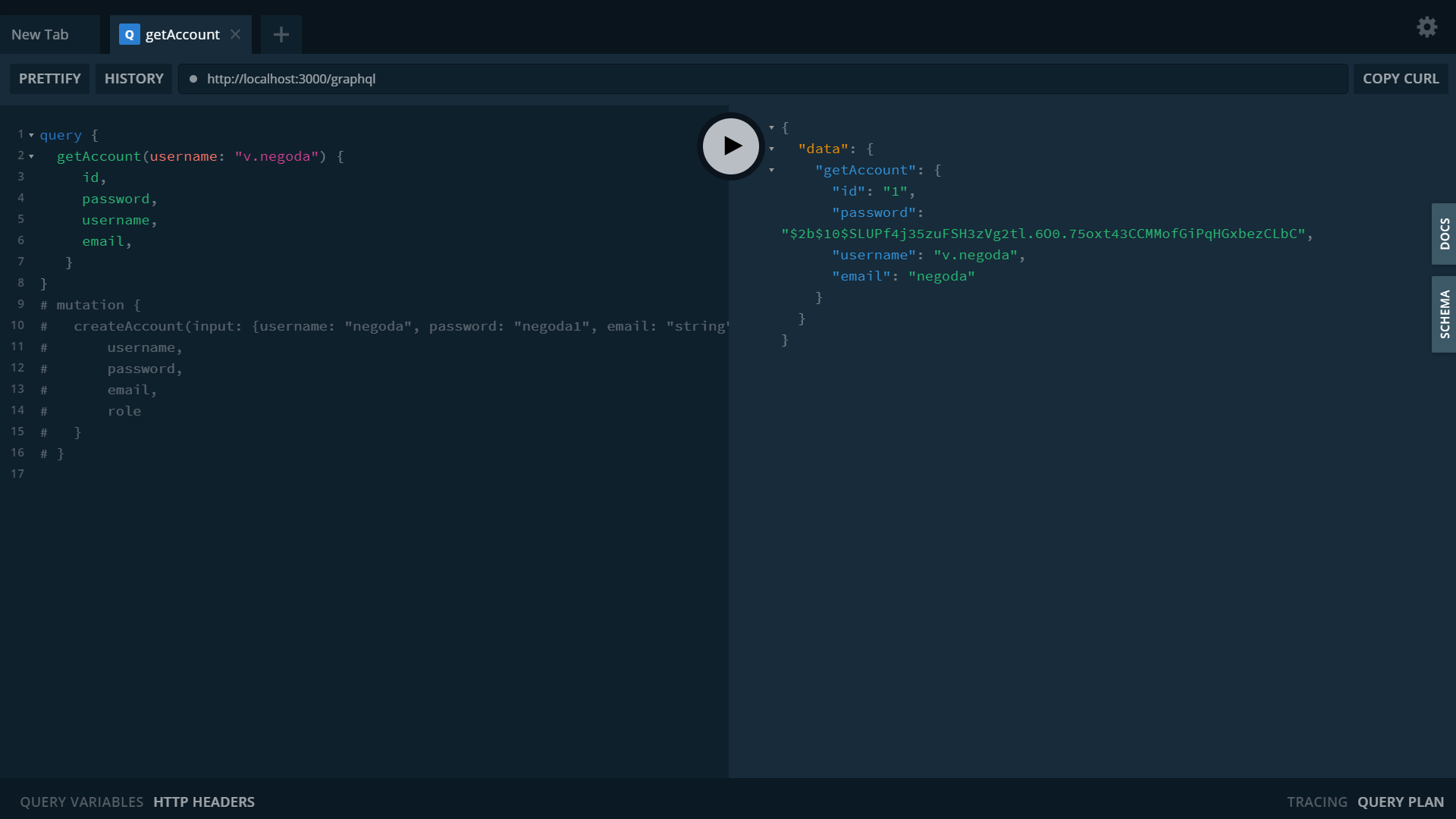


Рисунок 16. Площадка GraphQL

Стоит отметить, что на этой площадке есть документация по вашим запросам и также схема БД, все этого генерируется перед сборкой сервера. GraphQL ищет в проекте распознаватели (resolvers) для понимаю, что данный модуль относиться к QL.

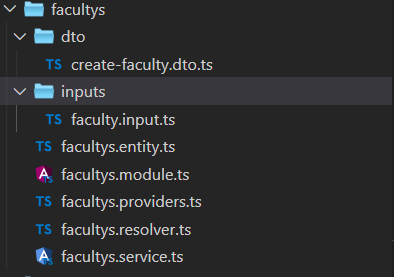


Рисунок 17. Пример файловой системы для QL

Для каждой из таблиц в БД создается папка в которой присутствуют сущности для БД, API QL (dto, input, entity), так же присутствует модуль таблицы в которой подключаются все зависимости, такие как сервис (набор методов) и распознаватели (запросы и мутации для QL).

Благодаря описанным сущностям в entitys при запуске серверной части создаются автоматически все таблицы в базе данных. Так же стоит отметить, что при создании их всех, текст SQL создания сущностей показывается в консоли.

const tableOptions: any = {

  tableName: 'facultys',

};

@Table(tableOptions)

export class Faculty extends Model<Faculty> {

  @Column({

    primaryKey: true,

    autoIncrement: true,

  })

  id: number;

  @Column

  faculty: string;

  @HasMany(() => Department)

  department: Department[];

  @HasMany(() => WorkProgramme)

  workProgramme: WorkProgramme[];

}

В самой сущности указывается имя таблицы (для создания в субд), описываются поля и их начальные параметр, к примеру, в декораторе @Column указывается дополнительные параметры первичного ключа и авто-инкрементирование для поля id. Также ниже описываются связи с другими таблицами с помощью тех же декораторов, в которых указывается к какой сущности какие ключи привязан.

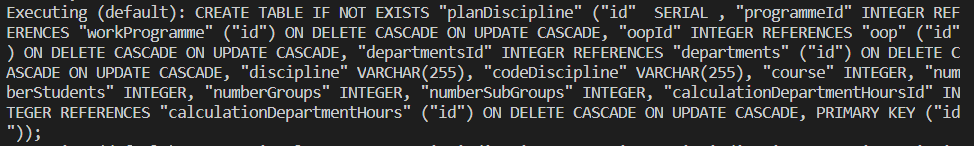


Рисунок 18. Пример SQL создания таблицы

Следующим этапом идет создания сервиса для каждой из таблицы. Во всех таблицах присутствуют CRUD функции (сокр. от англ. create, read, update, delete – «создать, прочесть, обновить, удалить»).

@Injectable()

export class FacultysService {

  constructor(

    @Inject('FACULTY\_REPOSITORY') private readonly FACULTYS\_REPOSITORY: typeof Faculty,

  ) {}

  async findAllFacultys(): Promise<Faculty[]> {

    return await this.FACULTYS\_REPOSITORY.findAll<Faculty>();

  }

  async getFaculty(id: number): Promise<Faculty> {

    return this.FACULTYS\_REPOSITORY.findByPk<Faculty>(id);

  }

  async createFaculty(faculty: FacultyInput): Promise<Faculty> {

    return this.FACULTYS\_REPOSITORY.create<Faculty>(faculty);

  }

  async updateFaculty(faculty: FacultyInput) {

    return this.FACULTYS\_REPOSITORY.update(faculty, {where: {id: faculty.id}});

  }

  async deleteFaculty(id: number) {

    return this.FACULTYS\_REPOSITORY.destroy({where: {id: id}});

  }

}

У каждого сервиса есть такое поле как Injectable. Эта концепция называется Внедрение зависимостей или dependency injection, в целом она позволяет получить более лаконичный код, уменьшить связность, и уменьшить возможные трудозатраты на поддержку кода в дальнейшем (задел на будущее).

Суть в следующем, есть объект-инжектор, при старте приложения Вы говорите ему: создай мне экземпляр класса A, он создает его, затем при помощи reflection ищет все поля, помеченные аннотацией @Inject и создает их вышеописанным же образом.

Польза в этом следующая: при разработке отдельных модулей Вы можете не оперировать конкретными объектами, а писать интерфейсы и создавать их экземпляры, не зная при этом их фактической реализации.

Но и недостатки тоже есть – в некоторых ситуациях сложно будет писать юнит тесты, так же, вы можете узнать каким будет класс только в рантайме, т.е. если в проекте сложная логика, не всегда очевидно, что в поле заинжектится, последнее затрудняет, например, навигацию в ide, и еще один немаловажный фактор – порог вхождения (иначе бы не пришлось Вам это здесь расписывать).

Стоит описать один момент связанный с TypeScript, благодаря тому, что мы имеет строгую типизацию, то мы на всем уровне реализации не можем отправлять те данные которые не были описаны в данном сервисе, которое указывает в Promise (объект, представляющий окончательное завершение или сбой асинхронной операции).

Теперь настало время создании мутаций и запросов для Graph QL.

Начнем с query запросов. При работе с GraphiQL, можно не указывать query перед фигурными скобками. При этом, при написании запросов на клиенте, если вы НЕ укажите явно тип запроса, будет ошибка.

Пример **запроса**:

@Query(() => CreateFacultyDto, { nullable: true })

  async getFaculty(@Args({ name: 'id', type: () => Int }) id: number) {

    return this.facultysService.getFaculty(id);

  }

Большинство обсуждений GraphQL сосредоточено на выборке данных, однако любая полная платформа также нуждается в способе изменения данных на стороне сервера.

В REST любой запрос может привести к некоторым побочным эффектам на сервере, но по соглашению предполагается, что GET-запросы не используются для изменения данных. GraphQL похож, - технически любой запрос может быть реализован, для того, чтобы перезаписать данные. Однако полезно установить соглашение о том, что любые операции, вызывающие изменения данных, должны быть отправлены явным образом через мутации.

Как и в запросах, если mutation поле возвращает тип объекта, Вы можете запросить вложенные поля. Это может быть полезно для извлечения нового состояния объекта после обновления.

Давайте посмотрим на простой mutation пример:

@Mutation(() => CreateFacultyDto)

  async createFaculty(@Args('faculty') faculty: FacultyInput) {

    return this.facultysService.createFaculty(faculty);

  }

Пример использования запросов и мутаций показан на Рисунок 16.

Как можно заметить из выше перечисленного по работе с запросами и мутациями (для изменения, добавления данных), клиенту при формировании запроса не нужно знать откуда поступают данные. Он просто спрашивает о них, а сервер GraphQL заботится об остальном.

После создания всей директории для работы с Graph QL при запуске серверной части, так же создается и API по работе с QL, которую можно посмотреть на тренировочной площадке перейдя на <http://localhost:3000/graphql>.



Рисунок 19. Пример схемы QL

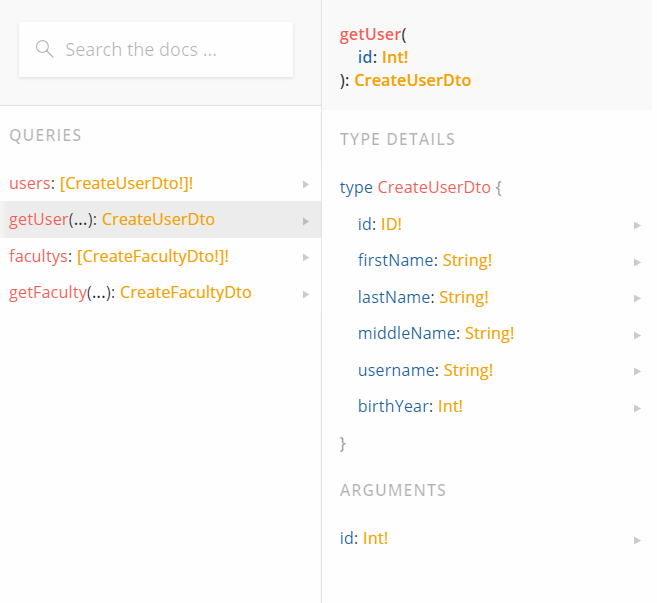


Рисунок 20. Пример запросов QL

## **Создание Паспорт стратегии для управления состоянием проверки подлинности во время аутентификации пользователя в системе**

Модуль аутентификация является неотъемлемой частью большинства приложений. Существует множество различных подходов и стратегий для обработки аутентификации. Подход, принятый для любого проекта, зависит от его конкретных прикладных требований.

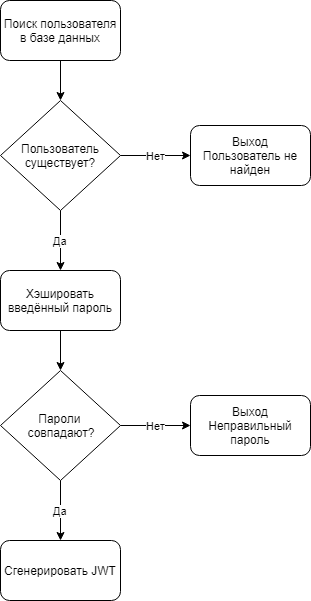


Рисунок 21. Блок схема алгоритма при аутентификации

Вот как выглядит схема действий, выполняемых в том случае, когда пользователь пытается войти в систему.

Вот что происходит при входе пользователя в систему:

* Клиент отправляет серверу комбинацию, состоящую из публичного идентификатора и приватного ключа пользователя. Обычно это – адрес электронной почты и пароль
* Сервер ищет пользователя в базе данных по адресу электронной почты
* Если пользователь существует в базе данных – сервер хэширует отправленный ему пароль и сравнивает то, что получилось, с хэшем пароля, сохранённым в базе данных
* Если проверка оказывается успешной – сервер генерирует так называемый токен или маркер аутентификации – JSON Web Token

JWT – это временный ключ. Клиент должен отправлять этот ключ серверу с каждым запросом к аутентифицированной конечной точке.

Паспорт является самым популярным узлом библиотеки аутентификации js, хорошо известная сообществу и успешно используемая во многих производственных приложениях. На высоком уровне Passport выполняет ряд шагов, чтобы:

* Проверка подлинности пользователя путем проверки его "учетных данных" (таких как имя пользователя / пароль, JSON Web Token (JWT) или Identity token от поставщика удостоверений)
* Управление состоянием проверки подлинности (путем выдачи переносного маркера, например, JWT, или создания Экспресс-сеанса)
* Прикрепите информацию об аутентифицированном пользователе к Request объекту для дальнейшего использования в обработчиках маршрутов

**Требования к стратегии:**

* Разрешить пользователям аутентифицировать с помощью имени пользователя / пароля, возвращая JWT для использования в последующих вызовах к защищенным конечным точкам API. Мы уже на пути к выполнению этого требования. Чтобы завершить его, нам нужно будет написать код, который выдает JWT
* Создание защищенных маршрутов API на основе наличия допустимого JWT в качестве токена носителя

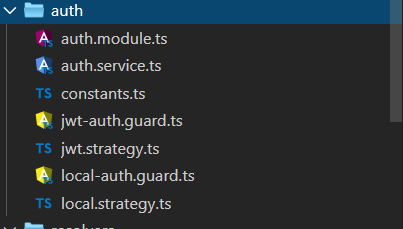


Рисунок 22. Пример файловой системы аутентификации

Пример проверки пользователя (для поиска пользователя используется уже готовые запросы к GraphQL):

async validateUser(username: string, pass: string): Promise<any> {

    const user = await this.usersService.getAccount(username);

    const compare = await bcrypt.compare(pass, user.password);

    if (user && compare) {

      const { password, ...result } = user;

      return result;

    }

    return null;

Пример обработчика запросов:

@UseGuards(LocalAuthGuard)

    @Post('auth/login')

    async login(@Request() req) {

        return this.authService.login(req.user);

    }

    @Post('auth/signUp')

    async signUp(@Request() req) {

        console.log(req.body);

        return this.authService.signUp(req.body);

    }

    @UseGuards(JwtAuthGuard)

    @Get('profile')

    getProfile(@Request() req) {

        return req.user;

    }

Во время обращения к методу login NestJS накладывает Защитника (LocalAuthGuard) для проверки принятых данных и отправки JWT токена.

Давайте более подробно рассмотрим, как POST /auth/login обрабатывается запрос. Мы оформили маршрут, используя встроенный AuthGuard компонент, предусмотренный паспортно-локальной стратегией.

Это означает, что:

* Обработчик маршрута будет вызван только в том случае, если пользователь был проверен
* Req Параметр будет содержать userсвойство (заполненное Passport во время потока passport-local authentication)

Имея это в виду, теперь мы можем, наконец, создать настоящий JWT и вернуть его в этом маршруте. Чтобы сохранить наши услуги чисто модульными, обрабатывания и генерация производиться в JWTauthService.

Так как Nest использует @nestjs/jwt библиотеку, которая предоставляет sign() функцию для создания нашего JWT из подмножества свойств user объекта, которые мы затем возвращаем как простой объект с одним access\_token свойством.

### Модуль создания шаблона документа

Создание и сборка шаблона документа происходит с помощью стороннего плагина **docxtemplater.** Это инструмент слияния, который используется программно и обрабатывает условия, циклы и может быть расширен для вставки чего угодно (таблиц, html, изображений).

Docxtemplater использует JSON (Javascript objects) в качестве ввода данных, поэтому его также можно легко использовать с других языков. Он обрабатывает docx, но также и PPTX шаблоны. Он работает так же, как и шаблонный движок.

Многие решения, такие как docx.js, docx4j, python-docx могут генерировать docx, но они требуют, чтобы вы написали определенный код для создания заголовка, изображения и тд. В отличие от этого, docxtemplater основан на понятиях тегов, и каждый тип тегов предоставляет пользователю, пишущему шаблон, определенную функцию.

Docx – это сжатый формат, содержащий некоторое количество xml. Если вы хотите построить простую замену {tag} системой значений.

В данном модуле присутствуют 2 основных элемента это сам шаблон документа и сервис по заполнению данного шаблона.

Сервис DocxTemplaterService реализует один метод который принимает массив объектов, который используется в дальнейшем для подставленный в готовый загруженный шаблон.

export class DocxTemplaterService {

    async createDocxTemplater(data: any) {

        let content = fs.readFileSync('E:/Diplom project/Back-end/src/docx-templater/templater/template.docx', 'binary');

        let zip = new PizZip(content);

        let doc = new Docxtemplater();

        doc.loadZip(zip);

        //set the templateVariables

        doc.setData(data);

        try {

            // render the document

            doc.render();

        }

        catch (error) {

            // The error thrown here contains additional information when logged with JSON.stringify (it contains a properties object containing all suberrors).

            function replaceErrors(key, value) {

                if (value instanceof Error) {

                    return Object.getOwnPropertyNames(value).reduce(function(error, key) {

                        error[key] = value[key];

                        return error;

                    }, {});

                }

                return value;

            }

            console.log(JSON.stringify({error: error}, replaceErrors));

            if (error.properties && error.properties.errors instanceof Array) {

                const errorMessages = error.properties.errors.map(function (error) {

                    return error.properties.explanation;

                }).join("\n");

                console.log('errorMessages', errorMessages);

                // errorMessages is a humanly readable message looking like this

                // 'The tag beginning with "foobar" is unopened'

            }

            throw error;

        }

        let buf = doc.getZip().generate({type: 'nodebuffer'});

        fs.writeFileSync('E:/Diplom project/Back-end/src/docx-templater/templater/output.docx', buf);

    }

}

Пример шаблона указан в Приложении.

# Заключение

В ходе данной разработки практики были освоены практические навыки работы различных серверных и фронтовых фреймворков таких как Angular и NestJS, что позволило создать концепт автоматизированной системы поддержки рабочих программ. Рассмотрены и выявлены основные цели моделирования данных в ходе проектирования информационной системы, которая обеспечит простоту создания индивидуального плана преподавателя.

Были анализированы представления учебных планов образовательных программ и выявления главных семантик.

При этом было исследована предметная область «Автоматизированных рабочих мест» и «Автоматизированных систем», а также рассмотрен процесс создания индивидуальных планов преподавателей.

Поскольку разработка комплексной системы автоматизации формирования индивидуальных планов является нетривиальной задачей, в рамках выпускной квалификационной работы будет реализован только основной функционал обучающей системы, остальная часть разработки планируется в магистратуре, поскольку проект требует не одного человека, как вариант планируется привлечь, заинтересовать людей для выбора себе частей системы и их реализации.

Проанализированы итоги прототипа и сферы применения ВКР, выделяются 3 сферы использования:

1. WEB-версия редактора
2. Аудитория: преподаватели

Преимущественно образовательный сегмент для создания индивидуального плана преподавателя на год

# Список используемых источников

1. Книга «Чистая архитектура. Искусство разработки программного обеспечения» – Режим доступа: https://habr.com/ru/company/piter/blog/353170/
2. Философия использования Nest CLI – Режим доступа: <https://docs.nestjs.com/#philosophy>
3. PostgreSQL Tutorial – Режим доступа: <https://www.postgresqltutorial.com/>
4. Язык запросов для вашего API – Режим доступа: <https://graphql.org/>
5. jExcel CE is a JavaScript plugin – Режим доступа: <https://github.com/paulhodel/jexcel>
6. Docxtemplater – Режим доступа: <https://docxtemplater.readthedocs.io/en/latest/>
7. Индивидуальный план преподавателя на 18-19 год – Режим доступа: ксерокопия
8. Расчет штатов – Режим доступа: электронная таблица excel
9. Что же такое этот GraphQL? – Режим доступа: <https://habr.com/ru/post/326986/>
10. Простой шаблон управления состоянием - NGXS – Режим доступа: <https://www.ngxs.io/>
11. Angular Ngxs Tutorial-альтернатива Ngrx для управления состояниями – Режим доступа: <https://coursetro.com/posts/code/152/Angular-NGXS-Tutorial---An-Alternative-to-Ngrx-for-State-Management>
12. MongoDB и PostgreSQL мысли – Режим доступа: <https://stackoverflow.com/questions/4426540/mongodb-and-postgresql-thoughts>
13. Is Postgres NoSQL Better Than MongoDB? – Режим доступа: <http://www.aptuz.com/blog/is-postgres-nosql-database-better-than-mongodb/>
14. Туториал по Angular HttpClient – Режим доступа: <https://ua-blog.com/туториал-по-angular-httpclient/>
15. Диаграмма связей базы данных – Режим доступа: <https://dbdiagram.io/d/5ea0718e39d18f5553fe0ae6>
16. Текстовые редакторы и текстовые процессоры – Режим доступа: https://inf1.info/wordprocessor - Загл. с экрана.
17. node.bcrypt.js – Режим доступа: <https://www.npmjs.com/package/bcrypt>
18. Pidcrypt (JS encryption library) for node.js/browserify – Режим доступа: <https://www.npmjs.com/package/pidcrypt>
19. Angular курсы – Режим доступа: <https://www.udemy.com/>
20. Руководство по TypeScript – Режим доступа: <https://metanit.com/web/typescript/>
21. БЭМ (Блок, Элемент, Модификатор) — компонентный подход к веб-разработке. – Режим доступа: <https://ru.bem.info/>

CSS с суперсилой – Режим доступа: https://sass-scss.ru/

# Приложение

## Шаблон индивидуального плана

