

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»

К защите допустить  
Зав. кафедрой ВТ  
\_\_\_\_\_ К.В. Святков

«    » июня 2020г.

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к выпускной квалификационной работе**

на тему: **«Автоматизация формирования индивидуальных планов»**

Направление: 09.03.01 «Информатика и вычислительная техника»

Выполнил студент гр. ИВТАПбд-41 \_\_\_\_\_ Кондратьев П.С.

Научный руководитель \_\_\_\_\_ Негода В.Н., профессор

**Ульяновск - 2020**

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ  
федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

09.03.01 «Информатика и вычислительная техника»

(шифр и наименование направления)

«Вычислительные машины, комплексы, системы и сети»

(профиль направления подготовки)

Квалификация Бакалавр

(бакалавр/магистр/специалист)

Факультет Информационных систем и технологий

(наименование факультета, где осуществляется обучение по направлению/магистерской программе/специальности)

**УТВЕРЖДАЮ**

Заведующий кафедрой

«Вычислительная техника»

(наименование кафедры)

к.т.н., доцент Святков К.В.

(ученая степень, ученое звание, Ф.И.О.)

(подпись)

«\_\_\_» июня 2020г.

**ЗАДАНИЕ**

на выполнение

выпускной работы бакалавра

(наименование выпускной квалификационной работы)

Студенту группы **ИВТАПбд-41**  
обучения

очной формы

Кондратьеву Павлу Сергеевичу

(фамилия, имя, отчество)

1. Тема: «Автоматизация формирования индивидуальных планов»

(наименование ВКР)

утверждена приказом по Университету от «\_\_\_» \_\_\_\_\_ 202\_г. № \_\_\_\_\_

2. Срок сдачи студентом выпускной работы бакалавра «\_\_\_» июня 2020г.

(наименование ВКР)

3. Цели и задачи:

Разработать автоматизированную систему формирования индивидуальных планов.

\* Название темы указывается в точном соответствии с приказом.

4. Исходные данные:

— Расчет штатов ВТ;

- Нормы времени;
- Индивидуальный план преподавателя;
- Рабочие программы прошлых лет

Автоматизированная система должна реализовать:

- Авторизация/Регистрация преподавателей с помощью JWT стратегии;
- Модуль по просмотру выбранного Индивидуального плана;
- Создание единой системы запросов к БД;
- Модуль по работе с электронными таблицами;
- Создание отдельных шагов (модулей), с возможностью для преподавателя заполнить поля в ИП;
- Данные из расчета штатов статичны – вбиваются вручную в тестовую БД;
- Данные по Учебно-методической работе заполняются вручную по шаблону прошлого года;
- Возможность по заполненным данным создавать Индивидуальный план в формате Dosh;
- Аккаунты администраторов задаются с помощью должностного лица, по умолчанию, на стадии разработки, создан 1 глобальный аккаунт;

## 5. Перечень вопросов, подлежащих разработке, или краткое содержание

выпускной работы бакалавра:

(наименование ВКР)

- расширенная постановка задачи;
- проектирование системы;
- реализация системы

## 6. Перечень иллюстративного материала:

Общая структура системы, диаграмма вариантов использования, диаграмма последовательности, диаграмма деятельности, диаграмма компонентов,

диаграмма классов ER-диаграммы, алгоритм работы рассмотренных аналогов и алгоритм работы реализованной системы.

Дата выдачи задания «\_\_\_»\_\_\_\_\_ 2020 г.

Руководитель \_\_\_\_\_ / В.Н. Негода /  
должность, учёная степень, ученое звание      подпись      инициалы, фамилия

Задание принял к исполнению \_\_\_\_\_ / \_\_\_\_\_ /  
подпись обучающегося                      инициалы, фамилия

## Содержание

Аннотация .....	7
Введение .....	8
Глава 1. Анализ инструментальных средства автоматизации организационного управления учебного процесса.....	10
1.1. Метрологическое обеспечение. Основные понятия.....	10
1.2. Анализ предметной области .....	11
1.3. Анализ проблематики проектирования .....	13
1.4. Анализ методов и средств автоматизации проектирования.....	14
1.5. Понятие «Спецификации индивидуального плана», структура, алгоритм создания документа рабочей программы .....	15
Глава 2. Разработка расширенной постановки задачи .....	22
2.1. Анализ проблематики проектирования АСУ.....	22
2.2. Анализ основных видов проектных решений и инструментальных средств поддержки их формирования .....	23
2.3. Анализ методов и средств создания информационного обеспечения САПР .....	24
2.4. Анализ основных видов проектных решений и инструментальных средств.....	25
2.5. Выбор модели базы данных .....	30
2.6. Требования к документированию и оформлению исходного кода .....	36
2.7. Требования к уровню реализации функций в бакалаврской работе .....	40
Глава 3. Проектирование приложения.....	41
3.1. Проектирование ER модели базы данных.....	41
3.2. Разработка структурно-функциональной организации Автоматизация формирования индивидуальных планов.....	43
3.2.1. Разработка диаграммы компонентов .....	43
3.2.2. Разработка диаграммы классов .....	44
3.2.3. Разработка диаграмм вариантов использования (Use case diagram).....	49
3.2.2. Разработка диаграммы деятельности.....	50
Глава 4. Реализация проектных решений .....	51
4.1. Описание исходных кодов и процесса их отладки в Back-end .....	51

4.1.1 Создания одной точки вхождения для API запросов (единых, автоматических генерируемых, сущностей БД) с использованием GraphQL .....	51
4.1.2 Создание Паспорт стратегии для управления состоянием проверки подлинности во время аутентификации пользователя в системе .....	57
4.1.3 Модуль создания шаблона документа .....	61
4.2. Описание исходных кодов и процесса их отладки в Front-end .....	63
4.2.1 Модуль аутентификация и посадочной страницы .....	63
4.2.2 Модуль отображения списка шаблонов пользователя .....	65
4.2.3 Основной модуль редактора индивидуального плана .....	66
Заключение .....	75
Список использованных источников .....	76
Приложение .....	79
Листинг программы .....	79
Шаблон индивидуального плана .....	104

**Аннотация**  
**к бакалаврской работе Кондратьева Павла Сергеевича на тему**  
**«Автоматизация формирования индивидуальных планов» Кафедра ВТ**  
**УлГТУ, 2020г.**

Научный руководитель профессор, Негода В.Н

**Работа** содержит 107 страницы, включающих, 31 иллюстраций, 1 приложение и список информационных источников из 38 наименования.

**Ключевые слова:** автоматизированная система, индивидуальный план преподавателя, автоматизация организационного управления учебного процесса.

Представленная бакалаврская работа посвящена созданию системы автоматизации формирования индивидуальных планов.

Пояснительная записка содержит следующие разделы:

1. Анализ инструментальных средства автоматизации организационного управления учебного процесса.
2. Разработка расширенной задачи. В данной главе рассматриваются описание задач системы, цели и задачи проекта, описываются реализуемые процессы и также приведен комплекс используемых технических средств и обоснование данного выбора.
3. Проектирование приложения. Эта глава включает в себя описание процесса функционирования системы, описание ER – диаграммы сущностей базы данных, компонентов, классов, диаграммы деятельности.
4. В четвертой главе описан процесс реализация проектных решений и приведен пример работы приложения.

## **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

# **Глава 1. Анализ инструментальных средства автоматизации организационного управления учебного процесса**

## **1.1. Метрологическое обеспечение. Основные понятия**

К основным задачам метрологического обеспечения производства продукции можно отнести:

- Обеспечение единства измерений при разработке, производстве и испытаниях продукции
- Анализ и установление рациональной номенклатуры измеряемых параметров и оптимальных норм точности измерений при контроле показателей качества продукции, параметров технологических процессов, контроле характеристик технологического оборудования
- Организация и обеспечение метрологического обслуживания средств измерений: учета, хранения, поверки, калибровки, юстировки, наладки, ремонта
- Разработка и внедрение в производственный процесс методик выполнения измерений, гарантирующих необходимую точность измерений
- Осуществление надзора за контрольным, измерительным и испытательным оборудованием в реальных условиях эксплуатации, за соблюдением установленных метрологических правил и норм
- Проведение метрологической экспертизы конструкторской и технологической документации
- Организация и обеспечение метрологического обслуживания испытательного оборудования: учет, аттестация в соответствии с установленными требованиями, ремонт

- Организация и обеспечение метрологического обслуживания средств допускового контроля: учет, аттестация, поверка, калибровка, наладка
- Внедрение современных методов и средств измерений, автоматизированного контрольно-измерительного оборудования, измерительных систем
- Оценивание технических и экономических последствий неточности измерений
- Разработка и внедрение нормативных документов, регламентирующих вопросы метрологического обеспечения
- Оценивание экономической эффективности затрат на метрологическое обеспечение

Кроме этого, одной из основных задач является работа над повышением эффективности метрологического обеспечения.

## **1.2. Анализ предметной области**

Для выполнения выпускной бакалаврской работы необходимо определить предметную область и основные понятия, которые необходимо изучить. Начать необходимо с понятия «Автоматизированная система».

Автоматизированная система (АС) - это организованная совокупность средств, методов и мероприятий, используемых для регулярной обработки информации для решения задачи.

Задачи, которые реализует АС, могут быть различны: создание модели управления информационной системой, повышение качества работы уже существующей подсистемы или ее отдельных показателей и т.д. Задачей автоматизированной системы поддержки рабочих программ является уменьшение количества затраченного времени преподавателем на создание рабочих программ, уменьшение количества ошибок в рабочих программах за счет частичной вставки заранее подготовленных данных. Такая система

поможет преподавателям более эффективно распределять свое время и уделять больше внимания процессу обучения.

Автоматизированное рабочее место (АРМ) - комплекс среди вычислительной техники и программного обеспечения, располагающийся, непосредственно на рабочем месте сотрудника и предназначенный для автоматизации его работы в рамках специальности.

Основной частью АРМ принято считать персональный компьютер. Специалисты, использующие в своей работе АРМ имеют разный род занятий, поэтому АРМ включает в себя: средства связи, анализаторы, комплексы автоматизированных устройств, приборы, отвечающие за контроль и регистрацию параметров технологических процессов, а также комплексы, которые включают в себя ряд устройств, занимающиеся измерениями информации.

В зависимости от рода деятельности структура автоматизированной системы варьируется, но основные принципы создания остаются неизменными:

- принцип эффективности
- принцип системности
- принцип гибкости
- принцип устойчивости

Под **эффективностью** системы стоит рассматривать экономическую выгоду от реализации проекта с учетом затрат на создание и эксплуатацию системы.

Большую роль в создании играет принцип **системности**. Под этим следует понимать взаимосвязь компонентов системы.

Принцип **гибкости** означает способность системы к возможным усовершенствованиям не только программного обеспечения, но и комплекса вспомогательных технических средств.

**Устойчивость** заключается в выполнении заложенных функций, независимо от факторов воздействия. Если же не удалось избежать сбоев,

работоспособность системы должна быстро восстанавливаться, а все неполадки отдельных элементов должны легко ликвидироваться. Так же в такой системе (архитектуре) должна быть способность к взаимодействию, при этом независимость компонентов друг от друга.

Такую архитектуру называют КОП-системы (компонентно-ориентированного программирования) построена таким образом, что компоненты не зависят друг от друга и являются взаимозаменяемыми. Архитектор может собирать требуемую систему из компонентов как из кубиков детского конструктора. В случае изменения функциональности компонента, от программиста может потребоваться перекомпилировать отдельный компонент, но не все приложение.

При этом компонент не автономен - он находится в тесном взаимодействии со средой. О среде компонент знает ровно столько, сколько ему необходимо для реализации заявленной функциональности

Таким образом, автоматизированная система - это комплекс, состоящий из персонала и комплекса средств автоматизации его деятельности, реализующая информационную технологию выполнения установленных функций.

### **1.3. Анализ проблематики проектирования**

На данном этапе разработки выявлено 4 проблемы:

1. Проектирование архитектуры системы, которая является определением автоматизируемых функций, т.е. представляет собой организационно-техническую систему, предназначенную для автоматизации процесса проектирования, состоящую из персонала и комплекса технических, программных и других средств автоматизации его деятельности
2. Разделение программных модулей по типам (универсальные программные модули, программные модули слоя правил бизнеса, программные модули слоя документов, программные модули

управления бизнес-процессом и интерфейсные программные модули)

3. Обеспечение целостности больших информационных систем требует поддержания моделей системы в актуальном состоянии на всех этапах разработки
4. Формирование архитектуры системы как логической модели информационной системы, объединяющей модели бизнес-процессов, информационной модели и модели организационной структуры

#### **1.4. 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

### **1.5. Понятие «Спецификации индивидуального плана», структура, алгоритм создания документа рабочей программы**

Индивидуальный план является нормативным документом, устанавливающим требования, порядок и правила составления индивидуального плана работы лиц профессорско-преподавательского состава вуза, занимающих штатные должности профессоров, доцентов, старших преподавателей, преподавателей и ассистентов кафедр на полных и неполных ставках. Стандарт устанавливает нормы времени для расчета учебной работы кафедр и преподавателя, примерные нормы времени для расчета трудоемкости учебнометодических, научно-исследовательских и научно-методических, организационно-методических и других видов работ.

Индивидуальный план работы преподавателя охватывает все виды поручений, выполняемых преподавателем в учебном году, и является основным документом, конкретизирующим должностные обязанности преподавателя из раздела «Права и обязанности работника» трудового договора, заключенного между преподавателем и университетом.

Он ежегодно составляется на предстоящий учебный год и оформляется на типовом бланке установленного образца (Форма ИП представлена в Приложение 1).

Индивидуальный план преподавателя является основным документом, регламентирующим планирование и выполнение:

1. Аудиторная нагрузка
2. Учебно-методической работы
3. Научно-исследовательской работы
4. Учебно-воспитательной и организационно-методической работы
5. Сводная таблица за год

Объем всех поручений преподавателя исчисляется в часах.

### **Аудиторная нагрузка (Контактная работа, 1-ая половина дня)**

Учебная работа преподавателя фиксируется в Разделе I «Учебная работа» индивидуального плана работы преподавателя в соответствии с распределением учебной нагрузки между преподавателями кафедры в новом учебном году.

В объем учебной работы профессорско-преподавательского состава входит: чтение лекций, проведение лабораторных, практических и семинарских занятий, тематических дискуссий и деловых игр, прием зачетов, экзаменов, проведение консультаций студентов и аспирантов, рецензирование контрольных работ студентов-заочников, текущий контроль успеваемости студентов очной и очно-заочной форм обучения, руководство курсовыми работами, руководство выпускными квалификационными (дипломными) работами, учебной, производственной и другими видами практик, рецензирование выпускных квалификационных (дипломных) работ, работа в государственных экзаменационных комиссиях и др.

### **Учебно-методической работы (2-ая половина дня)**

К видам учебно-методических работ, относятся:

#### **1) Подготовка учебного издания:**

- Авторского нового издания
- Авторского переиздания с доработкой
- Составительского нового издания
- Составительского переиздания с доработкой

#### **2) Подготовка электронного учебного издания:**

- Разработка электронного учебного издания
- Разработка электронного учебного издания с использованием возможностей компьютерных технологий: гипертекст, мультимедиа, видеоролики, аудио файлы и т.д.)



Учебно-методическая работа преподавателя фиксируется в разделе II индивидуального плана работы. Преподаватель обязан обеспечить соответствующей учебно-методической документацией все виды работ.

Особое внимание в процессе планирования учебно-методической работы преподавателя необходимо уделять разработке методик организации и контроля самостоятельной работы студентов, контрольно-измерительных материалов по выявлению степени освоения компетенций студентами с целью повышения качества предоставляемых образовательных услуг и повышения удовлетворённости потребителей.

Учебно-методическая работа включает в себя:

- Подготовка к аудиторным занятиям
- Подготовка к изданию конспектов лекций, сборников заданий для практических и лабораторных занятий, программ и других учебнометодических разработок, используемых в учебном процессе, включая методические указания по выполнению курсовых, выпускных квалификационных (дипломных, магистерских диссертаций) работ
- Подготовка раздаточного материала для лекционных и практических занятий, наглядных пособий, видеозаписей, интернет лекций и других учебно-методических материалов
- Работа по текущей аттестации и консультированию студентов по выполнению заданий для самостоятельной работы (проверка и анализ контрольных, самостоятельных работ, рефератов, эссе, тестов, практических задач и других видов контрольно-измерительных материалов, предусмотренных программами учебных дисциплин и учебнометодическими комплексами)
- Составление учебных планов по направлениям подготовки (специальностям)

- Составление новых и актуализация изданных рабочих программ учебных дисциплин, практик, государственной итоговой аттестации
- разработка новых и модернизация действующих лабораторных работ с
- Учётом научных разработок, внедрения инноваций и развития материально-технической базы «Гуманитарно-социального института»
- Работы, связанные с применением информационно-телекоммуникационных технологий в учебном процессе (разработка программ, задач, тестов и др.)
- Другие виды учебно-методических работ по заданию кафедры

### **Научно-исследовательской работы**

В разделе III «Научно-исследовательская работа» индивидуального плана работы преподавателя учитываются следующие виды нагрузки:

- Выполнение научно-исследовательских работ
- Написание и подготовка к изданию учебников и учебных пособий, монографий, брошюр, научных статей
- Рецензирование и редактирование учебников, учебных пособий, монографий, научных статей, докладов
- Руководство научными докладами на конференцию, конкурс
- Руководство кафедральными научными студенческими кружками
- Оказание помощи библиотеке
- Работа в редколлегии научных журналов
- Прочие виды научных работ по заданию кафедры

### **Учебно-воспитательной и организационно-методической работы**

Методическая служба образовательного учреждения формировалась с целью повышения уровня профессионального мастерства педагогов. На

протяжении почти столетия, в ситуации стабильного функционирования системы образования это служило основной целью работы методической службы. Однако сегодня, в ситуации перехода образования на качественно иной уровень, не менее важной целью становится научно-методическое обеспечение условий работы педагогов и всего образовательного учреждения в режиме развития.

**Основные задачи методической службы образовательного учреждения:**

- Обучение и развитие педагогических кадров, повышение их квалификации
- Выявление педагогических традиций в данном образовательном учреждении
- Обеспечение высокого методического уровня проведения всех видов занятий
- Подготовка методического обеспечения для осуществления образовательного процесса
- Анализ качества работы педагогов в режиме функционирования
- Приведение методического обеспечения учебных предметов в соответствие с методическими требованиями, предъявляемым к документам в области образования, учебным планам и программам
- Помощь в профессиональном становлении молодых (начинающих) педагогов
- Организация педагогических поисков новых технологий, форм и методов обучения
- Выявление, изучение, обобщение и распространение передового педагогического опыта
- Повышение качества проведения учебных занятий на основе внедрения новых педагогических технологий

- Анализ, апробация и внедрение нового методического обеспечения образовательного процесса
- Разработка учебных, научно-методических и дидактических материалов
- Организация взаимодействия с другими учебными заведениями, научно-исследовательскими учреждениями с целью обмена опытом и передовыми технологиями в области образования и др.

Как и всякий анализ деятельности, анализ методической работы образовательного учреждения является одной из самых трудных задач в деятельности администрации. Но поскольку анализ - это одна из последних стадий осуществления любого процесса, то он проходит наиболее успешно, если остальные стадии выполняются тщательно и грамотно.

Так успешное планирование методической работы во многом определяет глубину ее анализа. Немалую роль для последующего анализа играет привычка регулярного документирования методических мероприятий (не нужно вспоминать, что было сделано или проведено).

Большое подспорье для итогового анализа всей методической работы образовательного учреждения оказывает анализ конкретных мероприятий по мере их осуществления. Ведь все равно после проведенного мероприятия оно более или менее активно обсуждается в коллективе. Нужно только умело направить такое обсуждение и зафиксировать его результаты, которые потом войдут в общий анализ.

Одной из самых больших ошибок при анализе методической работы является его принципиально хвалебный тон. Вместо четкого, ясного критического анализа деятельности получается ее презентация, в которой то, что «получилось» выставляется на передний план, а то, что не получилось, скрывается. Такой анализ приносит большой вред коллективу образовательного учреждения, не позволяя ему совершенствоваться.

Большим недостатком анализа является его превращение в отчет по учебно-воспитательной работе. Методическая работа, провидимая в образовательном учреждении, должна быть в конечном итоге нацелена на повышение качества учебно-воспитательного процесса. Однако зависимость между ними не прямая. Так, низкое качество образования в образовательном учреждении почти всегда свидетельствует о невысоком качестве методической работы. Однако повышение качества образования не может прямо свидетельствовать об улучшении работы методических служб. Ведь на качество образования в данном учреждении, кроме методической работы, влияет еще множество факторов (внешние условия, материально-техническое обеспечение, качество управления и т. д.).

Здесь необходимо помнить, что методическая работа в образовательном учреждении связана, прежде всего, с организацией деятельности педагогического состава.

Некоторые интересные формы методической работы, направленные, например, на расширение кругозора учителей, оказывают опосредованное и часто отсроченное влияние на учащихся. Другие формы могут ставить своей задачей формирование педагогического коллектива, его сплочение, что опять-таки сказывается на качестве ученического образования лишь опосредованно.

#### **Сводная таблица за год**

Сводные данные о выполнении учебной нагрузки кафедры за полугодие и на конец учебного года дважды в год подаются в Учебное управление института. Отчет о выполнении индивидуальных планов за учебный год по установленной форме представляется деканами факультетов. В отчетах отдельными строками указывается объем учебной нагрузки (по всем формам обучения), который был запланирован кафедрам и факультету в целом на учебный год.

## **Глава 2. Разработка расширенной постановки задачи**

### **2.1. Анализ проблематики проектирования АСУ**

Анализ предметной области проводится с целью получения необходимой информации о процессах и объектах, относящихся к сфере действия информационной системы. Сферу действия информационной системы можно представить в виде области, границы которой отделяют процессы и объекты, относящиеся к информационной системе, от тех процессов и объектов предметной области, которые в данной информационной системе не будут представлены.

Автоматизированная система управления (сокращённо АСУ) - комплекс аппаратных и программных средств, а также персонала, предназначенный для управления различными процессами в рамках технологического процесса.

Разработка архитектуры системы же основывается на построении архитектуры системы, которая является определением автоматизируемых функций. Для этого для каждой функции ОМБП (Обобщенная модель бизнес-процессов) необходимо определить целесообразность ее автоматизации. Для каждой автоматизируемой функции необходимо определить степень автоматизации – будет ли функция автоматизирована полностью или функция будет разделена на автоматизируемую часть и часть, которая будет выполняться без средств автоматизации. В последнем случае необходимо преобразовать ОМБП для выделения функций, которые должны быть полностью автоматизированы. Следующим шагом является формирование программных комплексов на основе выделенных операций. Эта процедура производится на основе организационной структуры, в рамках которой функционирует ИС.

Далее идет разделение программных модулей по типам (универсальные программные модули, программные модули слоя правил бизнеса, программные модули слоя документов, программные модули управления бизнес-процессом и интерфейсные программные модули) производится на этапе разработки программной системы.

Завершает проектирование отдельной подсистемы, которая в свою очередь представляет собой процесс определения детального состава программных модулей подсистемы и уточнение таблиц баз данных, с которыми будут работать эти программные модули. Основной задачей проектирования отдельной подсистемы ИС является детальная проработка модели бизнес-процессов и информационной модели для отдельной подсистемы ИС.

Обеспечение целостности больших информационных систем требует поддержания моделей системы в актуальном состоянии на всех этапах разработки. Предлагается разделить проектирование информационных систем на два этапа: этап общего проектирования и этап проектирования отдельных подсистем.

На этапе общего проектирования формируется архитектура системы как логическая модель информационной системы, объединяющая модель бизнес-процессов, информационную модель и модель организационной структуры.

На этапе детального проектирования отдельных подсистем происходит уточнение архитектуры системы, что обеспечивает поддержание моделей системы в актуальном состоянии.

## **2.2. Анализ основных видов проектных решений и инструментальных средств поддержки их формирования**

Модель автоматизированного рабочего места можно разделить на 2 части: обеспечивающую и функциональную.

Отличительные особенности функций деятельности пользователя, которые необходимо автоматизировать, сущность определенного АРМ (Автоматизированное рабочее место) и представление совокупности взаимной связи задач определяет функциональная часть. Для разработки функционального обеспечения необходимо знание входных и выходных данных, пользовательских требований к АРМ, а также средств и методов достижения достоверности и качества информации. Это может включать в себя способы защиты от несанкционированного доступа, систему авторизации

и доступа для пользователя, логгирование изменений, а также возможность сохранения состояния системы в чрезвычайных ситуациях (сбой, нестандартные ситуации).

Обеспечивающая часть включает традиционные виды обеспечения:

- Информационное (описание организации, базы данных, структур связей подсистем)
- Программное (подразделяется на общее – операционные системы, вспомогательные программы т.п., и функциональное – универсальные программы и пакеты)
- Техническое (совокупность технических средств обработки информации)
- Технологическое (программное обеспечение, предназначенное для организации технологического процесса)
- Лингвистическое (программное обеспечение для организации совместных условий работы специалистов)

Основным модулем программного продукта является автоматизированное рабочее место преподавателя, которое выполняет задачу по созданию «рабочих программ» по закрепленному за ним предмету. Разработка базовых проектных решений системы поддержки формирования индивидуальных планов преподавателей

### **2.3. Анализ методов и средств создания информационного обеспечения САПР**

В рамках жизненного цикла промышленных изделий, САПР решает задачи автоматизации стадий проектирования и подготовки производства.

Основная цель создания САПР — повышение эффективности труда, включая в себя такие пункты как:

1. Сокращения трудоёмкости проектирования и планирования
2. Сокращения себестоимости проектирования и изготовления, уменьшение затрат на эксплуатацию



3. повышения качества и технико-экономического уровня результатов проектирования;

Достижение целей создания САПР обеспечивается путем

1. Автоматизации оформления документации
2. Информационной поддержки и автоматизации принятия решений
3. Использования технологий параллельного проектирования
4. Унификации проектных решений и процессов проектирования
5. Повторного использования проектных решений, данных и наработок


## **2.4. 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.

	<p>NestJS</p> <p>Серверная часть приложения</p>
---	---

	Angular9 Клиентская часть приложения
	TypeScript Язык со строгой типизацией Система для работы с модулями и классами
	JavaScript Мультипарадигменный скриптовый язык программирования
	PostgreSQL Система управления базами данных
	Graph QL Облегчает агрегацию данных из нескольких источников
	Git Система контроля версий
	jExcel Легкий плагин JavaScript для создания интерактивных веб-таблицы и электронных таблицы
	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 разворачивать 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. Можно настроить проект таким образом, что любой некомпilierуемый код (фактически, код с синтаксической ошибкой) нельзя будет закоммитить. Случаи, в которых разработчик позволяет себе закоммитить сломанный код в обход проверок перед коммитом не рассматриваем

5. 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) - это платформа для построения эффективного, масштабируемого узла.JS-серверные приложения. Он использует прогрессивный JavaScript, построен с и полностью поддерживает TypeScript (но все же позволяет разработчикам кодировать в чистом JavaScript) и сочетает в себе элементы

ООП (объектно-ориентированное программирование), FP (функциональное программирование) и FRP (функциональное реактивное программирование).

Под капотом Nest использует надежные http-серверные фреймворки, такие как Express (по умолчанию), это одно из причин использования Nest взамен всем известного использования 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 уровень абстракции выше этих общих узлов. JS Framework (Express/Fastify), но также предоставляет свои API-интерфейсы непосредственно разработчику. Это позволяет разработчикам свободно использовать множество сторонних модулей, доступных для базовой платформы.

В последние годы, благодаря узлу. JS, JavaScript стал “lingua franca” в интернете как для фронтальных, так и для бэкенд-приложений. Это дало начало таким потрясающим проектам, как Angular, React и Vue, которые повышают производительность разработчиков и позволяют создавать быстрые, тестируемые и расширяемые интерфейсные приложения. Однако, Хотя существует множество превосходных библиотек, помощников и инструментов для Node (и серверного JavaScript), ни одна из них не решает эффективно главную проблему-архитектуру.

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

## 2.5. 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. Она работает практически на всех UNIX-платформах, включая UNIX-подобные системы, такие как FreeBSD и Linux. Ее можно применять на Windows NT Server и Windows 2000 Server, а для разработки годятся даже такие системы Microsoft для рабочих станций, как ME. 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.

Для сохранения шаблонов в процессе работы и поддержки версии, системе потребуется база данных.

Различают несколько видов баз данных:

- Иерархическая
- Объектная и объектно-ориентированная
- Объектно-реляционная
- Реляционная
- Сетевая
- Функциональная

Для реализации системы шаблонов рабочих программ необходимо реализовать следующие функциональные требования и решить следующие задачи:

- Система шаблонов рабочих программ должна предоставлять возможность гибко варьировать включение блока в итоговый шаблон

- Должна предоставить возможность увеличения количества строк в выходных таблицах, возможность дополнения дополнительными данными в процессе работы с шаблоном
- Целостность данных – безопасность от перезаписи уже существующих шаблонов
- Версионность
- Разграничение прав доступа к данным

### **Рассмотрим несколько баз данных:**

**MySQL** – это самая распространенная полноценная серверная СУБД. MySQL очень функциональная, свободно распространяемая СУБД, которая успешно работает с различными сайтами и веб приложениями. Обучиться использованию этой СУБД довольно просто, так как на просторах интернета вы легко найдете большее количество информации.

**Заметка:** стоит заметить, что благодаря популярности этой СУБД, существует огромное количество различных плагинов и расширений, облегчающих работу с системой.

Несмотря на то, что в ней не реализован весь SQL функционал, MySQL предлагает довольно много инструментов для разработки приложений. Так как это серверная СУБД, приложения для доступа к данным, в отличие от SQLite работают со службами MySQL.

### **Преимущества MySQL:**

- Простота в работе - установить MySQL довольно просто. Дополнительные приложения, например, GUI, позволяет довольно легко работать с БД
- Богатый функционал - MySQL поддерживает большинство функционала SQL
- Безопасность - большое количество функций, обеспечивающих безопасность, которые поддерживается по умолчанию

- Масштабируемость - MySQL легко работает с большими объемами данных и легко масштабируется
- Скорость - упрощение некоторых стандартов позволяет MySQL значительно увеличить производительность

### **Недостатки MySQL:**

- Известные ограничения - по задумке в MySQL заложены некоторые ограничения функционала, которые иногда необходимы в особо требовательных приложениях
- Проблемы с надежностью - из-за некоторых способов обработки данных MySQL (связи, транзакции, аудиты) иногда уступает другим СУБД по надежности
- Медленная разработка - Хотя MySQL технически открытое ПО, существуют жалобы на процесс разработки. Стоит заметить, что существуют другие довольно успешные СУБД, созданные на базе MySQL

### **PostgreSQL**

PostgreSQL свободно распространяемая и максимально соответствующая стандартам SQL. В PostgreSQL или Postgres полностью применяются и обновляются ANSI/ISO SQL стандарты своевременно с выходом новых версий.

От других СУБД PostgreSQL отличается поддержкой востребованного объектно-ориентированного и/или реляционного подхода к базам данных. Полная поддержка надежных транзакций, т.е. атомарность, последовательность, изоляционность, прочность (Atomicity, Consistency, Isolation, Durability (ACID).) Благодаря мощным технологиям Postgres очень производительна. Параллельность достигнута не за счет блокировки операций чтения, а благодаря реализации управления многовариантным параллелизмом (MVCC), что также обеспечивает соответствие ACID. PostgreSQL имеет возможность расширение доступных процедур, которые называются



хранимые процедуры. Эти функции упрощают использование постоянно повторяемых операций.

### **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 – PostgreSQL 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)

Стоит сказать, что Postgres рекомендуется, если ваши будущие приложения имеют сложную схему, которая требует большого количества соединений или все данные имеют отношения, или если у нас есть тяжелая

запись. Postgres - это открытый исходный код, более быстрый, совместимый с ACID и использующий меньше памяти на диске, а также все вокруг хорошо работает для хранения JSON и включает в себя полную сериализуемость транзакций с 3 уровнями изоляции транзакций.

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. Базовый движок очень стабилен и хорошо справляется с большим диапазоном объемов данных. Он также работает на вашем выборе аппаратного обеспечения и операционной системы. Postgres предоставляет возможности NoSQL наряду с полной поддержкой транзакций, храня документы JSON с ограничениями на данные полей.

#### **Общие ограничения для Postgres:**

1. Масштабирование Postgres по горизонтали значительно сложнее, но выполнимо
2. Быстрое чтение операций не может быть полностью достигнуто с помощью Postgres

В течении разработки ВКР было добавлено к задачам с БД, изучение и возможно использование такой БД как **MongoDB** или парное использование с **Postgres**. Идея использования **MongoDB** возникло в то время, когда производилось изучение и анализ сервиса по работе с таблицами на стороне клиента (front-end), так как таблица представляла собой JSON объект.

MongoDB может превзойти Postgres в измерении "горизонтального масштаба". Хранение - JSON-это то, для чего Mongo оптимизирован. Mongo хранит свои данные в двоичном формате, называемом BSONb, который является (грубо говоря) просто двоичным представлением суперпозиции JSON. MongoDB хранит объекты точно так, как они были спроектированы. По данным MongoDB, для приложений с интенсивной записью Монго говорит,

что новый движок (Wired Tiger) дает пользователям до 10-кратного увеличения производительности записи (я должен попробовать это), с 80-процентным снижением использования памяти, помогая снизить затраты на хранение, добиться большего использования аппаратного обеспечения.

### **Общие ограничения MongoDB:**

1. Использование механизма хранения без схемы приводит к проблеме неявных схем. Эти схемы не определяются нашим механизмом хранения данных, а определяются на основе поведения и ожиданий приложений
2. Автономные технологии NoSQL не соответствуют стандартам ACID, поскольку они жертвуют критической защитой данных в пользу высокой пропускной способности для неструктурированных приложений. Нетрудно применить кислоту к базам данных NoSQL, но это сделает базу данных медленной и негибкой до некоторой степени. “Большинство ограничений NoSQL были оптимизированы в более новых версиях и выпусках, которые в значительной степени преодолели его предыдущие ограничения”

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 GraphQL 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
  }
}
```

## 2.6. Требования к документированию и оформлению исходного кода

В состав документации к подсистеме должны быть включены следующие описания:

- Необходимо описание основных структурных методов back-end
- Требуется охарактеризовать основные структурные элементы front-end
- Требуется сущности БД (таблица основных блоков и их компонентов, связи между ними)
- Необходимо описание классов и моделей шаблонов
- Требуются структурные схемы разработанных модулей

- Необходима диаграмма классов
- Требуется единый API для всего приложения, не ограничиваясь конкретным механизмом хранения данных.

## **Принципы написания исходного кода**

### **DRY – Don't repeat yourself. Не повторяйся**

Если вы вставили один и тот же блок в 10 разных регламентов и что-то меняете в нём, то у вас образуется 10 одинаковых мест, куда нужно внести одинаковые изменения. Вероятность забыть об этом тоже растёт, даже если вы пользуетесь бессмертным «Ctrl+C» - «Ctrl+V». Особенно в последней паре документов.

В программировании такие блоки выделяют в отдельные функции, классы, а в работе с документами повторяющиеся элементы нужно выносить в отдельные разделы или документы и давать ссылки на них.

### **KISS – Keep it simple, stupid. Не усложняй**

Посмотрите вокруг: люди привыкли к простоте. Интуитивно понятные интерфейсы и всплывающие подсказки в приложениях. Принцип одного окна в IT-обслуживании. Засилье коротких статей и обучающих видео в Интернете. Той же простоты они ждут и от рабочих документов.

Этот принцип обычно используют как маркер. Если описание процесса слишком сложное - нужно упростить процесс. Если инструкция к приложению слишком замысловатая – надо доработать приложение.

Обычно это проще и дешевле, чем писать сложную документацию, заставлять каждого нового сотрудника её читать, понимать и учиться действовать по ней. А ведь потом ещё нужно будет вносить изменения в документацию.

### **YAGNI – You ain't gonna need it. Тебе это не понадобится**

Это продолжение принципа KISS. Обычно я применяю его, когда собираюсь создать универсальное решение с заделом на будущее. Проблема в том, что такое решение — само по себе усложнение. Не факт, что в будущем

оно понадобится. Обычно нет смысла создавать всеобъемлющее описание бизнес-процессов направления, которое только сформировалось и активно развивается. Практика показывает, что будущее внесёт серьёзные коррективы.

Но здесь важно разумно подходить к вопросу. Есть вещи, изменение которых дорого обходится. Их как раз нужно готовить с заделом на будущее. В программировании эту работу относят к архитектуре приложения, а в документации просто сразу говорят: «Это нам не понадобится». И в будущем с трудом навёрстывают упущенное.

### **Принципы работы со структурой**

Чем объёмнее становится документация, тем сложнее поддерживать её актуальность. Изменение одного документа влечёт за собой каскадное изменение остальных, логическая целостность ломается, структура требует очередной переработки. К счастью, для программирования это стандартная проблема, и за прошедшие десятилетия выработались общие принципы написания легко поддерживаемого кода. Часть из них можно применять и к документам.

**SRP — Single Responsibility Principle. Принцип единственной ответственности**

Яркий маркер нарушения принципа - наличие побочных действий. Например, метод класса отвечает за отправку клиенту уведомления о новом заказе в интернет-магазине, но при этом ещё и обновляет текущий баланс клиента.

То же самое справедливо и для документации. Например, в регламент регистрации обращения клиента не нужно впихивать что-то «заодно»: принципы вежливого общения с клиентами, краткое описание self-service портала и т. д.

Нельзя рассчитывать, что такая информация будет усвоена или сотрудники запомнят, к какому документу нужно обращаться, чтобы её получить. Зато это запутывает документ, лишает структуру чёткости и требует дополнительного времени для актуализации. Если вы действительно хотите

напомнить о вежливости при общении с клиентом, вставьте в регламент ссылку на соответствующий документ (вспомним о DRY).

Соблюдение принципа позволяет делать документы короче. Их название соответствует назначению, а чтение не вызывает неприятных сюрпризов. Их проще прочитать за раз. И не забыть, с чего документ начинался.

### **SLAP — Single Level of Abstraction Principle. Принцип единого уровня абстракции**

Является логичным продолжением принципа единственной ответственности (SRP).

Применение принципов KISS и SRP побуждает писать небольшие, простые и понятные документы. Но эти документы нужно собрать в такую же простую и понятную структуру. Здесь помогает выделение и соблюдение уровней абстракций.

Например, очевидно, что нет смысла пытаться поместить в один огромный документ объёмную документацию вроде описания системы менеджмента качества. Намного логичнее разбить его на уровни абстракций и сделать для каждого уровня разные документы - верхнеуровневое описание системы, описание каждого процесса и т. д.

При этом важно соблюдать принцип единого уровня абстракций и относить информацию только к нужному уровню. Этот принцип дополняет SRP, заставляя автора контролировать не только то, за что отвечает документ, но и на каком уровне в иерархии он находится.

### **LoD — Law of Demeter (Don't talk to strangers). Закон Деметры (Не разговаривай с незнакомцами)**

Если применить закон к документации, то каждый документ:

- Должен обладать ограниченным знанием о других документах, т. е. «знать» только тех, которые имеют к нему непосредственное отношение
- Должен взаимодействовать только с известными ему документами-«друзьями» и не «разговаривать» с незнакомцами

Например, процесс управления IT-инцидентами взаимодействует с процессом управления проблемами напрямую, а с процессом управления изменениями - только через управление проблемами. Тогда в описании управления инцидентами можно ссылаться на описание управления проблемами.

## **2.7. Требования к уровню реализации функций в бакалаврской работе**

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. К основному функционалу относятся:

- 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 Dosh
- 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



## Глава 3. Проектирование приложения

### 3.1. 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:

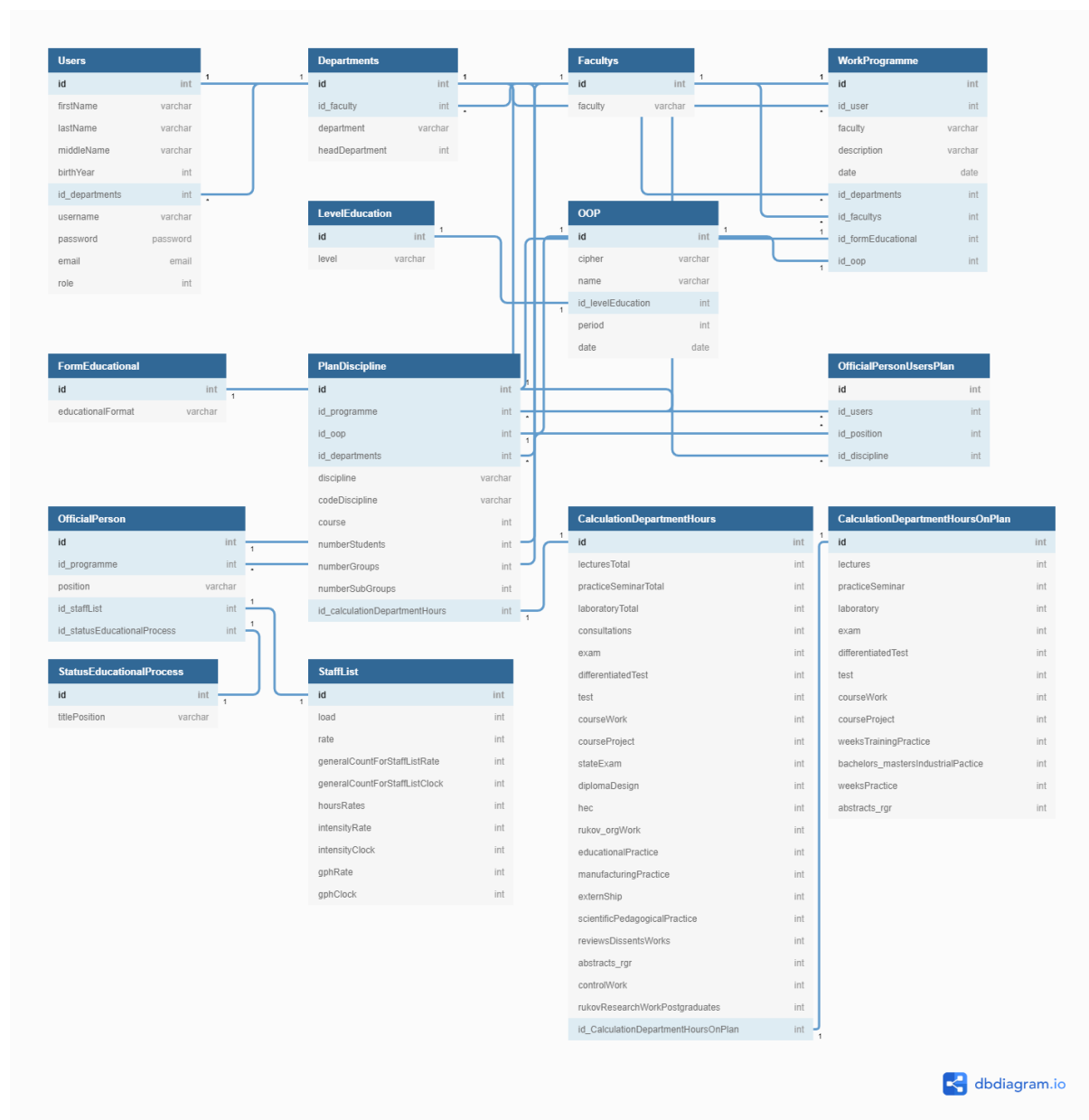


Рисунок 2 – ER диаграмма БД

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.

## 3.2. Разработка структурно-функциональной организации Автоматизация формирования индивидуальных планов

### 3.2.1. Разработка диаграммы компонентов

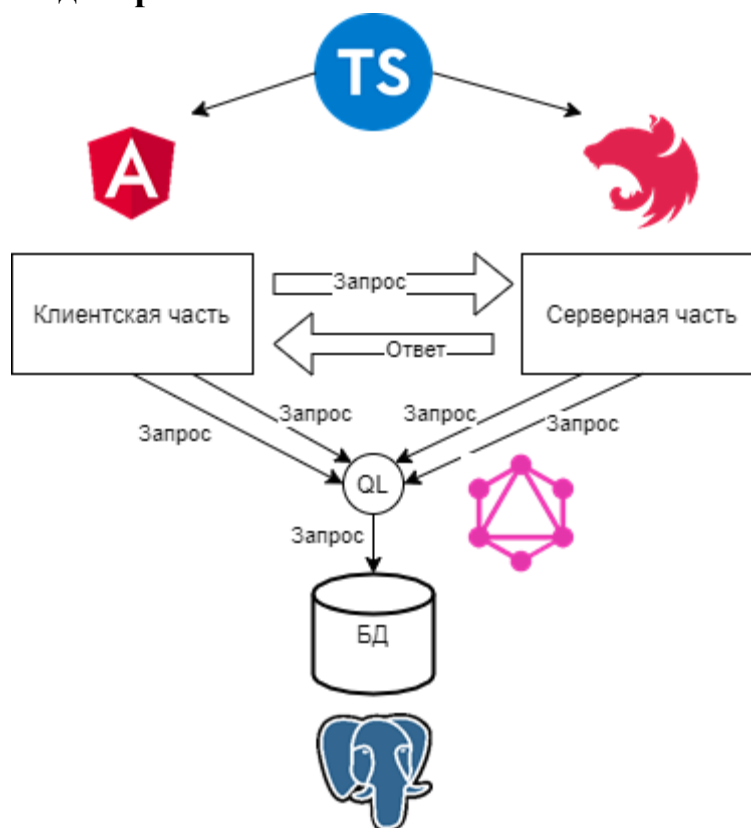


Рисунок 3. Диаграммы компонентов

## Концепция Архитектуры

□ использование в любых системах  
■ использование в текущей системе



Рисунок 4. Концепция автоматизированной системы создания индивидуальных планов преподавателя

### 3.2.2. Разработка диаграммы классов

Серверная часть:

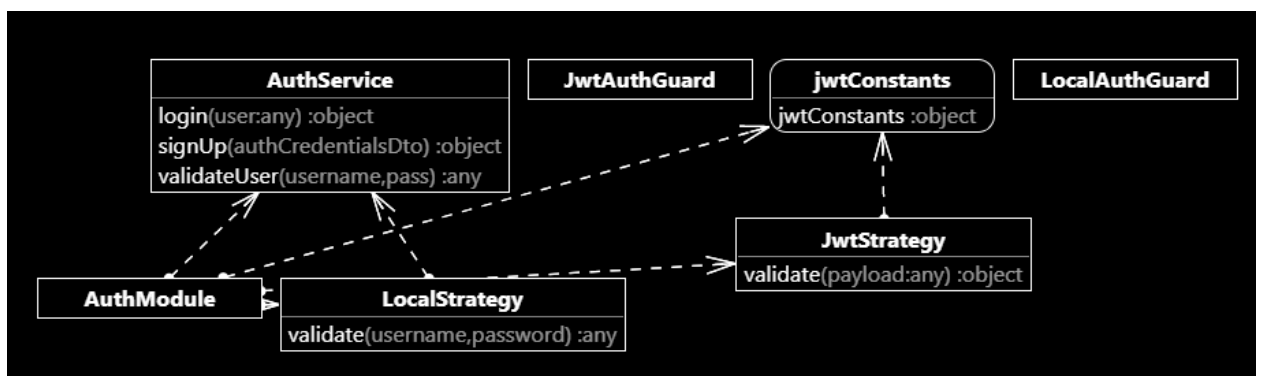


Рисунок 5. Диаграмма класса аутентификация

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.

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

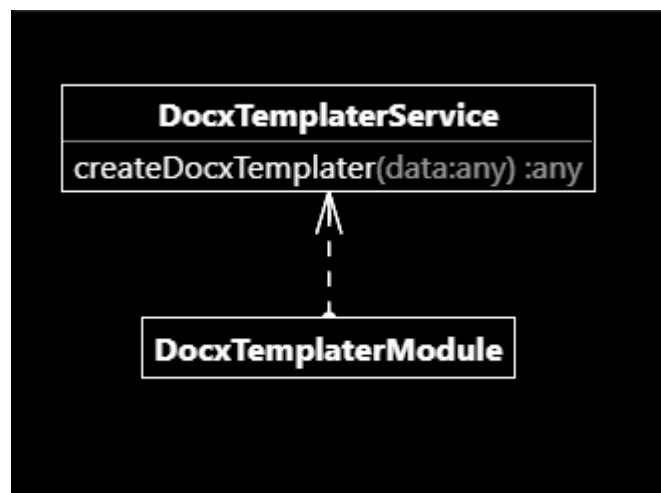


Рисунок 6. Диаграмма класса генератора документа

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

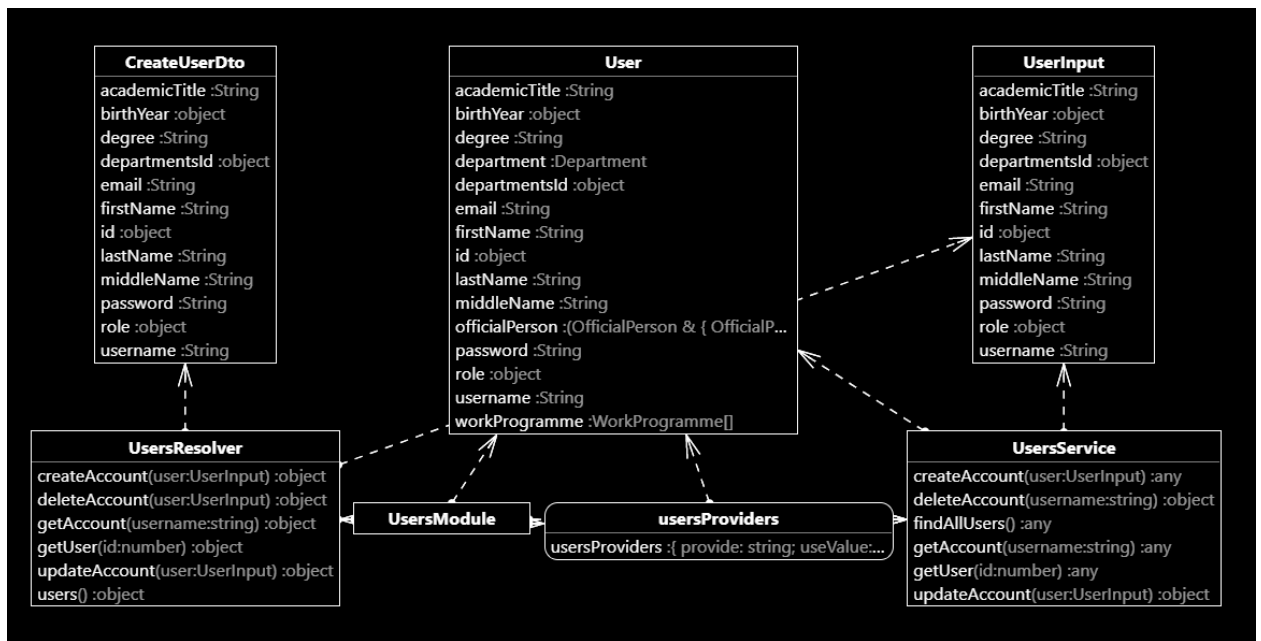


Рисунок 7. Диаграмма класса пользователя

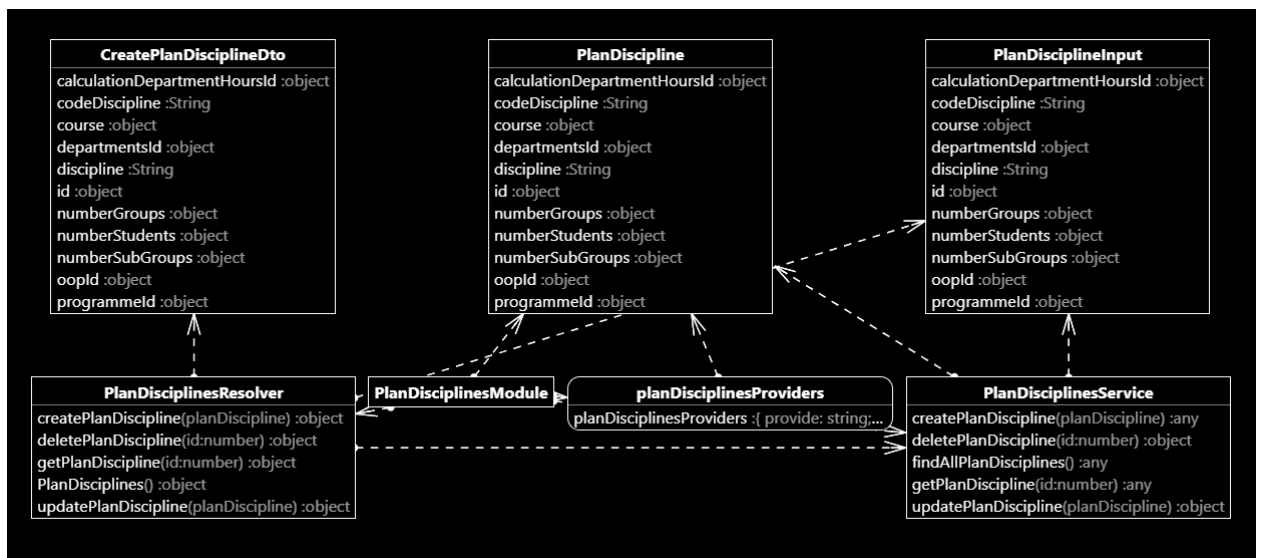


Рисунок 8. Диаграмма класса дисциплины

Появление таких сущностей как Dto и Input, вызвано тем, что для работы с 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.

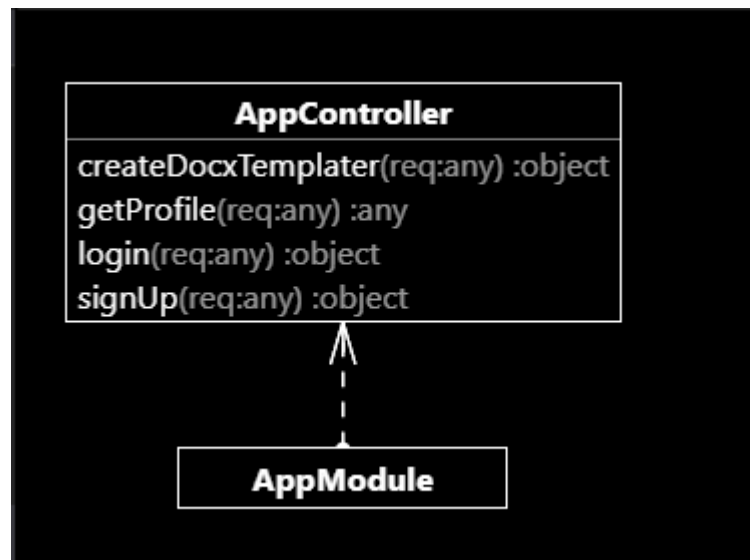


Рисунок 8. Диаграмма класса основного контроллера

#### Клиентская часть:

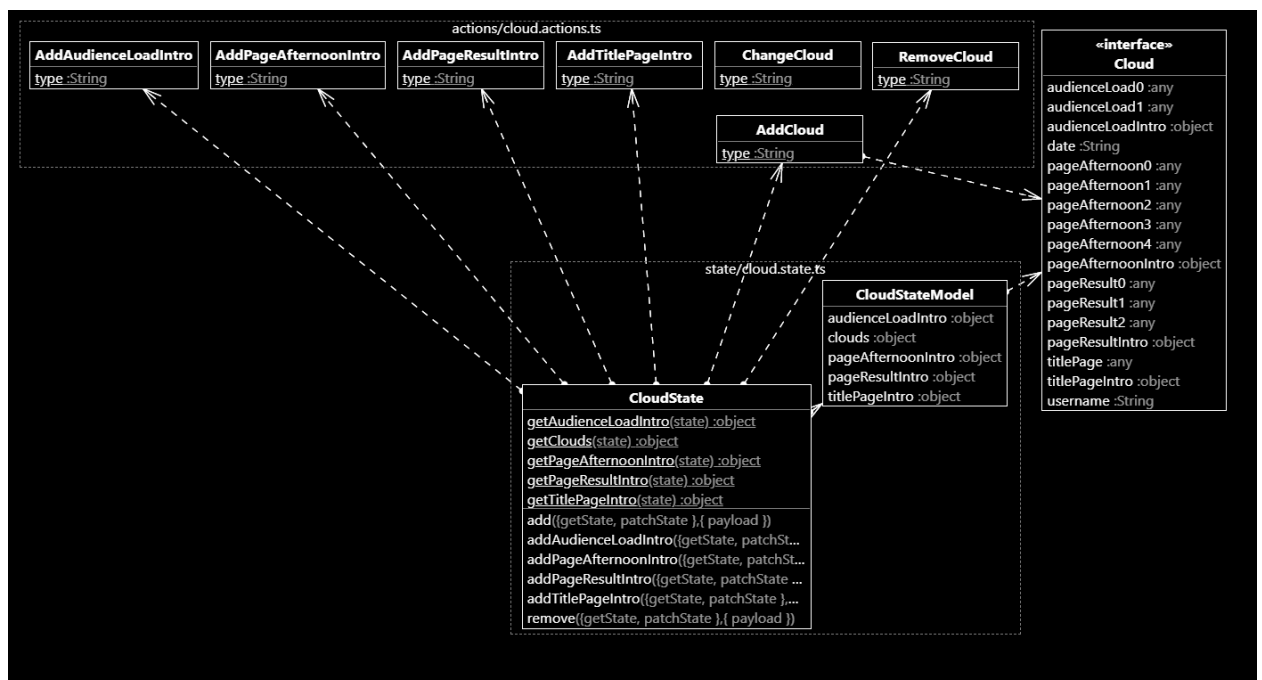


Рисунок 10. Диаграмма класса внутреннего хранилища

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.



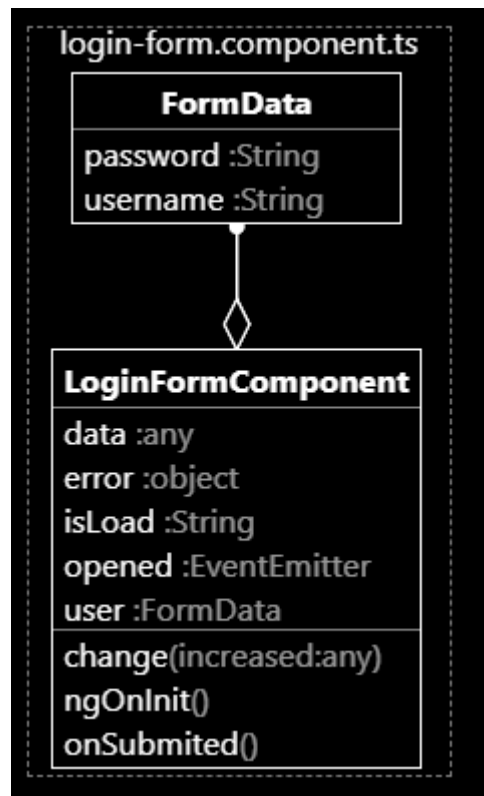


Рисунок 12. Диаграмма класса аутентификации

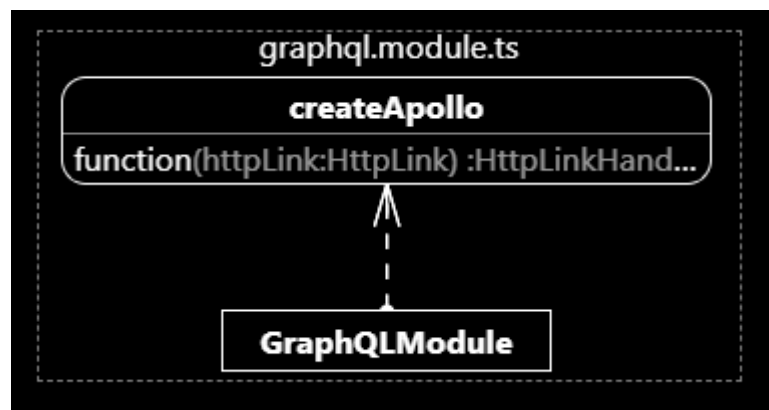


Рисунок 13. Диаграмма класса контроллера Graph QL

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.

### 3.2.3. 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

### 3.2.2. Разработка диаграммы деятельности

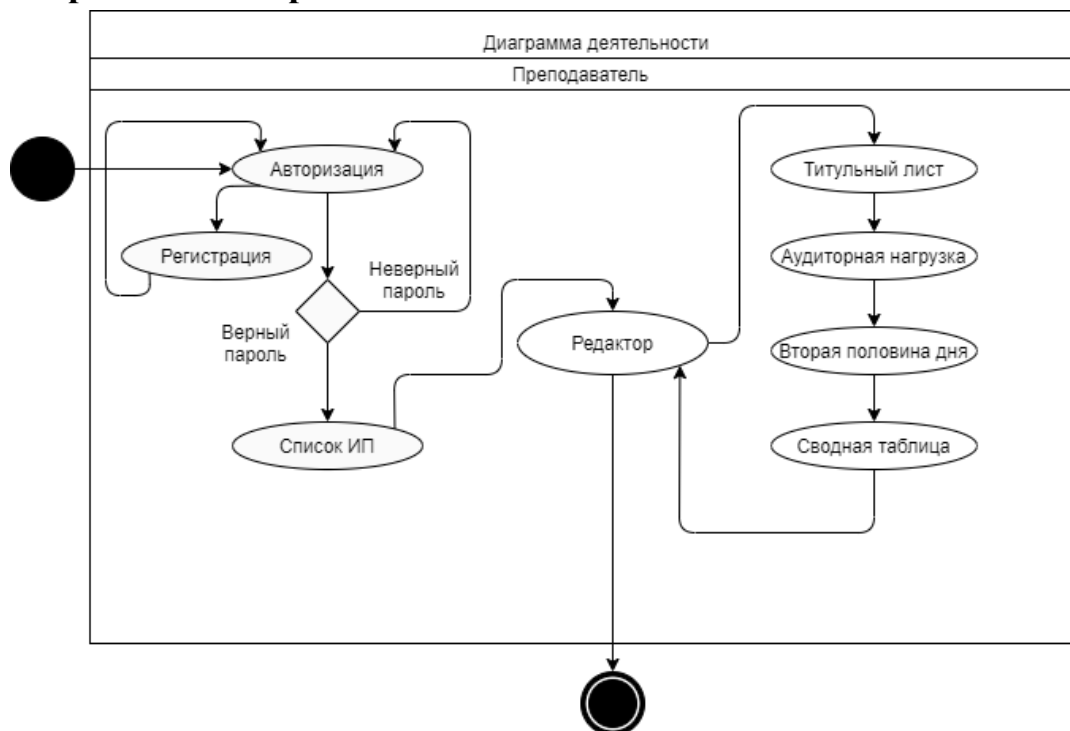


Рисунок 15. Диаграмма деятельности прототипа автоматизированной системы поддержки рабочих программ

## **Глава 4. 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

### **4.1. Описание исходных кодов и процесса их отладки в Back-end**

#### **4.1.1 Создания одной точки входа 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.

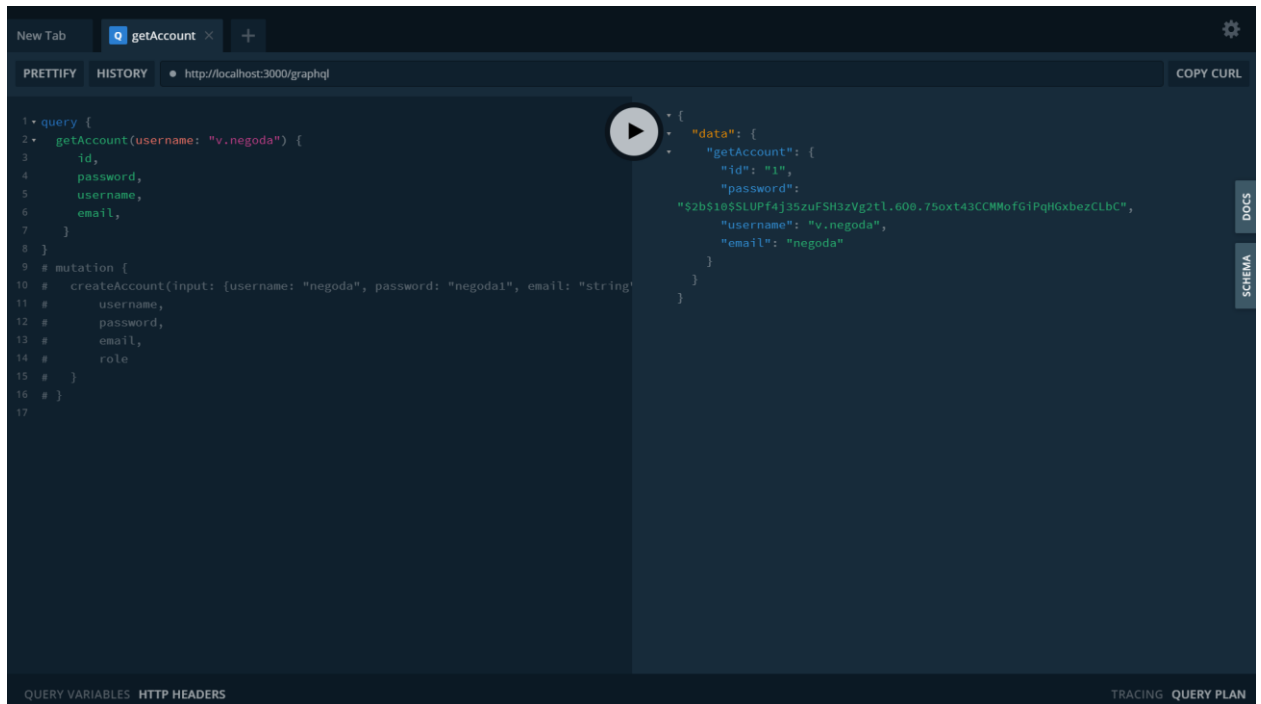


Рисунок 16. Площадка GraphQL

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.

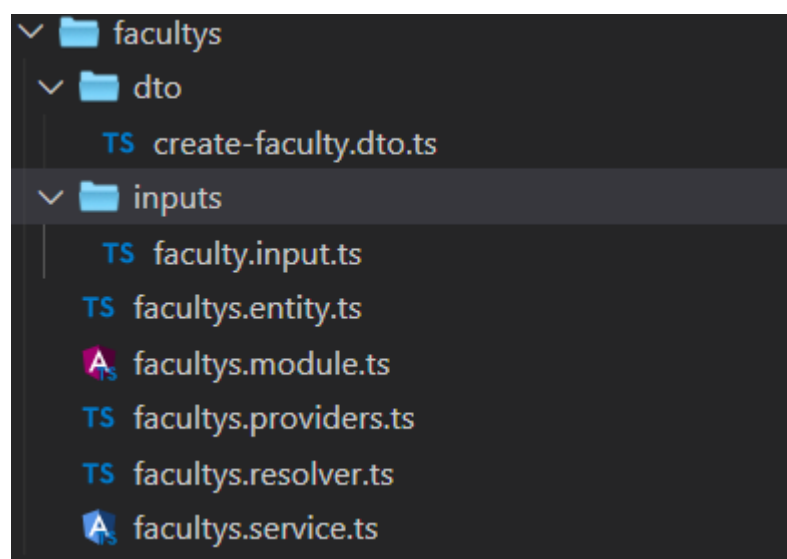


Рисунок 17. Пример файловой системы для QL

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: 'faculty',
};

@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 автоинкрементирования для поля id. Также ниже описываются связи с другими таблицами с помощью тех же декораторов, в которых указывается к какой сущности какие ключи привязан.

```
Executing (default): CREATE TABLE IF NOT EXISTS "planDiscipline" ("id" SERIAL , "programmeId" INTEGER REFERENCES "workProgramme" ("id") ON DELETE CASCADE ON UPDATE CASCADE, "oopId" INTEGER REFERENCES "oop" ("id") ON DELETE CASCADE ON UPDATE CASCADE, "departmentsId" INTEGER REFERENCES "departments" ("id") ON DELETE CASCADE ON UPDATE CASCADE, "discipline" VARCHAR(255), "codeDiscipline" VARCHAR(255), "course" INTEGER, "numberStudents" INTEGER, "numberGroups" INTEGER, "numberSubGroups" INTEGER, "calculationDepartmentHoursId" INTEGER REFERENCES "calculationDepartmentHours" ("id") ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY ("id"));
```

### Рисунок 19. 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 FacultyService {
  constructor(
    @Inject('FACULTYS_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 А, 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 созданию мутаций и запросов для GraphQL.

Начнем с query запросов. При работе с GraphQL, можно не указывать query перед фигурными скобками. При этом, при написании запросов на клиенте, если вы НЕ укажете 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 17, пункт 4.1.1.

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>.

```
type CreateFacultyDto {
  id: ID!
  department: String!
  director: Int!
}

type CreateUserDto {
  id: ID!
  firstName: String!
  lastName: String!
  middleName: String!
  username: String!
  birthYear: Int!
}

type Query {
  users: [CreateUserDto!]!
  getUser(id: Int!): CreateUserDto
  facultys: [CreateFacultyDto!]!
  getFaculty(id: Int!): CreateFacultyDto
}
```

Рисунок 18. Пример схемы QL



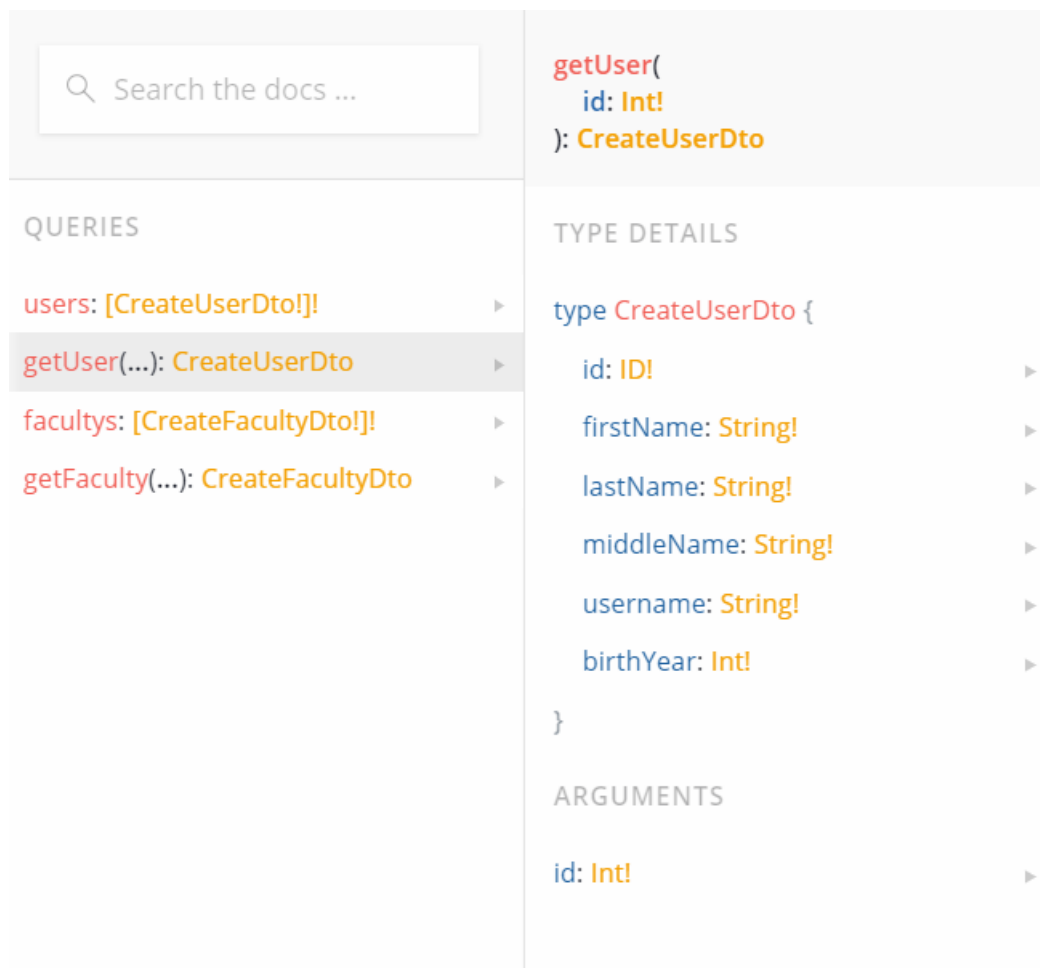


Рисунок 19. Пример запросов QL

#### 4.1.2 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.

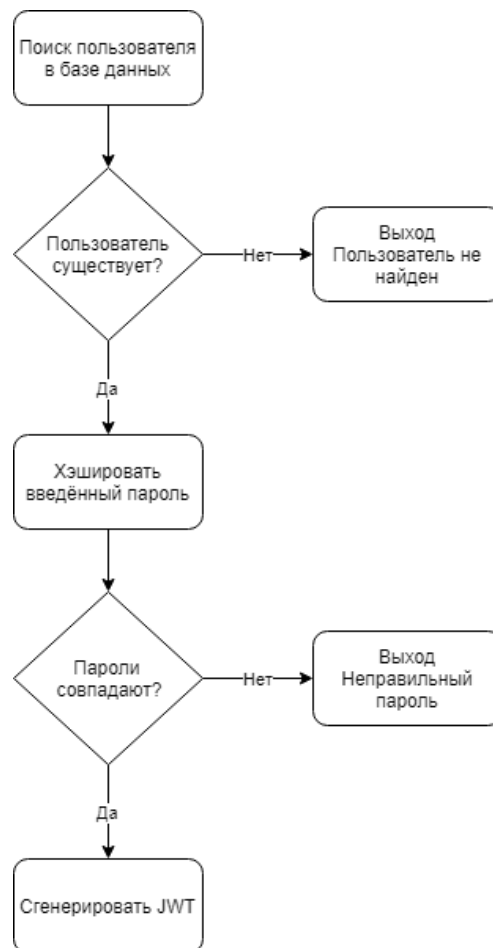


Рисунок 20. Блок схема алгоритма при аутентификации

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 Requestобъекту 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

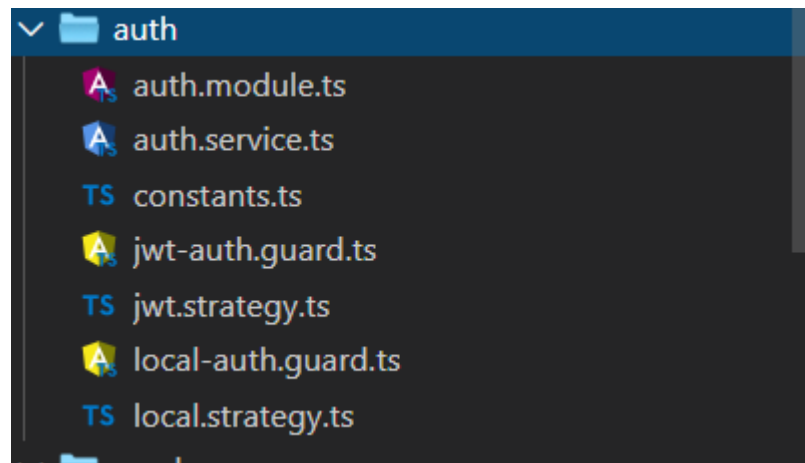


Рисунок 21. Пример файловой системы аутентификации

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 usersвойство (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, обработка и генерация производятся 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.

#### 4.1.3 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
r, 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.

## 4.2. Описание исходных кодов и процесса их отладки в Front-end

### 4.2.1 Модуль аутентификация и посадочной страницы

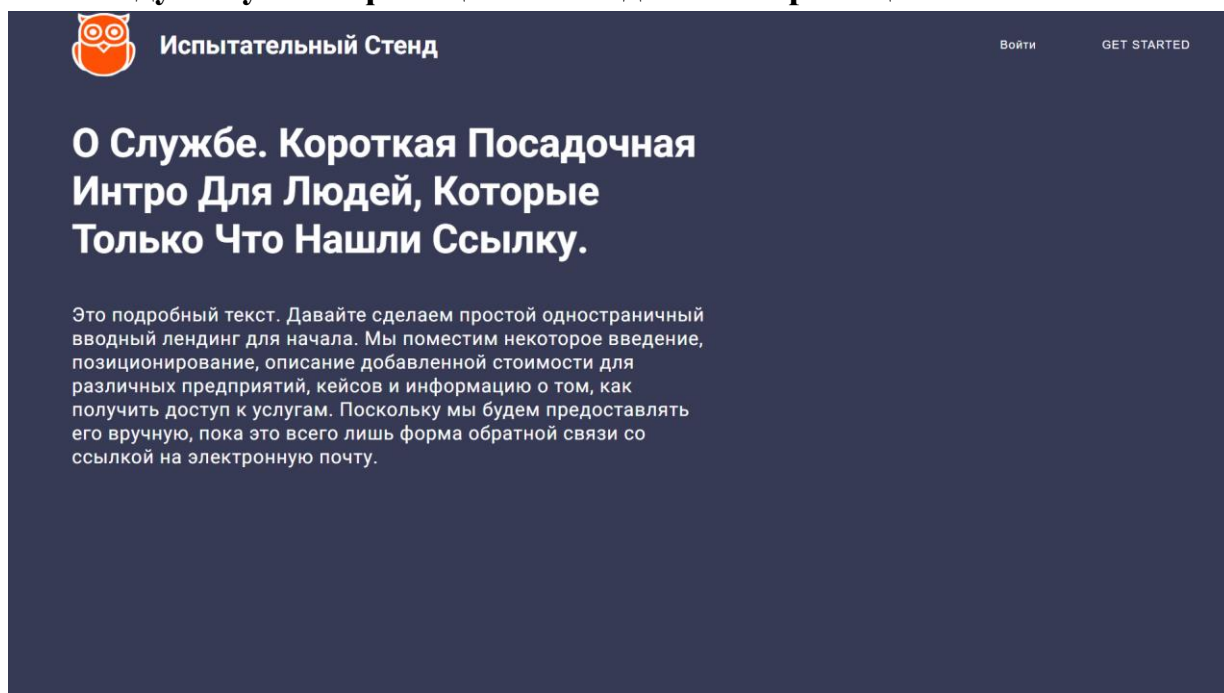


Рисунок 21. Прототип модуля посадочной страницы

← Назад

Форма Входа

a.lylova

.....

Вход ВОССТАНОВЛЕНИЕ ПАРОЛЯ

создать учетную запись Поддержка Помощь

Рисунок 11. Прототип модуля формы аутентификации

На стороне front-end для формы входа был реализован прототип `HttpClient`, для отправки `post` запросов на сервер с целью получения JWT токена и последующей отправки токена для аутентификации.

Пример обработчика:

```
this.isLoading = 'Вход...';

this.http.post('http://localhost:3000/auth/login', this.user, httpOptions).
subscribe(value => {
  const token = value['access_token'];
  if (token) {
    // localStorage.setItem('token', token);
    const httpOption = {
      headers: new HttpHeaders({
        'Content-Type': 'application/json',
        Authorization: `Bearer ${token}`
      })
    };
  }
});
```



```

        this.http.get('http://localhost:3000/profile', httpOption).subscribe(re
s => {
    console.log(res);
    this.isLoad = 'Вход';
    this.error = true;
    this.router.navigate(['/main/page-authorizing']);
  },
  error => {
    console.log(error);
  });
} else {
  this.isLoad = 'Вход';
  this.error = false;
  return;
}
},
error => {
  console.log(error);
  this.isLoad = 'Вход';
  this.error = false;
  return;
});
});

```

После успешной аутентификации происходит редирект пути на модуль отображения шаблонов, в котором можно выбрать шаблон для редактирования и последующего создания файла индивидуального плана преподавателя.

#### 4.2.2 Модуль отображения списка шаблонов пользователя

В рамках данного модуля основополагающим элементом является процесс визуального отображения выбранного из списка шаблона. Это необходимо для того, чтобы минимизировать возможную ошибку выбора не подходящего шаблона и избежать опечаток, для этого был выбран стиль Word документа.

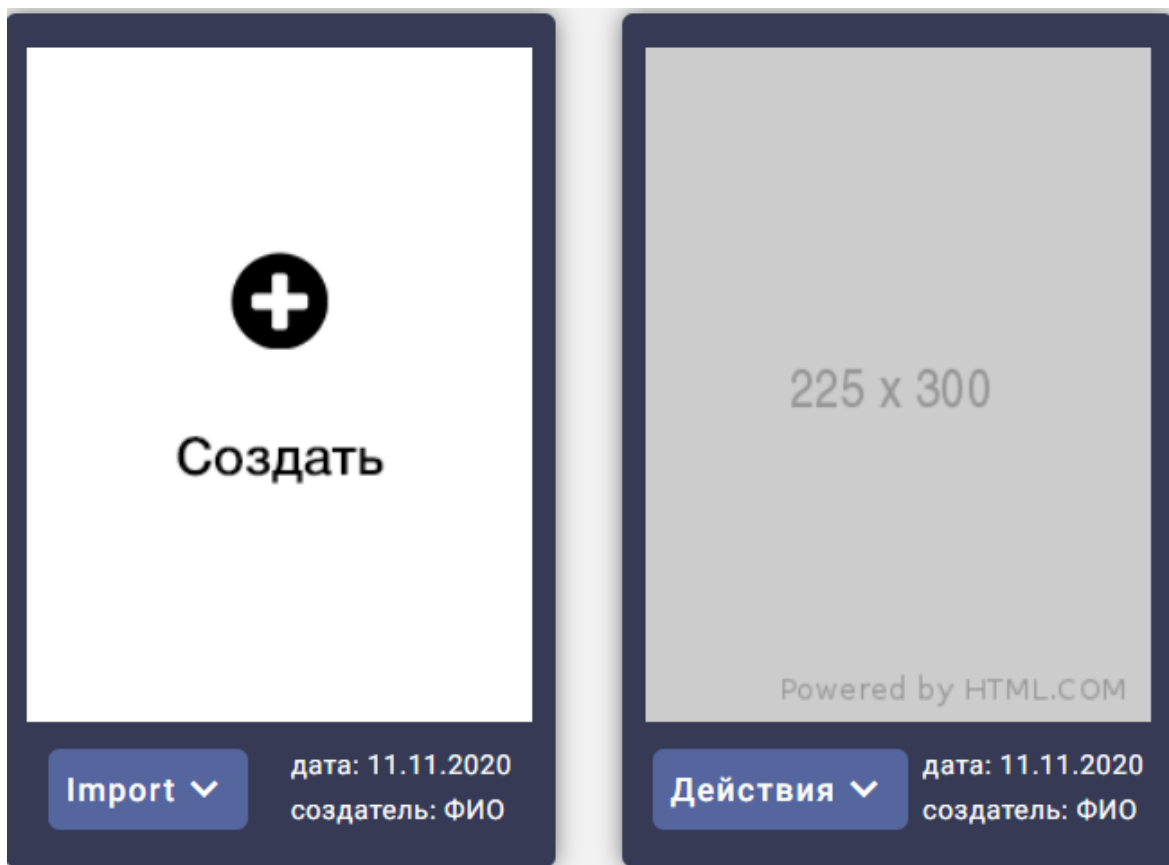


Рисунок 24. Прототип модуля отображения списка шаблонов пользователя

#### 4.2.3 Основной модуль редактора индивидуального плана

Данный модуль разделен на 4 части:

1. Титульный лист
2. Аудиторная нагрузка
3. Вторая половина дня
4. Сводная таблица

Также для каждой из модулей было создано краткое введение для текущего шага редактора, которое появляется только при первом посещении шага редактора.

Пример интро для Второй половины дня:

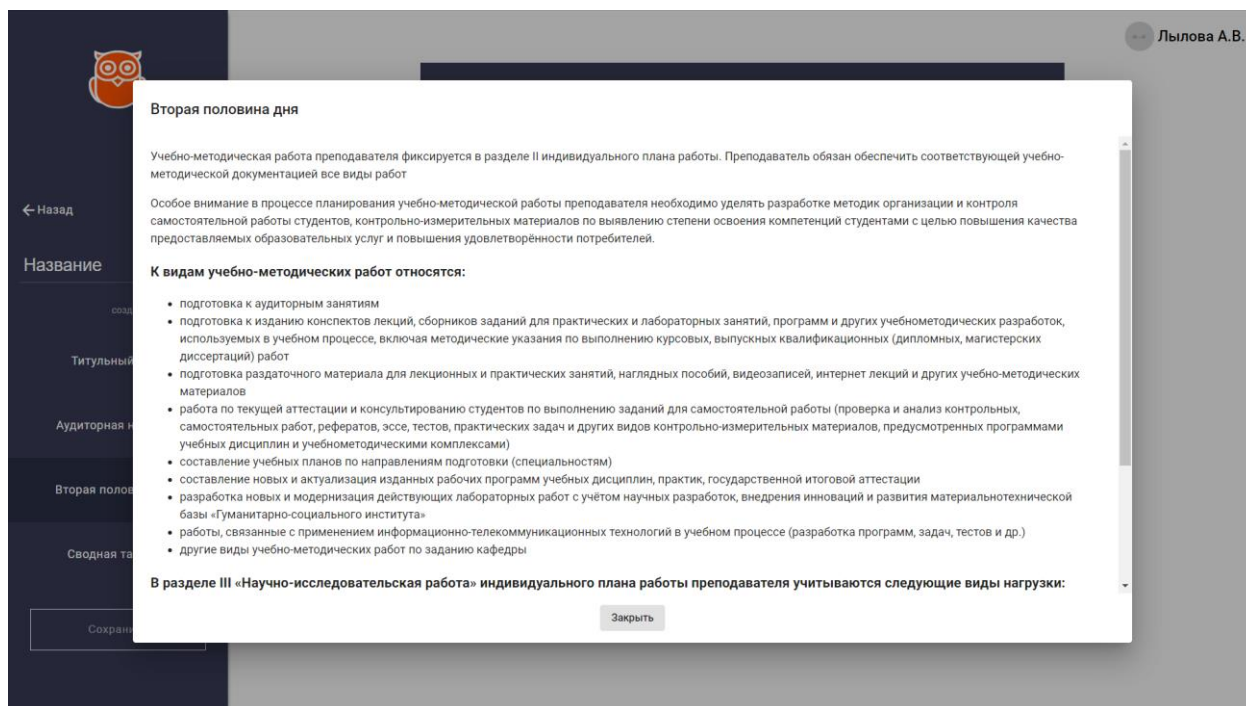


Рисунок 25. Введение в модуль редактора второй половины дня

**Титульный лист Индивидуального Плана Преподавателя (На 1 Год)**

Факультет ФИС	Кафедра Вычислительная техника
Фамилия Анна	Имя Лылова
Отчество Вячеславовна	Должность старший преподаватель
Год рождения 1979	Ученая степень и год присуждения
Ученое звание и год присвоения	

Проверьте правильность заполненных данных.  
В случае если ваши данные изменились, обратитесь в [ITC СОАИГЭКОУ](#).

[Сохранить](#)

Рисунок 26. Модуль редактора титульного листа

Данный титульный лист оформлен в виде простой таблицы, в которой авто-подставляются данные с БД. Преподаватель на данном шаге проверяет правильность заполнения личных данных.

Также данная таблица легко изменяемая и не требует использования лишних строк html кода для создания дополнительных или же удаления полей.

Это происходит из-за использования циклического создания полей, которое зависит от полученного объекта наименований полей.

```
<div class="information">
  <div class="group" *ngFor="let item of dataDto">
    <label for="{{item.title}}">{{item.placeholder}}</label>
    <input type="text" name="{{item.title}}" placeholder="{{item.placeholder}}
  </div>
  </div>
```

Следующие 3 шага, а это первая, вторая половина дня и сводная таблица, основаны на одном пере используемом компоненте по работе с таблицами.

Учебно-методическая работа

Вид работы	норматив времени	Осенний семестр		Весенний семестр	
		кол-во часов по плану	факт. выполнено	кол-во часов по плану	факт. выполнено
1	2	3	4	5	6
2.1. Подготовка к лекционным, практическим, семинарским, лабораторным занятиям и другие виды учебно-методической работы (для программ ВО)					
1	Подготовка к лекциям	до 0,75 на 1 а.ч.	36		72
2	Подготовка к лабораторным занятиям	до 0,25 на 1 а.ч.	12		
3	Подготовка к практическим занятиям	до 0,25 на 1 а.ч.	16		72
4	Подготовка занятий на курсовую работу	до 0,75 на 1 раб-у			28
5	Подготовка презентационных материалов	до 2ч. На 1 а.ч. но не более 100ч.	50		50
6	Переработка рабочей программы	до 0,25 на 1 а.ч.	47		50
Всего часов			161		272

Подготовка К Лек, Пр, Семинарским, Лабораторным Занятиям И Другие Виды Учебно-Методической Работы

ШАГ 1 ИЗ 6

Рисунок 27. Модуль карусельной таблицы

Из рисунка 27 видно, то что в данном компоненте пере используются названия таблиц, краткое описание к ним, превью таблицы и общее количество таблиц.

Данный модуль принимает массив объектов, содержащий следующие значения:

1. Поле name – имя таблицы
2. Поле urlTable – ссылку на шаблон таблицы
3. Поле title – название таблицы
4. Поле urlImg – ссылку на картинку для превью

В данном компоненте выбирается нужная таблица для редактирования после чего при нажатии на превью таблицы открывается модальное окно с редактором таблицы.

	A	B	C	D	E	F	G
1				Осенний семестр		Весенний семестр	
2				кол-во часов		кол-во часов	
3		Вид работы	нормы времени	по плану	факт. выполн.	по плану	факт. выполн.
4	1	2	3	4	5	6	7
5		2.1. Подготовка к лекционным, практическим, семинарским, лабораторным занятиям и другие виды учебно-методической работы (для программ ВО)					
6	1	Подготовка к лекциям	до 0,75 на 1 а.ч.	36		72	
7	2	Подготовка к лабораторным занятиям	до 0,25 на 1 а.ч.	12			
8	3	Подготовка к практическим занятиям	до 0,25 на 1 а.ч.	16		72	
9	4	Подготовка занятий на курсовую работу	до 0,75 на 1 раб-у			28	
10	5	Подготовка презентационных материалов	до 2ч. На 1 а.ч. но не более 100ч.	50		50	
11	6	Переработка рабочей программы	до 0,25 на 1 а.ч.	47		50	
12		Всего часов		161		272	

Рисунок 28. Модуль модального редактора таблицы

Данное модально окно подгружает выбранную таблицу из шаблона, в данном примере используется шаг второй половины дня, в которой данные задаются самим преподавателем, в магистратуре планируется реализовать контроллер для автоматизирования заполнения второй половины дня.

Пример генерации таблицы:

```
jexcel.fromSpreadsheet(this.table[this.table.activeSlide].urlTable, (result) => {
  if (!result.length) {
    console.error('JEXCEL: Something went wrong.');
```

```
  } else {
    if (result.length === 1) {
      const initialization = {...result[0], ...optionInitialization};
      this.spreadsheet = jexcel(this.spreadsheet.nativeElement, initialization);
    } else {
      jexcel.createTabs(this.spreadsheet.nativeElement, result);
    }
  }
});
```

После изменений вызывается оповещение об изменении в данной таблице с помощью материала дизайна Angular под названием `snackBar`. Данный элемент создает экземпляр компонента с указанными настройками:

1. Время жизни компонента
2. Позиционирование по горизонтали
3. Позиционирование по вертикали
4. Стиль модального окна (можно использовать кастомный, описанный в `scss`)

Пример вызова оповещения об изменениях:

```
openSnackBar(message: string, action, increased) {  
  this.opened.emit(increased);  
  const snackBarRef = this.snackBar.open(message, action, {  
    duration: 3000,  
    horizontalPosition: 'right'  
  });  
  
  snackBarRef.afterDismissed().subscribe(() => {  
    console.log('The snackBar was dismissed');  
  });  
  
  snackBarRef.onAction().subscribe(() => {  
    console.log('The snackBar action was triggered!');  
  });  
}
```

Также после того как изменения произошли и данные сохранились в локальном хранилище, для отправки данных на серверную часть в метод создания документа, которое срабатывает при переходе к следующему шагу, т.е. когда происходит `Destroy` компонента.

Теперь пару слов об `jExcel CE` (библиотека, плагин), который используется в формировании редактора таблиц, это легкий ванильный JavaScript плагин для создания удивительных интерактивных веб-таблиц HTML и электронных таблиц, совместимых с помощью Excel или любого другого программного обеспечения для электронных таблиц.

Благодаря ей можно создать электронную таблицу online из массива JS, Файлы JSON, CSV или XSLX. Вы можете скопировать из excel и вставить прямо в электронную таблицу JEXCEL CE и наоборот.

Это очень легко интегрировать любые сторонние плагины JavaScript для создания собственных пользовательских столбцов, пользовательских редакторов и настроить любой функция в вашем приложении. JEXCEL CE имеет множество различных вариантов ввода через свои собственные типы столбцов для покрытия наиболее распространенных веб-приложений требования.

Это комплексное решение для управления веб-данными. Создавайте удивительные приложения с помощью электронной таблицы JavaScript JEXCEL CE.

Основное преимущество:

- Сделайте богатые и удобные для пользователя веб-интерфейсы, и приложения
- Вы можете легко обрабатывать сложные входные данные таким образом, пользователи используются
- Улучшите свой пользовательский интерфейс программного обеспечения
- Создавайте богатые CRUDS и красивый пользовательский интерфейс
- Совместимость с excel: пользователи могут перемещать данные с помощью общих ярлыков копирования и вставки
- Простая настройка с помощью простых сторонних плагинов интеграции
- Постный, быстрый и простой в использовании
- Тысячи успешных случаев использования
- Ускорьте свою работу, связанную с трудным вводом данных в веб-программное обеспечение
- Бесплатное использование

Основные недостатки:

- Есть платная версия

- Все еще находиться в разработке
- Несколько разных документаций, в которых описаны только основные моменты кратко (пример, создание выпадающего списка для конкретной ячейки требует платной версии или по словам разработчик использовать старые и новые методы данного плагина, что влечет к нагромождению в коде)

Так же одной из важных составляющих этого плагина является большая документация на старую и новую версию плагина.

Текущей плагин находится на последней стадии разработки, поддерживается и исправляются ошибки каждый день.

Такой вывод преподносится из-за обмена информацией со стороны клиента (студента) и разработчика, как только плагин появился в прототипе ВКР, был брифинг с разработчиком из Лондона.

### **Авторское право и лицензия**

JEXCEL SE выпускается под лицензией [MIT license]. Авторские права принадлежат полу Ходелю paul.hodel@gmail.com

После того как все данные собраны и все шаги пройдены, появляется возможность сохранить данные и заполнить готовый шаблон, который лежит на сервере. После чего в этой же папке появиться готовый шаблон индивидуального плана преподавателя.

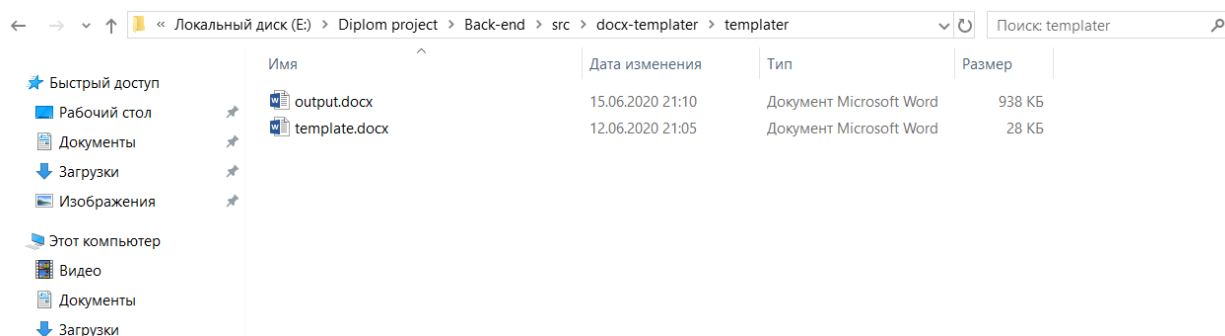


Рисунок 29. Директория шаблона



## ИНДИВИДУАЛЬНЫЙ ПЛАН ПРЕПОДАВАТЕЛЯ

(на 1 год)

Факультет ФИСТ  
Кафедра «Вычислительная техника»  
Фамилия Анна  
Имя Дмитрия  
Отчество Вячеславовна  
Должность старший преподаватель  
Год рождения 1979  
Ученая степень и год присуждения  
Ученое звание и год присвоения

Рисунок 30. Пример заполнения титульного листа

## 2. УЧЕБНО-МЕТОДИЧЕСКАЯ РАБОТА (учитывается по второй половине рабочего дня)

	Вид работы	нормы времени	Осенний семестр		Весенний семестр	
			кол-во часов		кол-во часов	
			по плану	факт. выполн.	по плану	факт. выполн.
1	2	3	4	5	6	7
<b>2.1. Подготовка к лекционным, практическим, семинарским, лабораторным занятиям и другие виды учебно-методической работы (для программ ВО)</b>						
1	Подготовка к лекциям	до 0,75 на 1 а.ч.	36		72	
2	Подготовка к лабораторным занятиям	до 0,25 на 1 а.ч.	12			
3	Подготовка к практическим занятиям	до 0,25 на 1 а.ч.	16		72	
4	Подготовка занятий на курсовую работу	до 0,75 на 1 раб-у			28	
5	Подготовка презентационных материалов	до 2ч. На 1 а.ч. но не более 100ч.	50		50	
6	Переработка рабочей программы	до 0,25 на 1 а.ч.	47		50	
	Всего часов		=SUM(D6:D11)		=SUM(F6:F11)	

Рисунок 31. Пример заполнения второй половины дня

На данном этапе создания документа видно, то что формулы, которые использовались в excel не преобразовываются. В дальнейшем есть 2 пути развития использования другого плагина по работе с электронными таблицами

(данный плагин протестирован на половину, на многие моменты такие как выпадающие списки для упрощения выбора дисциплин, не были воссозданы) либо использовать другой сторонний, удовлетворяющий плагин или на сборе данных реализовывать контроллер по преобразованию формул и последующего обращения к таблице по замене формул.

## **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
3. Преимущественно образовательный сегмент для создания индивидуального плана преподавателя на год

## Список использованных 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. Один каркас. Мобильные и настольные устройства — Режим доступа: <https://angular.io/>
4. JavaScript, который масштабируется. — Режим доступа: <https://www.typescriptlang.org/>
5. PostgreSQL Tutorial — Режим доступа: <https://www.postgresqltutorial.com/>
6. Язык запросов для вашего API — Режим доступа: <https://graphql.org/>
7. Оценка компаний: Анализ и прогнозирование с использованием отчетности — Режим доступа: <http://mdk-arbat.ru/book/7890>
8. jExcel CE is a JavaScript plugin — Режим доступа: <https://github.com/paulhodel/jexcel>
9. jExcel Pro версия — Режим доступа: <https://jexcel.net/v5/docs/getting-started>
10. Docxtemplater — Режим доступа: <https://docxtemplater.readthedocs.io/en/latest/>
11. Индивидуальный план преподавателя на 18-19 год — Режим доступа: ксерокопия
12. Расчет штатов — Режим доступа: электронная таблица excel
13. Что же такое этот GraphQL? — Режим доступа: <https://habr.com/ru/post/326986/>
14. Classdiagram-ts, простое создание диаграмм классов UML на основе исходных файлов typescript — Режим доступа:

<https://marketplace.visualstudio.com/items?itemName=AlexShen.classdiagram-ts>

15.Простой шаблон управления состоянием - NGXS – Режим доступа: <https://www.ngxs.io/>

16.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>

17.MongoDB и PostgreSQL мысли – Режим доступа: <https://stackoverflow.com/questions/4426540/mongodb-and-postgresql-thoughts>

18.Is Postgres NoSQL Better Than MongoDB? – Режим доступа: <http://www.aptuz.com/blog/is-postgres-nosql-database-better-than-mongodb/>

19.Тutorial по Angular HttpClient – Режим доступа: <https://ua-blog.com/туториал-по-angular-httpclient/>

20.Основы Angular, определение маршрутов – Режим доступа: <https://metanit.com/web/angular2/7.1.php>

21.Диаграмма связей базы данных – Режим доступа: <https://dbdiagram.io/d/5ea0718e39d18f5553fe0ae6>

22.Зачем нам UML? Или как сохранить себе нервы и время – Режим доступа: <https://habr.com/ru/post/458680/>

23.Material Design components for Angular – Режим доступа: <https://material.angular.io/>

24.ПОЛОЖЕНИЕ ОБ ИНДИВИДУАЛЬНЫХ ПЛАНАХ РАБОТЫ ПРЕПОДАВАТЕЛЕЙ – Режим доступа: [http://www.rzgmu.ru/images/upload/subdivisions/296\\_06102017.pdf](http://www.rzgmu.ru/images/upload/subdivisions/296_06102017.pdf)

25.ПОЛОЖЕНИЕ ОБ ИНДИВИДУАЛЬНЫХ ПЛАНАХ РАБОТЫ ПРЕПОДАВАТЕЛЕЙ – Режим доступа:

[http://www.chgpu.edu.ru/uploads/files/1556256339\\_pol.-ob-ind.plane-raboty-prepodavatelya-na-sayt.pdf](http://www.chgpu.edu.ru/uploads/files/1556256339_pol.-ob-ind.plane-raboty-prepodavatelya-na-sayt.pdf)

- 26.Текстовые редакторы и текстовые процессоры – Режим доступа: <https://infl.info/wordprocessor> - Загл. с экрана.
- 27.Автоматизированное рабочее место, его состав и назначение – Режим доступа: <https://tovaroveded.ru/lektsii-tovarovedenie/49-avtomatizirovannoe-rabochee-mesto-ego-sostav-i-naznachenie>
- 28.Напряженность труда: критический взгляд на действующие критерии оценки – Режим доступа: <http://www.kiout.ru/info/publish/6090> - Загл. с экрана.
- 29.Логическая модель предметной области – Режим доступа: <http://analyst.by/diagrams/logicheskaya-model-predmetnoy-oblasti>
- 30.Вёрстка модальных окон – Режим доступа: <https://zen.yandex.ru/media/id/5d4d3407f0d4f400ad6c9bd2/verstka-modalnyh-okon-5dbee9fe7cccba00afd5bbe0>
- 31.node.bcrypt.js – Режим доступа: <https://www.npmjs.com/package/bcrypt>
- 32.Pidcrypt (JS encryption library) for node.js/browserify – Режим доступа: <https://www.npmjs.com/package/pidcrypt>
- 33.Creately Draw – Режим доступа: <https://creately.com/app/>
- 34.Angular курсы – Режим доступа: <https://www.udemy.com/>
- 35.Руководство по TypeScript – Режим доступа: <https://metanit.com/web/typescript/>
- 36.Облачная программа для управления проектами небольших групп – Режим доступа: <https://trello.com/>
- 37.БЭМ (Блок, Элемент, Модификатор) — компонентный подход к веб-разработке. – Режим доступа: <https://ru.bem.info/>
- 38.CSS с суперсилой – Режим доступа: <https://sass-scss.ru/>

## Приложение

### Листинг программы

Подключение к базе данных (Graph Ql) и основных модулей серверной части

```
@Module({
  imports: [
    SequelizeModule.forRoot({
      dialect: 'postgres',
      host: 'localhost',
      port: 5432,
      username: 'postgres',
      password: 'postgresql',
      database: 'work_programme',
      autoLoadModels: true,
      synchronize: true,
      define: {
        timestamps: false
      }
    }),
    UsersModule,
    FacultyModule,
    DepartmentsModule,
    OfficialPersonUsersPlansModule,
    PlanDisciplinesModule,
    WorkProgrammesModule,
    FormEducationalModule,
    StatusEducationalProcessModule,
    OOPModule,
    StaffListModule,
    LevelEducationModule,
    CalculationDepartmentHoursModule,
    CalculationDepartmentHoursOnPlanModule,
    OfficialPersonModule,
    GraphQLModule.forRoot({
      autoSchemaFile: 'schema.gql',
    }),
    AuthModule,
    DocxTemplaterModule,
  ],
  controllers: [AppController],
})
export class AppModule {}
```

Контроллер локального хостинга

```
@Controller()
export class AppController {
  constructor(
    private readonly authService: AuthService,
```

```

        private readonly docxTemplaterService: DocxTemplaterService
    ) {}

    @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;
    }

    @Post('templater')
    async createDocxTemplater(@Request() req) {
        return this.docxTemplaterService.createDocxTemplater(req.body);
    }
}

```

### Сервис по работе с пользователем

```

@Injectable()
export class UsersService {
    constructor(
        @Inject('USERS_REPOSITORY') private readonly USERS_REPOSITORY: typeof User,
    ) {}

    async findAllUsers(): Promise<User[]> {
        return await this.USERS_REPOSITORY.findAll<User>();
    }

    async getUser(id: number): Promise<User> {
        return this.USERS_REPOSITORY.findByPk<User>(id);
    }

    async getAccount(username: string): Promise<User> {
        return this.USERS_REPOSITORY.findOne<User>({where: {username: username}});
    }

    async createAccount(user: UserInput): Promise<User> {
        const account = await this.getAccount(user.username);

        if(!account) {

```



```

    const salt = await bcrypt.genSalt(10);
    const password = await bcrypt.hash(user.password, salt);
    const userDto = {...user, password};

    return this.USERS_REPOSITORY.create<User>(userDto);
  } else {
    throw new UnauthorizedException();
  }
}

async updateAccount(user: UserInput) {
  return this.USERS_REPOSITORY.update(user, {where: {id: user.id}});
}

async deleteAccount(username: string) {
  return this.USERS_REPOSITORY.destroy({where: {username: username}});
}
}

```

Распознаватель для работы с юзером с помощью Graph QL

```

@Resolver()
export class UsersResolver {
  constructor(private readonly userService: UsersService) {}

  @Query(() => [CreateUserDto])
  async users() {
    return this.userService.findAllUsers();
  }

  @Query(() => CreateUserDto, { nullable: true })
  async getUser(@Args({ name: 'id', type: () => Int }) id: number) {
    return this.userService.getUser(id);
  }

  @Query(() => CreateUserDto, { nullable: true })
  async getAccount(@Args({ name: 'username', type: () => String }) username: string) {
    return this.userService.getAccount(username);
  }

  @Mutation(() => CreateUserDto)
  async createAccount(@Args('user') user: UserInput) {
    return this.userService.createAccount(user);
  }

  @Mutation(() => CreateUserDto)
  async updateAccount(@Args('user') user: UserInput) {
    return this.userService.updateAccount(user);
  }

  @Mutation(() => CreateUserDto)

```

```

    async deleteAccount(@Args('username') user: UserInput) {
      return this.usersService.deleteAccount(user.username);
    }
  }
}

```

## Сущность распознавателя GraphQL

```

@InputType()
export class UserInput {
  @Field(() => ID, { nullable: true })
  readonly id?: number;

  @Field()
  readonly firstName: string;

  @Field()
  readonly lastName: string;

  @Field()
  readonly middleName: string;

  @Field(() => Int)
  readonly birthYear: number;

  @Field(() => Int)
  readonly departmentsId: number;

  @Field()
  readonly username: string;

  @Field()
  readonly password: string;

  @Field()
  readonly email: string;

  @Field()
  readonly degree: string;

  @Field()
  readonly academicTitle: string;

  @Field(() => Int)
  readonly role?: number;
}

```

## Файл API QL

```

input CalculationDepartmentHoursInput {
  id: ID
  lecturesTotal: Int!
  practiceSeminarTotal: Int!
}

```

```

laboratoryTotal: Int!
consultations: Int!
exam: Int!
differentiatedTest: Int!
test: Int!
courseWork: Int!
courseProject: Int!
stateExam: Int!
diplomaDesign: Int!
hec: Int!
rukov_orgWork: Int!
educationalPractice: Int!
manufacturingPractice: Int!
externShip: Int!
scientificPedagogicalPractice: Int!
reviewsDissentsWorks: Int!
abstracts_rgr: Int!
controlWork: Int!
rukovResearchWorkPostgraduates: Int!
CalculationDepartmentHoursOnPlanId: Int!
}
type CreateCalculationDepartmentHoursDto {
  id: ID!
  lecturesTotal: Int!
  practiceSeminarTotal: Int!
  laboratoryTotal: Int!
  consultations: Int!
  exam: Int!
  differentiatedTest: Int!
  test: Int!
  courseWork: Int!
  courseProject: Int!
  stateExam: Int!
  diplomaDesign: Int!
  hec: Int!
  rukov_orgWork: Int!
  educationalPractice: Int!
  manufacturingPractice: Int!
  externShip: Int!
  scientificPedagogicalPractice: Int!

```

```

reviewsDissentsWorks: Int!
abstracts_rgr: Int!
controlWork: Int!
rukovResearchWorkPostgraduates: Int!
CalculationDepartmentHoursOnPlanId: Int!
}
type CreateDepartmentDto {
  id: ID!
  facultyId: Int!
  department: String!
  headDepartment: Int!
}
type CreateFacultyDto {
  id: ID!
  faculty: String!
}
type CreateFormEducationalDto {
  id: ID!
  educationalFormat: String!
}
type CreateLevelEducationDto {
  id: ID!
  level: String!
}
type CreateOfficialPersonDto {
  id: ID!
  programmeId: Int!
  position: String!
  staffListId: Int!
  statusEducationalProcessId: Int!
}
type CreateOfficialPersonUsersPlanDto {
  id: ID!
  usersId: Int!
  positionId: Int!
  disciplineId: Int!
}
type CreateOOPDto {
  id: ID!
  cipher: String!
}

```

```

    name: String!
    levelEducationId: Int!
    period: Int!
    date: String!
}
type CreatePlanDisciplineDto {
    id: ID!
    programmeId: Int!
    oopId: Int!
    departmentsId: Int!
    discipline: String!
    codeDiscipline: String!
    course: Int!
    numberStudents: Int!
    numberGroups: Int!
    numberSubGroups: Int!
    calculationDepartmentHoursId: Int!
}
type CreateStaffListDto {
    id: ID!
    load: Int!
    rate: Int!
    generalCountForStaffListRate: Int!
    generalCountForStaffListClock: Int!
    hoursRates: Int!
    intensityRate: Int!
    intensityClock: Int!
    gphRate: Int!
    gphClock: Int!
}
type CreateUserDto {
    id: ID!
    firstName: String!
    lastName: String!
    middleName: String!
    birthYear: Int!
    departmentsId: Int!
    username: String!
    password: String!
    email: String!

```

```

    degree: String!
    academicTitle: String!
    role: Int!
}
type CreateWorkProgrammeDto {
    id: ID!
    userId: Int!
    title: String!
    description: String!
    date: DateTime!
    departmentsId: Int!
    facultysId: Int!
    formEducationalId: Int!
    oopId: Int!
}
input DepartmentInput {
    id: ID
    facultyId: Int!
    department: String!
    headDepartment: Int!
}
input FacultyInput {
    id: ID
    faculty: String!
}
type Mutation {
    createAccount(user: UserInput!): CreateUserDto!
    updateAccount(user: UserInput!): CreateUserDto!
    deleteAccount(username: UserInput!): CreateUserDto!
    createFaculty(faculty: FacultyInput!): CreateFacultyDto!
    updateFaculty(faculty: FacultyInput!): CreateFacultyDto!
    deleteFaculty(id: Float!): CreateFacultyDto!
    createDepartment(department: DepartmentInput!): CreateDepartmentDto!
    updateDepartment(department: DepartmentInput!): CreateDepartmentDto!
    deleteDepartment(id: Float!): CreateDepartmentDto!
    createPersonUsersPlan(personUsersPlan:           OfficialPersonUsersPlanInput!):
CreateOfficialPersonUsersPlanDto!
    updatePersonUsersPlan(personUsersPlan:           OfficialPersonUsersPlanInput!):
CreateOfficialPersonUsersPlanDto!
    deletePersonUsersPlan(id: Float!): CreateOfficialPersonUsersPlanDto!

```

```

createPlanDiscipline(planDiscipline: PlanDisciplineInput!): CreatePlanDisciplineDto!
updatePlanDiscipline(planDiscipline: PlanDisciplineInput!): CreatePlanDisciplineDto!
deletePlanDiscipline(id: Float!): CreatePlanDisciplineDto!
createProgramme(wProgramme: WorkProgrammeInput!): CreateWorkProgrammeDto!
updateProgramme(wProgramme: WorkProgrammeInput!): CreateWorkProgrammeDto!
deleteProgramme(id: Float!): CreateWorkProgrammeDto!
createFormEducational(formEducational: FormEducationalInput!): CreateFormEducationalDto!
updateFormEducational(formEducational: FormEducationalInput!): CreateFormEducationalDto!
deleteFormEducational(id: Float!): CreateFormEducationalDto!
createEducationalProcess(edProcess: StatusEducationalProcessInput!): CreateStatusEducationalProcessDto!
updateEducationalProcess(edProcess: StatusEducationalProcessInput!): CreateStatusEducationalProcessDto!
deleteEducationalProcess(id: Float!): CreateStatusEducationalProcessDto!
createOOP(oop: OOPInput!): CreateOOPDto!
updateOOP(oop: OOPInput!): CreateOOPDto!
deleteOOP(id: Float!): CreateOOPDto!
createStaffList(staffList: StaffListInput!): CreateStaffListDto!
updateStaffList(staffList: StaffListInput!): CreateStaffListDto!
deleteStaffList(id: Float!): CreateStaffListDto!
createLevelEducation(levelEducation: LevelEducationInput!): CreateLevelEducationDto!
updateLevelEducation(levelEducation: LevelEducationInput!): CreateLevelEducationDto!
deleteLevelEducation(id: Float!): CreateLevelEducationDto!
createCalculationDepartmentHours(cDepartmentHours:          CalculationDepartmentHoursInput!):
CreateCalculationDepartmentHoursDto!
updateCalculationDepartmentHours(cDepartmentHours:          CalculationDepartmentHoursInput!):
CreateCalculationDepartmentHoursDto!
deleteCalculationDepartmentHours(id: Float!): CreateCalculationDepartmentHoursDto!
createCalculationDepartmentHoursOnPlan(cDepartmentHoursOnPlan:
CalculationDepartmentHoursOnPlanInput!): CreateCalculationDepartmentHoursOnPlanDto!
updateCalculationDepartmentHoursOnPlan(cDepartmentHoursOnPlan:
CalculationDepartmentHoursOnPlanInput!): CreateCalculationDepartmentHoursOnPlanDto!
deleteCalculationDepartmentHoursOnPlan(id: Float!): CreateCalculationDepartmentHoursOnPlanDto!
createOfficialPerson(officialPerson: OfficialPersonInput!): CreateOfficialPersonDto!
updateOfficialPerson(officialPerson: OfficialPersonInput!): CreateOfficialPersonDto!
deleteOfficialPerson(id: Float!): CreateOfficialPersonDto!
}
input PlanDisciplineInput {
  id: ID
  programmeId: Int!
  oopId: Int!
  departmentsId: Int!

```

```

discipline: String!
codeDiscipline: String!
course: Int!
numberStudents: Int!
numberGroups: Int!
numberSubGroups: Int!
calculationDepartmentHoursId: Int!
}

type Query {
  users: [CreateUserDto!]!
  getUser(id: Int!): CreateUserDto
  getAccount(username: String!): CreateUserDto
  facultys: [CreateFacultyDto!]!
  getFaculty(id: Int!): CreateFacultyDto
  departments: [CreateDepartmentDto!]!
  getDepartment(id: Int!): CreateDepartmentDto
  personUsersPlans: [CreateOfficialPersonUsersPlanDto!]!
  getPersonUsersPlan(id: Int!): CreateOfficialPersonUsersPlanDto
  PlanDisciplines: [CreatePlanDisciplineDto!]!
  getPlanDiscipline(id: Int!): CreatePlanDisciplineDto
  workProgrammes: [CreateWorkProgrammeDto!]!
  getWorkProgramme(id: Int!): CreateWorkProgrammeDto
  formEducational: [CreateFormEducationalDto!]!
  getFormEducational(id: Int!): CreateFormEducationalDto
  allStatusEducationalProcess: [CreateStatusEducationalProcessDto!]!
  getStatusEducationalProcess(id: Int!): CreateStatusEducationalProcessDto
  allOOP: [CreateOOPDto!]!
  getOOP(id: Int!): CreateOOPDto
  staffLists: [CreateStaffListDto!]!
  getStaffList(id: Int!): CreateStaffListDto
  levelsEducation: [CreateLevelEducationDto!]!
  getLevelEducation(id: Int!): CreateLevelEducationDto
  allCalculationDepartmentHours: [CreateCalculationDepartmentHoursDto!]!
  getCalculationDepartmentHours(id: Int!): CreateCalculationDepartmentHoursDto
  allCalculationDepartmentHoursOnPlan: [CreateCalculationDepartmentHoursOnPlanDto!]!
  getCalculationDepartmentHoursOnPlan(id: Int!): CreateCalculationDepartmentHoursOnPlanDto
  officialPersons: [CreateOfficialPersonDto!]!
  getOfficialPerson(id: Int!): CreateOfficialPersonDto
}

```



## Сервис создания документа индивидуального плана

```
@Injectable()
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
r, 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);
  }
}
```

```
}
```

### Паспорт стратегия, проверка пользователя

```
@Injectable()
export class LocalStrategy extends PassportStrategy(Strategy) {
  constructor(private authService: AuthService) {
    super();
  }

  async validate(username: string, password: string): Promise<any> {
    const user = await this.authService.validateUser(username, password);
    if (!user) {
      throw new UnauthorizedException();
    }
    return user;
  }
}
```

### JWT стратегия по получению токена

```
@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor() {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: jwtConstants.secret,
    });
  }

  async validate(payload: any) {
    return { id: payload.sub, username: payload.username };
  }
}
```

### Сервис аутентификация

```
@Injectable()
export class AuthService {
  constructor(
    private usersService: UsersService,
    private readonly jwtService: JwtService,
  ) {}

  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;
    }

    async login(user: any) {
        const payload = { username: user.dataValues.username, sub: user.dataValues.id };
    };
    return {
        access_token: this.jwtService.sign(payload),
    };
}

async signUp(authCredentialsDto: User) {
    try{
        await this.usersService.createAccount(authCredentialsDto);
    }catch(error){
        return error;
        // console.log(error);
        // this.logger.error(error);
        // if(error.code == 23505){//duplicate username
        //     throw new ConflictException("Username already exists");
        // }else{
        //     throw new InternalServerErrorException();
        // }
    }
}
}
}

```

### Компонент по обработке модального редактора эл. Таблицы

```

export class ModalTableComponent implements OnInit {
    @ViewChild('spreadsheet', {static: false}) spreadsheet: any;
    @Output() opened = new EventEmitter<boolean>();
    @Input() table;
    @Select(CloudState.getClouds) clouds$: Observable<Cloud[]>;
    tableName: string;
    tableStore: any;

    constructor(
        private store: Store,
        private snackBar: MatSnackBar
    ) { }

    ngOnInit() {
        this.tableName = this.table[this.table.activeSlide].name + this.table.activeSlide;
        this.tableStore = {
            [this.tableName]: false
        };

        jexcel.fromSpreadsheet(this.table[this.table.activeSlide].urlTable, (result) => {

```

```

        if (!result.length) {
            console.error('JEXCEL: Something went wrong.');
```

```
        } else {
            if (result.length === 1) {
                const initialization = {...result[0], ...optionInitialization};
                this.spreadsheet = jexcel(this.spreadsheet.nativeElement, initialization);

                // console.log(this.spreadsheet.table.rows[2]);
            } else {
                jexcel.createTabs(this.spreadsheet.nativeElement, result);
            }
        }
    });

    this.clouds$.subscribe(v1 => {
        v1.forEach(element => {
            if (element[this.tableName]) {
                this.tableStore[this.tableName] = true;
            }
        });
    });
}

openSnackBar(message: string, action, increased) {
    this.opened.emit(increased);
    const snackBarRef = this.snackBar.open(message, action, {
        duration: 3000,
        horizontalPosition: 'right'
    });

    snackBarRef.afterDismissed().subscribe(() => {
        console.log('The snackBar was dismissed');
    });

    snackBarRef.onAction().subscribe(() => {
        console.log('The snackBar action was triggered!');
    });
}

ngOnDestroy() {
    if (!this.tableStore[this.tableName]) {
        console.log(this.spreadsheet.getJson());
        this.store.dispatch(new AddCloud({[this.tableName]: this.spreadsheet.getJson()}));
    } else {
        // update
    }
}

```

```
}
```

## Модуль подключения к API QL с клиентской части

```
const uri = 'http://localhost:3000/graphql'; // <-  
- add the URL of the GraphQL server here  
export function createApollo(httpLink: HttpLink) {  
  return {  
    link: httpLink.create({uri}),  
    cache: new InMemoryCache(),  
  };  
}  
  
@NgModule({  
  exports: [ApolloModule, HttpLinkModule],  
  providers: [  
    {  
      provide: APOLLO_OPTIONS,  
      useFactory: createApollo,  
      deps: [HttpLink],  
    },  
  ],  
})  
export class GraphQLModule {}
```

## Классы действий в локальном хранилище

```
export class AddCloud {  
  static readonly type = '[CLOUD] Add';  
  constructor(public payload: Cloud) {}  
}  
export class AddTitlePageIntro {  
  static readonly type = '[titlePageIntro] Add';  
  constructor(public payload: boolean) {}  
}  
export class AddAudienceLoadIntro {  
  static readonly type = '[audienceLoadIntro] Add';  
  constructor(public payload: boolean) {}  
}  
export class AddPageAfternoonIntro {  
  static readonly type = '[pageAfternoonIntro] Add';  
  constructor(public payload: boolean) {}  
}  
export class AddPageResultIntro {  
  static readonly type = '[pageResultIntro] Add';  
  
  constructor(public payload: boolean) {}  
}  
export class RemoveCloud {  
  static readonly type = '[CLOUD] Remove';  
  constructor(public payload: string) {}  
}
```

```
export class ChangeCloud {
  static readonly type = '[CLOUD] Change';
  constructor(public payload: string) {}
}
```

## Состояния действий в локальном хранилище

```
// Section 1
import { State, Action, StateContext, Selector } from '@ngxs/store';
import { Cloud } from '../../models/cloud.model';
import { AddCloud, RemoveCloud, ChangeCloud, AddTitlePageIntro, AddAudienceLoadIntro, AddPageAfternoonIntro, AddPageResultIntro } from '../../actions/cloud.actions';

// Section 2
export class CloudStateModel {
  clouds?: Cloud[];
  titlePageIntro?: boolean;
  audienceLoadIntro?: boolean;
  pageAfternoonIntro?: boolean;
  pageResultIntro?: boolean;
}

// Section 3
@State<CloudStateModel>({
  name: 'clouds',
  defaults: {
    clouds: [],
    titlePageIntro: false,
    audienceLoadIntro: false,
    pageAfternoonIntro: false,
    pageResultIntro: false
  }
})

export class CloudState {

  // Section 4
  @Selector()
  static getClouds(state: CloudStateModel) {
    return state.clouds;
  }

  @Selector()
  static getTitlePageIntro(state: CloudStateModel) {
    return state.titlePageIntro;
  }

  @Selector()
  static getAudienceLoadIntro(state: CloudStateModel) {
    return state.audienceLoadIntro;
  }
}
```

```

@Selector()
static getPageAfternoonIntro(state: CloudStateModel) {
    return state.pageAfternoonIntro;
}

@Selector()
static getPageResultIntro(state: CloudStateModel) {
    return state.pageResultIntro;
}

// Section 5
@Action(AddCloud)
add({getState, patchState }: StateContext<CloudStateModel>, { payload }: AddC
loud) {
    const state = getState();
    patchState({
        clouds: [...state.clouds, payload]
    });
}

@Action(AddTitlePageIntro)
addTitlePageIntro({getState, patchState }: StateContext<CloudStateModel>, pay
load: boolean) {
    const state = getState();
    patchState({
        titlePageIntro: payload
    });
}

@Action(AddAudienceLoadIntro)
addAudienceLoadIntro({getState, patchState }: StateContext<CloudStateModel>,
payload: boolean) {
    const state = getState();
    patchState({
        audienceLoadIntro: payload
    });
}

@Action(AddPageAfternoonIntro)
addPageAfternoonIntro({getState, patchState }: StateContext<CloudStateModel>,
payload: boolean) {
    const state = getState();
    patchState({
        pageAfternoonIntro: payload
    });
}

@Action(AddPageResultIntro)
addPageResultIntro({getState, patchState }: StateContext<CloudStateModel>, pa
yload: boolean) {

```

```

        const state = getState();
        patchState({
            pageResultIntro: payload
        });
    }

    @Action(RemoveCloud)
    remove({getState, patchState }: StateContext<CloudStateModel>, { payload }: RemoveCloud) {
        patchState({
            clouds: getState().clouds.filter(a => a.username !== payload)
        });
    }
}

```

Модуль сбора данных после прохождения шагов редактора

```

export class MainAreaComponent implements OnInit {
    currentRoute: string;
    now: string;
    objDocx: any;
    @Select(CloudState.getClouds) clouds$: Observable<Cloud[]>;

    constructor(
        private router: Router,
        private store: Store,
        private http: HttpClient
    ) { }

    ngOnInit() {
        this.currentRoute = '/main/page-authorizing';
        this.now = formatDate(new Date(), 'dd.MM.yyyy', 'en');
        this.objDocx = {};

        this.store.dispatch(new AddCloud({date: this.now}));
    }

    onBack() {
        this.router.navigate([this.currentRoute]);
    }

    getButtonClassName() {
        let className = 'button';
        if (this.router.url !== '/main/plan-editor/result-page') {
            className += ' button--disable';
        }
        return className;
    }

    getValue() {
        return 'Название';
    }
}

```



```

}

generateDocx() {
  this.clouds$.subscribe(v1 => {
    v1.forEach(element => {
      if (element.date) {
        element.date = element.date.substr(-4);
      }
      this.objDocx = {...this.objDocx, ...element};
    });

    this.objDocx.date = parseInt(this.objDocx.date, 10);
    this.objDocx.nextDate = this.objDocx.date + 1;
  });

  this.filterTableData(this.objDocx);
}

filterTableData(obj: any) {
  const pageAfternoon = new PageAfternoon();
  const pageResult = new PageResult();
  const audienceLoad = new AudienceLoad();

  const filterData = {
    date: obj.date,
    nextDate: obj.nextDate,
    titlePage: obj.titlePage,

    audienceLoad0: obj.audienceLoad0 ? obj.audienceLoad0.filter((el, i) => {
      return i !== 0 && i !== 1;
    }) : audienceLoad.audienceLoad0,
    audienceLoad1: obj.audienceLoad1 ? obj.audienceLoad1.filter((el, i) => {
      return i !== 0 && i !== 1;
    }) : audienceLoad.audienceLoad1,

    pageAfternoon0: obj.pageAfternoon0 ? obj.pageAfternoon0.filter((el, i) => {
      return i !== 0 && i !== 1 && i !== 2 && i !== 3 && i !== 4;
    }) : pageAfternoon.pageAfternoon0,
    pageAfternoon1: obj.pageAfternoon1 ? obj.pageAfternoon1.filter((el, i) => {
      return i !== 0 && i !== 1 && i !== 2 && i !== 3 && i !== 4;
    }) : pageAfternoon.pageAfternoon1,
    pageAfternoon2: obj.pageAfternoon2 ? obj.pageAfternoon2.filter((el, i) => {
      return i !== 0 && i !== 1 && i !== 2 && i !== 3 && i !== 4;
    }) : pageAfternoon.pageAfternoon2,
    pageAfternoon3: obj.pageAfternoon3 ? obj.pageAfternoon3.filter((el, i) => {
      return i !== 0 && i !== 1 && i !== 2 && i !== 3 && i !== 4;
    }) : pageAfternoon.pageAfternoon3,
    pageAfternoon4: obj.pageAfternoon4 ? obj.pageAfternoon4.filter((el, i) => {
      return i !== 0 && i !== 1 && i !== 2 && i !== 3;
    }) : pageAfternoon.pageAfternoon4,
  };

```

```

        pageAfternoon5: obj.pageAfternoon5 ? obj.pageAfternoon5 : pageAfternoon.pageAfternoon5,

        pageResult0: obj.pageResult0 ? obj.pageResult0 : pageResult.pageResult0,
        pageResult1: obj.pageResult1 ? obj.pageResult1 : pageResult.pageResult1,
        pageResult2: obj.pageResult2 ? obj.pageResult2 : pageResult.pageResult2
    };

    this.http.post('http://localhost:3000/templater', filterData, httpOptions).subscribe(res => {
        // console.log(res);
        alert('This demo generate document\nOutput File: output.docx\nPath:E:/Diplom project/Back-end/src/docx-templater/templater/output.docx');
    },
    error => {
        console.log(error);
    });
}
}

```

Модель заполнения титульного листа в редакторе

```

export class TitlePageComponent implements OnInit {
    dataDto = [
        {
            title: 'faculty',
            placeholder: 'Факультет',
            value: ''
        },
        {
            title: 'department',
            placeholder: 'Кафедра',
            value: ''
        },
        {
            title: 'firstName',
            placeholder: 'Фамилия',
            value: ''
        },
        {
            title: 'lastName',
            placeholder: 'Имя',
            value: ''
        },
        {
            title: 'middleName',
            placeholder: 'Отчество',
            value: ''
        },
        {
            title: 'position',
            placeholder: 'Должность',

```

```

        value: ''
    },
    {
        title: 'birthYear',
        placeholder: 'Год рождения',
        value: ''
    },
    {
        title: 'degree',
        placeholder: 'Ученая степень и год присуждения',
        value: ''
    },
    {
        title: 'academicTitle',
        placeholder: 'Ученое звание и год присвоения',
        value: ''
    }
];
dataList: any;
titleStore: boolean;
@Select(CloudState.getClouds) clouds$: Observable<Cloud[]>;
@Select(CloudState.getTitlePageIntro) titlePageIntro$: Observable<boolean>;

constructor(
    private apollo: Apollo,
    private store: Store,
    private dialog: MatDialog
) { }

ngOnInit() {
    this.titleStore = false;

    this.titlePageIntro$.subscribe(intro => {
        if (!intro) {
            const dialogRef = this.dialog.open(DialogTitlePageComponent);
            dialogRef.afterClosed().subscribe(result => {
                console.log(`Dialog closed`);
            });
        }
    });

    this.apollo.query({
        query: gql`query {
            getAccount(username: "a.lylova") {
                firstName, lastName, middleName, birthYear, degree, academicTitle
            }
            getFaculty(id: 1) {
                faculty
            }
            getDepartment(id: 1) {

```

```

        department
      }
      getOfficialPerson(id: 2) {
        position
      }
    }`
  }).subscribe(({ data }) => {
    /*****
    *****/

    Переписать

    /*****
    *****/
    this.dataList = {...this.dataList, ...data['getAccount'], ...data['getFaculty'], ...this.dataList, ...data['getDepartment'], ...this.dataList, ...data['getOfficialPerson']];
    delete this.dataList.__typename;

    this.dataDto.forEach(element => {
      for (const key in this.dataList) {
        if (key === element.title) {
          element.value = this.dataList[key];
        }
      }
    });

    this.clouds$.subscribe(v1 => {
      v1.forEach(element => {
        if (element.titlePage) {
          this.titleStore = true;
        }
      });
    });
  });
}

ngOnDestroy() {
  if (!this.titleStore) {
    this.store.dispatch(new AddTitlePageIntro(true));
    this.store.dispatch(new AddCloud({titlePage: this.dataList}));
  }
}
}

```

Модуль валидации и сбора данных с формы входа

```

export class FormData {
  username: string;
  password: string;
}

```

```

const httpOptions = {
  headers: new HttpHeaders({ 'Content-Type': 'application/json; charset=utf-8' })
};

export class LoginFormComponent implements OnInit {
  @Output() opened = new EventEmitter<boolean>();
  user: FormData;
  error: boolean;
  isLoading: string;
  data: any;

  constructor(
    private router: Router,
    private http: HttpClient
  ) { }

  ngOnInit() {
    this.user = {
      username: '',
      password: ''
    };
    this.error = true;
    this.isLoading = 'Вход';
  }

  change(increased: any) {
    this.opened.emit(increased);
  }

  onSubmit() {
    this.isLoading = 'Вход...';

    this.http.post('http://localhost:3000/auth/login', this.user, httpOptions).
    subscribe(value => {
      const token = value['access_token'];
      if (token) {
        // localStorage.setItem('token', token);
        const httpOption = {
          headers: new HttpHeaders(
            {
              'Content-Type': 'application/json',
              Authorization: `Bearer ${token}`
            }
          )
        };
      }
    });

    this.http.get('http://localhost:3000/profile', httpOption).subscribe(re
s => {
      console.log(res);
    });
  }
}

```

```

        this.isLoad = 'Вход';
        this.error = true;
        this.router.navigate(['/main/page-authorizing']);
    },
    error => {
        console.log(error);
    });
} else {
    this.isLoad = 'Вход';
    this.error = false;
    return;
}
},
error => {
    console.log(error);
    this.isLoad = 'Вход';
    this.error = false;
    return;
});
}
}

```

### Файл типографии

```

/*
    Typography
*/

$space: 16px;
$space--smaller: 8px;

@mixin base-list {
    margin: 0;
    padding: 0;
    list-style: none;
}

@mixin list {
    display: flex;
    flex-direction: row;
    font-size: 16px;

    &:hover {
        background: #292d43;
    }
}

@mixin headline {
    text-transform: capitalize;
    line-height: 2rem;
}

```

```

@mixin headline-1 {
  @include headline();
  font-size: 1.5rem;
  font-weight: 400;
  letter-spacing: normal;
}

@mixin headline-2 {
  @include headline();
  font-size: 1.25rem;
  font-weight: 500;
  letter-spacing: 0.0125rem;
}

@mixin sub-title-1 {
  font-size: 1rem;
  line-height: 1.75rem;
  font-weight: 400;
  letter-spacing: 0.009375em;
}

@mixin sub-title-2 {
  font-size: 0.875rem;
  line-height: 1.375rem;
  font-weight: 500;
  letter-spacing: 0.0071428571em;
}

@mixin text {
  font-weight: 400;
}

@mixin text-1 {
  @include text();
  font-size: 1rem;
  line-height: 1.5rem;
  letter-spacing: 0.03125em;
}

@mixin text-2 {
  @include text();
  font-size: 0.875rem;
  line-height: 1.25rem;
  letter-spacing: 0.0178571429em;
}

@mixin button {
  font-size: 0.875rem;
  font-weight: 500;
  line-height: 2.25rem;
}

```

```

letter-spacing: 0.0892857143em;
}

@mixin caption {
  font-size: 0.75rem;
  font-weight: 400;
  line-height: 1.25rem;
  letter-spacing: 0.0333333333em;
}

@mixin overline {
  font-size: 0.75rem;
  font-weight: 500;
  line-height: 2rem;
  letter-spacing: 0.1666666667em;
}

```

## Шаблон индивидуального плана

Министерство науки и высшего образования Российской Федерации  
 федеральное государственное бюджетное образовательное  
 учреждение высшего образования  
 «УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

### ИНДИВИДУАЛЬНЫЙ ПЛАН ПРЕПОДАВАТЕЛЯ (на 1 год)

Факультет	<u>{#titlePage}{faculty}/{titlePage}</u>
Кафедра	<u>«{#titlePage}{department}/{titlePage}»</u>
Фамилия	<u>{#titlePage}{firstName}/{titlePage}</u>
Имя	<u>{#titlePage}{lastName}/{titlePage}</u>
Отчество	<u>{#titlePage}{middleName}/{titlePage}</u>
Должность	<u>{#titlePage}{position}/{titlePage}</u>
Год рождения	<u>{#titlePage}{birthYear}/{titlePage}</u>
Ученая степень и год присуждения	<u>{#titlePage}{degree}/{titlePage}</u>
Ученое звание и год присвоения	<u>{#titlePage}{academicTitle}/{titlePage}</u>



Утверждаю

Зав. кафедрой \_\_\_\_\_  
год \_\_\_\_\_

## 1. УЧЕБНАЯ РАБОТА

### 1.1 Нагрузка преподавателя по программам высшего образования (ВО)

{date}/{nextDate} уч.

а) Осенний семестр

“ ” 20\_\_ г

Дисциплина N группы (потока)	количество часов по видам учебной работы																
	Студентов	Лекции	Факт занятия и семинары	лабораторные занятия	курсовое проектирование	консультации	зачеты	экзамены	Диф. Зачеты	руководство практиками студентов	расчетно-графические работы	рук-во зан. проектом (зан. работой)	рецензирование зан. проектов (зан. работ)	работа в ГИА	Рефераты, РГР	руководство аспирантами	Другие виды работ
{#audienceLoad0}{0}	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}	{9}	{10}	{11}	{12}	{13}	{14}	{15}	{16}	{17}
																	всего часов фактически
																	{19}

б) Весенний семестр

Дисциплина N группы (потока)	количество часов по видам учебной работы																
	Студентов	Лекции	Факт занятия и семинары	лабораторные занятия	курсовое проектирование	консультации	зачеты	экзамены	Диф. Зачеты	руководство практиками студентов	расчетно-графические работы	рук-во зан. проектом (зан. работой)	рецензирование зан. проектов (зан. работ)	работа в ГИА	Рефераты, РГР	руководство аспирантами	Другие виды работ
{#audienceLoad1}{0}	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}	{9}	{10}	{11}	{12}	{13}	{14}	{15}	{16}	{17}
																	всего часов фактически
																	{19}

## 2. УЧЕБНО-МЕТОДИЧЕСКАЯ РАБОТА (учитывается по второй половине рабочего дня)

1	Вид работы	нормы времени	Осенний семестр		Весенний семестр	
			кол-во часов		кол-во часов	
			по плану	факт. выполн.	по плану	факт. выполн.
	2	3	4	5	6	7
2.1. Подготовка к лекционным, практическим, семинарским, лабораторным занятиям и другие виды учебно-методической работы (для программ ВО)						
{#pageAfternoon0}{0}	{1}	{2}	{3}	{4}	{5}	{6}
2.2. Подготовка электронных обучающих ресурсов (ЭОР) в ЭИОС						
{#pageAfternoon1}{0}	{1}	{2}	{3}	{4}	{5}	{6}

**3. НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ РАБОТА**  
(учитывается по второй половине рабочего дня для ВО)

	Вид работы	нормы времени	Осенний семестр		Весенний семестр	
			кол-во часов		кол-во часов	
			по плану	факт. выполн.	по плану	факт. выполн.
1	2	3	4	5	6	7
<b>3.1. Написание и подготовка к изданию монографий, научных статей, докладов, заявок на изобретение</b>						
{#pageAfternoon2}{0}	{1}	{2}	{3}	{4}	{5}	{6}{/}

**4. ОРГАНИЗАЦИОННО-МЕТОДИЧЕСКАЯ РАБОТА**  
(учитывается по второй половине рабочего дня)

	Вид работы	нормы времени	Осенний семестр		Весенний семестр	
			кол-во часов		кол-во часов	
			по плану	факт. выполн.	по плану	факт. выполн.
1	2	3	4	5	6	7
<b>4.1. Участие в проведении работы по профессиональной ориентации молодежи при поступлении в вуз, исполнение обязанностей, проведение олимпиад, конференций и другие организационно-методические работы</b>						
{#pageAfternoon3}{0}	{1}	{2}	{3}	{4}	{5}	{6}{/}

**Перечень опубликованных в учебном году научных и научно-методических работ**

№	Название	Вид публикации (учебник, учебное пособие, монография, методические)	Объем (п.д.)	Издательство	Год
{#pageAfternoon4}{0}	{1}	{2}	{3}	{4}	{5}{/}

**Участие в хозяйственной НИР**  
(сверх 6-часового рабочего дня преподавателя – для ВО)

{#pageAfternoon5}{0}	{1}	{2}	{3}{/}
----------------------	-----	-----	--------

**СВОДНАЯ ТАБЛИЦА**  
нормируемой работы, выполняемой в течение 6-часового рабочего дня  
**на 2017/2018 учебный год**  
(количество занимаемых ставок – 0,5)

{#pageResult0}{0}	{1}	{2}	{3}/{/}
-------------------	-----	-----	---------

Примечание: общий объем работы преподавателя в течение учебного года (строка "всего") должен составлять 1540 часов на одну ставку, при этом объем учебной работы (первая половина дня) не должен превышать 900 часов на одну ставку

План повышения квалификации за учебный год

{#pageResult1}{0}	{1}/{/}
-------------------	---------

Замечания зав. кафедрой и декана о выполнении преподавателем плана работы за учебный год

{#pageResult2}{0}	{1}/{/}
-------------------	---------

Преподаватель ({use{name}}.)

Зав кафедрой (Святков К.В.)

Декан факультета (Святков К.В.)