МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

Гомельский государственный технический университет

имени П.О. Сухого

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

**Специальность 1-40 04 01 «Информатика и технологии программирования»**

**Отчёт по преддипломной практике**

на тему:

«Программный комплекс для автоматизации ведения индивидуального плана преподавателей»

Исполнитель: студент гр. ИП-41

Езепенко Р.О.

Руководитель от предприятия:

Гридина Е.И.

Руководитель от университета:

Ст. преподаватель Романькова Т.Л.

Дата проверки:

Дата допуска к защите:

Дата защиты:

Оценка работы:

Подписи членов комиссии:

Гомель 2023

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc133519851)

[1 Общие сведения об организации 4](#_Toc133519852)

[1.1 История организации 4](#_Toc133519853)

[1.2 Охрана труда и техника безопасности на рабочем месте 5](#_Toc133519854)

[1.3 Технологии, используемые на предприятии, программное обеспечение 7](#_Toc133519855)

[2 Аналитический обзор существующих методов и средств автоматизации ведения индивидуального плана преподавателей 9](#_Toc133519856)

[2.1 Обзор существующих систем автоматизации 9](#_Toc133519857)

[2.2 Постановка задачи 13](#_Toc133519858)

[2.3 Выбор средств реализации серверной части приложения 14](#_Toc133519859)

[2.4 Выбор средств реализации *Android* приложения 16](#_Toc133519860)

[2.5 Выбор средств реализации *WEB* приложения 18](#_Toc133519861)

[2.6 Выбор хранилища данных 19](#_Toc133519862)

[3 Архитектура приложения 23](#_Toc133519863)

[3.1 Моделирование базы данных 23](#_Toc133519864)

[3.3 Структурная схема 32](#_Toc133519865)

[3.4 Функциональная схема 33](#_Toc133519866)

[ЗАКЛЮЧЕНИЕ 35](#_Toc133519867)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 36](#_Toc133519868)

# ВВЕДЕНИЕ

Современные условия работы образовательных учреждений требуют от преподавателей высокой профессиональной компетенции и организованности. Индивидуальные планы преподавателей являются одним из основных инструментов, позволяющих обеспечить контроль и планирование работы каждого преподавателя.

Однако, процесс планирования и контроля работы преподавателей часто остается вручную и может быть трудоемким и неэффективным. Для решения этой проблемы в современном образовании все чаще используются различные автоматизированные системы управления учебным процессом, которые позволяют повысить эффективность работы и оптимизировать расходы.

Цель данной дипломной работы состоит в разработке системы автоматизации индивидуальных планов преподавателей, которая будет облегчать процесс планирования и контроля работы преподавателей, а также повышать эффективность их деятельности. Основными задачами дипломной работы являются:

* анализ существующих систем управления индивидуальными планами преподавателей, или систем, способных выполнять ведения планов;
* разработка требований к системе автоматизации индивидуальных планов преподавателей;
* проектирование и разработка системы автоматизации индивидуальных планов преподавателей;
* тестирование и анализ эффективности работы разработанной системы;
* внедрение системы автоматизации индивидуальных планов преподавателей в учебный процесс университета.

Актуальность темы заключается в том, что индивидуальные планы преподавателей являются неотъемлемой частью управления учебным процессом в современном образовании, а процесс планирования и контроля их работы может быть долгим и неэффективным. Разработка системы автоматизации индивидуальных планов преподавателей поможет ускорить этот процесс и повысить эффективность работы преподавателей.

### **1 ОБЩИЕ СВЕДЕНИЯ ОБ ОРГАНИЗАЦИИ**

### **1.1 История организации**

*EPAM Systems* – американская ИТ-компания, основанная в 1993 году. Производитель заказного [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), специалист по консалтингу, резидент [Белорусского парка высоких технологий](https://ru.wikipedia.org/wiki/%D0%91%D0%B5%D0%BB%D0%BE%D1%80%D1%83%D1%81%D1%81%D0%BA%D0%B8%D0%B9_%D0%BF%D0%B0%D1%80%D0%BA_%D0%B2%D1%8B%D1%81%D0%BE%D0%BA%D0%B8%D1%85_%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D0%B9). Штаб-квартира компании расположена в Ньютона, штат Пенсильвания, а её отделения представлены более чем в 40 странах мира. Компания *EPAM* была основана в [1993 году](https://ru.wikipedia.org/wiki/1993_%D0%B3%D0%BE%D0%B4) двумя одноклассниками [Аркадием Добкиным](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%B1%D0%BA%D0%B8%D0%BD,_%D0%90%D1%80%D0%BA%D0%B0%D0%B4%D0%B8%D0%B9_%D0%9C%D0%B8%D1%85%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D0%B8%D1%87) и Леонидом Лознером. Название компании происходило от «*Effective Programming for America*». Первые офисы были открыты в [США](https://ru.wikipedia.org/wiki/%D0%A1%D0%A8%D0%90) и [Белоруссии](https://ru.wikipedia.org/wiki/%D0%91%D0%B5%D0%BB%D0%BE%D1%80%D1%83%D1%81%D1%81%D0%B8%D1%8F). Позже и в [Австрии](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%81%D1%82%D1%80%D0%B8%D1%8F), [Австралии](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%81%D1%82%D1%80%D0%B0%D0%BB%D0%B8%D1%8F), [Армении](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D0%BC%D0%B5%D0%BD%D0%B8%D1%8F), [Болгарии](https://ru.wikipedia.org/wiki/%D0%91%D0%BE%D0%BB%D0%B3%D0%B0%D1%80%D0%B8%D1%8F), [Великобритании](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BB%D0%B8%D0%BA%D0%BE%D0%B1%D1%80%D0%B8%D1%82%D0%B0%D0%BD%D0%B8%D1%8F), [Венгрии](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BD%D0%B3%D1%80%D0%B8%D1%8F), [Германии](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D1%80%D0%BC%D0%B0%D0%BD%D0%B8%D1%8F), [Индии](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B8%D1%8F), [Ирландии](https://ru.wikipedia.org/wiki/%D0%98%D1%80%D0%BB%D0%B0%D0%BD%D0%B4%D0%B8%D1%8F), [Казахстане](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D0%B7%D0%B0%D1%85%D1%81%D1%82%D0%B0%D0%BD), [Канаде](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BD%D0%B0%D0%B4%D0%B0), [Китае](https://ru.wikipedia.org/wiki/%D0%9A%D0%B8%D1%82%D0%B0%D0%B9%D1%81%D0%BA%D0%B0%D1%8F_%D0%9D%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F_%D0%A0%D0%B5%D1%81%D0%BF%D1%83%D0%B1%D0%BB%D0%B8%D0%BA%D0%B0), [Мексике](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%BA%D1%81%D0%B8%D0%BA%D0%B0), [Нидерландах](https://ru.wikipedia.org/wiki/%D0%9D%D0%B8%D0%B4%D0%B5%D1%80%D0%BB%D0%B0%D0%BD%D0%B4%D1%8B), [ОАЭ](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%B4%D0%B8%D0%BD%D1%91%D0%BD%D0%BD%D1%8B%D0%B5_%D0%90%D1%80%D0%B0%D0%B1%D1%81%D0%BA%D0%B8%D0%B5_%D0%AD%D0%BC%D0%B8%D1%80%D0%B0%D1%82%D1%8B), [Польше](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D1%88%D0%B0), [Сингапуре](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D0%B3%D0%B0%D0%BF%D1%83%D1%80), [Украине](https://ru.wikipedia.org/wiki/%D0%A3%D0%BA%D1%80%D0%B0%D0%B8%D0%BD%D0%B0), [Узбекистане](https://ru.wikipedia.org/wiki/%D0%A3%D0%B7%D0%B1%D0%B5%D0%BA%D0%B8%D1%81%D1%82%D0%B0%D0%BD), [Чехии](https://ru.wikipedia.org/wiki/%D0%A7%D0%B5%D1%85%D0%B8%D1%8F), [Швеции](https://ru.wikipedia.org/wiki/%D0%A8%D0%B2%D0%B5%D1%86%D0%B8%D1%8F), [Швейцарии](https://ru.wikipedia.org/wiki/%D0%A8%D0%B2%D0%B5%D0%B9%D1%86%D0%B0%D1%80%D0%B8%D1%8F). В 2021 году *EPAM Systems* заняла 1804 место в списке *[Forbes Global](https://ru.wikipedia.org/wiki/Forbes_Global_2000" \o "Forbes Global 2000)* [2000](https://ru.wikipedia.org/wiki/Forbes_Global_2000" \o "Forbes Global 2000) и была включена в [*S&P* 500](https://ru.wikipedia.org/wiki/S%26P_500).

К деятельности данной компании относиться:

* ит-консалтинг;
* разработка программного обеспечения;
* интеграция приложений;
* портирование и миграция приложений;
* тестирование программного обеспечения;
* создание выделенных центров разработки на базе *EPAM Systems*;
* разработка цифровых стратегий.

Среди клиентов *EPAM Systems*:

* *sap, Microsoft, Oracle*;
* ренессанс-капитал, Тройка-Диалог, *Barclays* *Capital*;
* *thomson Reuters, London Stock Exchange, МICEX*, Ингосстрах;
* росгосстрах, Ренессанс Страхование, *РОСНО*;
* альфаСтрахование, ИНТАЧ Страхование, *Aviva*;
* *s7 Airlines*, Аэрофлот, *Expedia*;
* «газпром нефть», «Роснефть», «Славнефть», «Концерн Росэнергоатом»;
* *bosch, Colgate-Palmolive, The Coca-Cola Company* и другие.

*EPAM* обладает обширным списком престижных клиентов по всему миру, среди которых многие члены списка *Fortune* 500 и ведущие компании-разработчики программного обеспечения [1].

Благодаря своей инженерной ДНК, многолетнему технологическому опыту и компетенциям в сфере консалтинга, дизайна и инновационных стратегий, *EPAM* тесно сотрудничает со своими клиентами для создания передовых решений, которые превращают сложные бизнес-задачи в реальные бизнес-возможности. Команды *EPAM* работают с заказчиками в более чем 30 странах в Северной Америке, Европе, Азии и Австралии. *EPAM* признана одной из четырех технологических компаний, входящих в список *Forbes* «25 самых быстрорастущих публичных технологических компаний» каждый раз начиная с 2013 года, а также единственным представителем индустрии ИТ-сервисов в списке *Fortune* 2019 «100 самых быстрорастущих компаний».

## **1.2 Охрана труда и техника безопасности на рабочем месте**

Техника безопасности для программиста – это залог долговременной и беспроблемной работы такого специалиста. Техника безопасности программистов регулируется «Инструкцией по охране труда», где все разложено по пунктам и очень подробно описано. Знать ее нужно, если программист работает в большом офисном здании, где к его компьютеру имеют косвенный доступ несколько человек. В этом случае он обязан следовать инструкциям техники безопасности, чтобы не подвергать опасности свое здоровье и здоровье окружающих его коллег. Плюс программист просто обязан знать, как вести себя во время чрезвычайных ситуаций. Охрана труда – система обеспечения безопасности жизни и здоровья работающих в процессе трудовой деятельности, включающая правовые, социально-экономические, организационные, технические, психофизиологические, санитарно-гигиенические, лечебно-профилактические, реабилитационные и иные мероприятия и средства.

Руководством *EPAM Systems* уделяется большое внимание улучшению эргономики рабочих мест, обеспечению гигиены и совершенствованию организации труда, регламентации режимов труда и отдыха.

Вся деятельность в области охраны труда на *EPAM Systems* регламентирована действующим законодательством Республики Беларусь, санитарными нормами и правилами, гигиеническими нормативами, предписаниями надзорных органов.

Важным фактором создания безопасных условий труда является оптимизация организации рабочих мест. Рабочее место, хорошо приспособленное к трудовой деятельности инженера, правильно и целесообразно организованное в отношении пространства, формы, размера, обеспечивает ему удобное положение при работе и высокую производительность труда при наименьшем физическом и психическом напряжении.

При правильной организации рабочего места производительность труда инженера возрастает с 8 до 20 процентов.

Рабочее место для выполнения работ в положении сидя организуется в соответствии с ГОСТ 12.2.032-78 «Система стандартов безопасности труда. Рабочее место при выполнении работ сидя. Общие эргономические требования».

Требования к организации работы при использовании персонального компьютера и организационных средств определяются СанПиН 9-131 РБ 2000 «Гигиенические требования к видеодисплейным терминалам, электронно-вычисли-тельным машинам и организации работы».

Согласно ГОСТ 12.2.032-78, конструкция рабочего места и взаимное расположение всех его элементов должно соответствовать антропометрическим, физическим и психологическим требованиям. Большое значение имеет также характер работы. В частности, при организации рабочего места программиста должны быть соблюдены следующие основные условия:

– оптимальное размещение оборудования, входящего и состав рабочего места;

– достаточное рабочее пространство, позволяющее осуществлять все необходимые движения и перемещения;

– уровень акустического шума не должен превышать допустимого значения.

Рабочая поза сидя вызывает минимальное утомление программиста. Рациональная планировка рабочего места предусматривает четкий порядок и постоянство размещения предметов, средств труда и документации. То, что требуется для выполнения работ чаще, расположено в зоне легкой досягаемости рабочего пространства.

Помещения для работы программиста должны иметь естественное и искусственное освещение.

Искусственное освещение в помещениях эксплуатации ВДТ и ПЭВМ должно осуществляться системой общего равномерного освещения. В административно-общественных помещениях, в случаях преимущественной работы с документами, допускается применение системы комбинированного освещения (к общему освещению дополнительно устанавливаются светильники местного освещения, предназначенные для освещения зоны расположения документов).

Освещенность на поверхности стола в тоне размещения рабочего документа должна быть 300 – 500 лк. Местное освещение не должно создавать бликов на поверхности экрана и увеличивать освещенность экрана более 300 лк.

В качестве источников света при искусственном освещении должны применяться преимущественно люминесцентные лампы.

Конструкция рабочего стола должна обеспечивать оптимальное размещение на рабочей поверхности используемого оборудования с учетом его количества и конструктивных особенностей (размер ВДТ и ПЭВМ, клавиатуры и др.), характера выполняемой работы. При этом допускается использование рабочих столов различных конструкций, отвечающих современным требованиям эргономики.

Конструкция рабочего стула (кресла) должна обеспечивать поддержание рациональной рабочей позы при работе, позволять изменять позу с целью снижения статического напряжения мышц шейно-плечевой области и спины для предупреждения развития утомления.

Тип рабочего стула (кресла) должен выбираться в зависимости от характера и продолжительности работы.

Рабочий стул (кресло) должен быть подъемно-поворотным и регулируемым по высоте и углам наклона сиденья и спинки, а также расстоянию спинки от переднего края сиденья, при этом регулировка каждого параметра должна быть независимой, легко осуществляемой и иметь надежную фиксацию.

Экран видеомонитора должен находиться от глаз пользователя на оптимальном расстоянии 600 – 700 мм, но не ближе 500 мм с учетом размеров алфавитно-цифровых знаков и символов.

Высота рабочей поверхности стола должна регулироваться в пределах 680 – 800 мм, при отсутствии такой возможности высота рабочей поверхности стола должна составлять 725 мм.

Рабочий стол должен иметь пространство для ног высотой не менее 600 мм, шириной – не менее 500 мм, глубиной на уровне колен – не менее 450 мм и на уровне вытянутых ног – не менее 650 мм.

Клавиатуру следует располагать на поверхности стола на расстоянии не менее чем 300 мм от края, обращенного к пользователю или на специальной, регулируемой по высоте рабочей поверхности, отделенной от основной столешницы.

Помимо требований к организации рабочего места СанПиН 9-131 РБ 2000 устанавливает требования к микроклимату рабочей зоны: влажности, температуре, скорости потока воздуха и пр.

## **1.3 Технологии, используемые на предприятии, программное обеспечение**

В компании используются следующие технологии:

* проектирование, разработка и сопровождение:
* приложений для таких платформ как: *Java, .Net, NodeJS, Python, C/C++, Rudy, Cotlin*;
* приложений для извлечения, преобразования, загрузки массивов данных с использованием *ETL*-технологий;
* приложений в среде «облачных» вычислений, таких как: *Amazon Web Services, Microsoft Azure, Google Cloud, Oracle Cloud, Alibaba*;
* приложений для работы с *BigData* и *SemanticWeb: Operational Big* *Data Technologies, Operational Big Data Technologies*;
* приложений для мобильных устройств на различных платформах: *Windows, Linux, IOS, MacOS, Android*;
* приложений с использованием концепции *SPA (Single Page Application)* с используя *Angular, React JS* фреймворки;
* приложений на языке *ABAP* для систем *SAP*;
* продуктов Интернет и *e-business*.
* Десктопных продуктов с помощью с применением: *Windows Presentation Foundation, Windows Forms*;
* создание и развитие интегрированных систем управления предприятием на базе решений *SAP;*
* миграция программных продуктов между разнообразными платформами и средами;
* разработка мобильный приложений;
* разработка *AI & Machine Learning;*
* разработка мобильных игр, с помощью *.Net, Unity*;
* проектирование и администрирование баз данных *(DB2, Oracle, MSSQL Server, DynamoDb, MognoDb, Cassandra,* *Postgresql, MySQL, Amazon RDS, CouchDB);*
* проектирование *e-business* приложений: [*Content management systems*](https://www.templafy.com/blog/marketing-asset-management-is-your-key-2018-investment/)*, Customer relationship management systems, Accounting software, Business intelligence, Cloud software, Enterprise resource planning;*
* разработка автоматизированных процессов непрерывной интеграции и непрерывного развертывания: *Jenkins, CirlceCI, TeamCity, Bamboo, GitLab, GitHud Actions;*
* банковские технологии: *Digital Bank, Account Creation, Application Programming Interfaces, Cloud Computing, Video Conferencing, Blockchain, Person-To-Person Payments.*

# 2 АНАЛИТИЧЕСКИЙ ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ И СРЕДСТВ АВТОМАТИЗАЦИИ ВЕДЕНИЯ ИНДИВИДУАЛЬНОГО ПЛАНА ПРЕПОДАВАТЕЛЕЙ

# 

**2.1 Обзор существующих систем автоматизации**

Многие университеты создают свои локальные программы для автоматизации необходимых процессов. Такой подход, создания многих небольших программ, позволяет быстро разработать приложение, которое будет выполнять необходимые задачи. В случае необходимости оперативно устранить неполадки или добавлять необходимый новый функционал. Все эти программы связаны с серверной частью университета, которая выражена в виде локальной СУБД, со всей необходимой информацией для комплекса.

Информационно-программный комплекс «*UNIVER*» – представляет собой систему администрирования и управления учебной деятельностью университета с полным циклом охвата учебного процесса. Пример пользовательского интерфейса «*UNIVER*» представлен на рисунке 2.1.

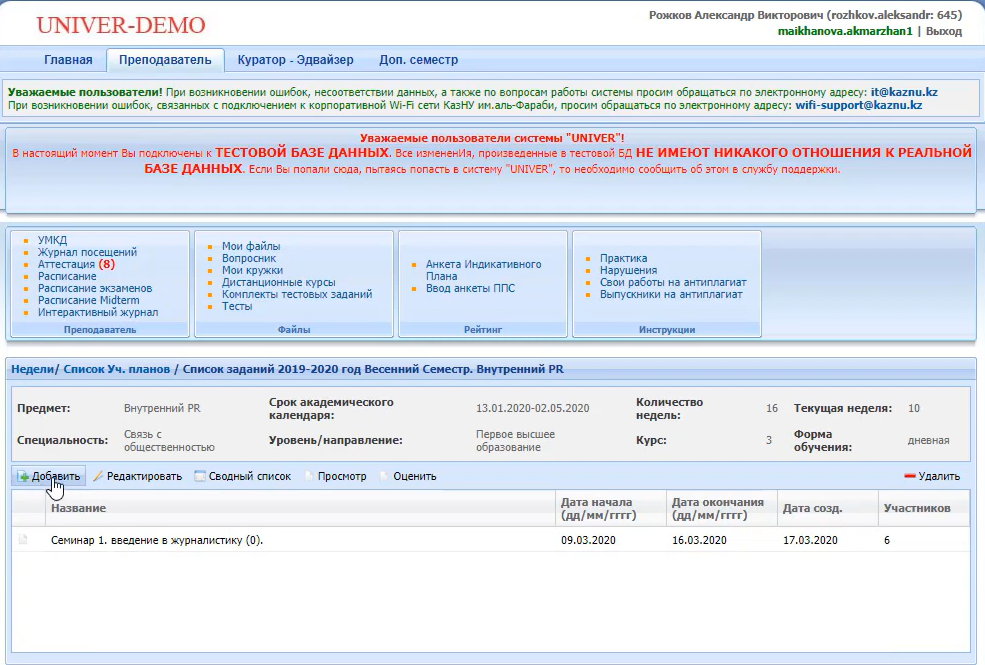


Рисунок 2.1 – Пример пользовательского интерфейса «*UNIVER*»

К преимуществам данной системы создатели относят:

* формирование фундамента для построения самоорганизующейся системы управления вузом, который помогает создать эффективную систему управления университетом, учета и отчетности;
* повышение оперативности и качества принятия управленческих решений, а также совершенствование контроля за их исполнением;
* осуществление управления учебной деятельностью: формирование учебных планов, аттестация и т.д.;
* управление и организация электронного учебного документооборота на весь технологический цикл учебного процесса [2].

Недостатками данной системы выступают:

* высокая стоимость: данная система является довольно дорогостоящей, что не располагает к приобретению данного продукта;
* избыточный функционал: система «*UNIVER*» имеет широкий спектр предоставляемого функционала, что является избыточным для поставленной цели.

Система автоматизации делопроизводства и документооборота «Дело» – разработана компанией [ЭОС](https://www.tadviser.ru/index.php/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D0%B8%D1%8F:%D0%AD%D0%9E%D0%A1), с полным набором инструментов для автоматизации работы с документами, задачами и процессами. Позволяет оптимизировать документооборот, повысить его эффективность, снизить долю рутинного труда при обработке документов за счет использования технологий и сервисов с элементами искусственного интеллекта [3]. Пример пользовательского интерфейса «Дело» представлен на рисунке 2.2.

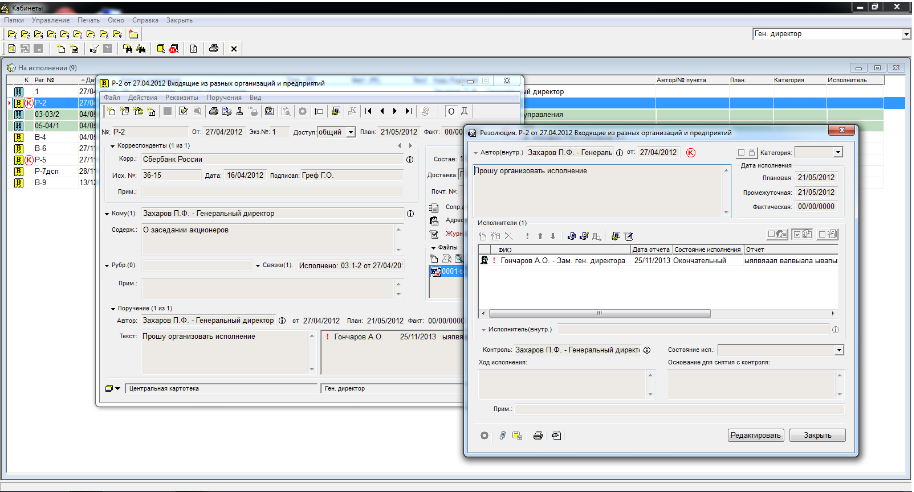


Рисунок 2.2 – Пример пользовательского интерфейса «Дело»

Система «Дело» обладает преимуществами:

* универсальность, которая выражается в неограниченности по сфере деятельности и размерам организации, также система рассчитана на максимальные нагрузки как по объему базы данных, так и по количеству пользователей;
* удобство: система легко адаптируема к специфике сложившихся в организации процессов. Интерфейс создан с учетом актуальных потребностей пользователей, постоянно совершенствуется и прост в освоении;
* расширенная функциональность: система относится к сегменту СЭД/[ECM](https://www.tadviser.ru/index.php/%D0%A1%D0%AD%D0%94), но ее функционал позволяет решать задачи за рамками классического документооборота; Наличие дополнительных опций, специализированных приложений и сервисов делает возможной автоматизацию широкого спектра [бизнес-процессов](https://www.tadviser.ru/index.php/BPM);
* индивидуальный пользовательский интерфейс: для пользователей доступна персонализация рабочего пространства – настройки главной страницы, фильтров, поисковых запросов и личных папок, карточки документов и их атрибутный состав также легко адаптировать под потребности пользователей;
* интеграционные возможности: система обладает открытым [*API*](https://www.tadviser.ru/index.php/API), что позволяет реализовать взаимодействие с другими информационными системами; поддерживает бесшовное взаимодействие при работе с документами между СЭД и офисными приложениями (МойОфис, *[Microsoft Office](https://www.tadviser.ru/index.php/%D0%9F%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82:Microsoft_Office" \o "Microsoft Office)*); взаимодействие с почтовым клиентом; реализованы возможности регистрации документа непосредственно из текстового редактора и редактирование файла документа из интерфейса СЭД.

К недостаткам следует отнести:

* стоимость: «Дело» является довольно дорогой системой, что может не подойти для многих университетов;
* сложность внедрения: данная система сложна во внедрении, поскольку обладает обширным функционалом и требует квалифицированных специалистов;
* необходимость квалифицированных сотрудников: использование системы требует подготовки персонала.

АСМО-Документооборот. Программный комплекс предназначен для комплексной автоматизации процессов организации делопроизводства на предприятии, работы с электронными документами, включая процесс их создания, изменения, согласования, управления доступом к ним, ведения жизненного цикла, а также контроля их исполнения. Поддерживается механизм акцептования с помощью электронно-цифровой подписи (ЭЦП). Пример пользовательского интерфейса комплекса представлен на рисунке 2.3.

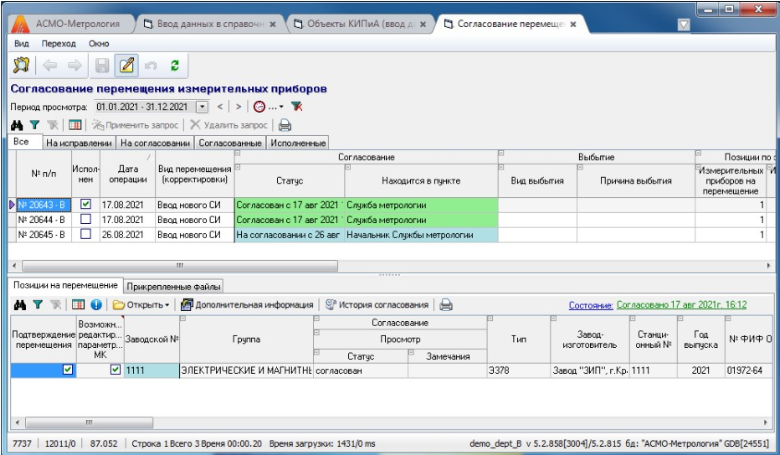


Рисунок 2.3 – Пример пользовательского интерфейса «АСМО-Документооборот»

Преимущества данной системы:

* регистрация и хранение данных о входящей, исходящей, внутренней корреспонденции, организационно-распорядительной документации;
* автоматическое прикрепление распознанных копий документов;
* ведение полного жизненного цикла и взаимосвязей документов;
* поиск документов с использованием механизмов полнотекстового и контекстного поиска;
* делегирование прав и замещение сотрудников при работе с документами;
* ведение параллельного и последовательного согласования документов;
* настройка маршрутов и участников согласования документов;
* формирование поручений и контроль их исполнения для всех типов документов;
* информирование участников процесса о ходе согласования и исполнения поручений;
* ведение личного кабинета с формированием входящих и исходящих задач пользователя;
* ведение избранных документов пользователя;
* настройка шаблонов почтовых, мгновенных сообщений и задач личного кабинета;
* формирование отчетности;
* настройка и предоставление доступа к каждому документу в соответствии с матрицей прав доступа [4].

К недостаткам комплекса относятся:

* обобщенность: данная особенность может выступать, как и преимуществом системы, так и недостатком, так как требуется дополнительная настройка приложения, под необходимые задачи университета;
* высокая стоимость использования, что важно в сфере образования;
* излишний функционал, который усложняет использование и повышает входной порог для пользователей.

Таким образом, после произведенного анализа существующих систем, можно сделать вывод о том, что некоторые системы являются обобщенным решением, которые не учитывают специфики заданной предметной области, что является серьезным недостатком, а также можно выделить следующие требования к комплексу:

* высокая стоимость использования: система не должна быть дорогой в обслуживании и дальнейшем сопровождении;
* надежность: комплекс должен быть отказоустойчивой, способной функционировать без необходимости в постоянной поддержке;
* отсутствие избыточного функционала, который усложняет использование и повышает входной порог для пользователей системы.

## **2.2 Постановка задачи**

Перед разработкой программного обеспечения необходимо сформировать список требований, которые должны быть реализованы. К функциональным требованиям относятся:

* предоставление доступа для просмотра и редактирования собственных индивидуальных планов;
* автоматическая генерация учебно-методической нагрузки;
* помощь при генерации или редактировании компонентов индивидуального плана, таких как: учебно-методическая деятельность, научно-исследовательская деятельность, идеологическая работа и т.д.;
* возможность генерации конечного плана в *PDF* формате;
* отправка уведомлений пользователю по средства электронной почты, или мобильного приложения;
* сбор и анализ статистики индивидуальных планов;
* организация ролевого доступа к приложению;
* наличие система авторизации и аутентификации.

К нефункциональным требованиям относится:

* предоставление платформенной независимости;
* предоставление поддержки контейнеризации;
* проведение нагрузочного тестирования;
* локализация приложения;
* разработка *WEB UI* и *Android Application*;
* проработка пользовательского интерфейса, с высоким уровнем *UX*;
* придерживание выбранной общей стилизации системы.

Создание серверной части при помощи платформы *.NET Core* с использованием технологии *ASP.NET Core*. Для разработки клиентской часть выбран *React*, а также мобильное *Android* приложение, разработанное с помощью языка программирования *Java*.

**2.3 Выбор средств реализации серверной части приложения**

Исходя из поставленных нефункциональных требований к системе, для создания серверной части системы, был выбран объектно-ориентированный язык программирования *C#* под управлением платформы *.NET Core 6*. Ниже приведены основные преимущества использования платформы *.NET* и языка программирования C#:

* кроссплатформенность: *.NET* поддерживает разработку приложений для различных операционных систем, включая *macOS*, *Linux* и *Windows*, что позволяет разработчикам создавать приложения, которые могут быть запущены на разных устройствах и операционных системах, что увеличивает охват аудитории;
* обширный набор библиотек: *.NET* имеет обширный набор библиотек классов, который включает в себя множество функций и инструментов, упрощающих разработку приложений; это уменьшает время разработки, сокращает количество ошибок и улучшает качество кода;
* безопасность: платформа имеет встроенные механизмы безопасности, которые обеспечивают защиту приложения от злоумышленников, а также, *C#* является типизированным языком, что обеспечивает проверку типов и сокращает вероятность ошибок, связанных с типами данных;
* высокая производительность: *.NET* и *C#* обеспечивают высокую производительность приложений, благодаря компиляции в машинный код и оптимизации при выполнении;
* легкость обслуживания: *.NET* и *C#* обладают высокой читабельностью и поддерживают объектно-ориентированный подход, что упрощает поддержку и сопровождение приложений;
* интеграция: *.NET* может интегрироваться с различными технологиями и платформами, такими как базы данных, веб-сервисы, месседж брокеры и другие; это обеспечивает множество возможностей для создания интегрированных приложений и систем;
* сообщество: *.NET* и *C#* имеют широкое сообщество разработчиков, которое обеспечивает доступ к различным ресурсам, включая библиотеки, фреймворки, инструменты, что облегчает разработку приложений и повышает качество кода;
* масштабируемость: *.NET* и *C#* обеспечивают высокую масштабируемость приложений, благодаря поддержке распределенных вычислений, приложения могут масштабироваться горизонтально, то есть добавлением дополнительных серверов и узлов; это позволяет увеличивать производительность приложения и обеспечивать его работоспособность при росте нагрузки;
* многофункциональность: *.NET* поддерживает множество технологий и возможностей, включая разработку настольных приложений, мобильных приложений, веб-приложений и многих других, данная особенность обеспечивает гибкость и универсальность в разработке приложений;
* поддержка интегрированной среды разработки: *C#* поддерживает множество интегрированных сред разработки, таких как *Visual Studio*, *JetBrains Rider, Visual Studio Code* и другие; это облегчает процесс разработки, обеспечивает поддержку отладки, автодополнения и других функций, ускоряющих разработку приложений [5, c. 16].

В целом, платформа *.NET* и язык программирования *C#* обеспечивают разработчикам широкие возможности для создания высококачественных, безопасных и масштабируемых приложений для различных операционных систем и платформ.

Для разработки *WEB API* системы, выбрана технология *ASP.NET Core* – это кроссплатформенный веб-фреймворк, разработанный *Microsoft*, который позволяет разработчикам создавать веб-приложения и *WEB API*. *ASP.NET Core* предлагает ряд особенностей и преимуществ, среди которых:

* кроссплатформенность: *ASP.NET Core* является кроссплатформенным фреймворком, который может использоваться для создания приложений на разных операционных системах;
* модульность: фреймворк основан на модульной архитектуре, что позволяет разработчикам выбирать только необходимые компоненты для своих приложений, что позволяет сократить размер приложения и улучшить его производительность;
* высокая производительность: *ASP.NET Core* имеет оптимизированную архитектуру, которая обеспечивает высокую производительность при обработке запросов и отправке ответов, а также поддерживает асинхронное выполнение кода, что позволяет более эффективно использовать ресурсы сервера;
* поддержка облака: *ASP.NET Core* поддерживает облачные платформы, такие как *Azure, Amazon WEB Services, Google Cloud* и прочие, что облегчает развертывание приложений в облаке и управление ими;
* безопасность: фреймворк предлагает ряд механизмов безопасности, таких как аутентификация и авторизация, встроенные по умолчанию; это обеспечивает защиту от несанкционированного доступа и повышает уровень безопасности приложения;
* легкость развертывания: *ASP.NET Core* поддерживает различные способы развертывания приложений, включая развертывание в контейнерах *Docker*, что облегчает процесс развертывания и управления приложением [5, c. 314];
* открытый исходный код: фреймворк является проектом с открытым исходным кодом, что позволяет разработчикам сотрудничать, улучшать и оптимизировать данную технологию.

Исходя из проведенного обзора технологий, можно сделать вывод о том, что платформа *.NET Core* и язык программирования *C#*, со всеми перечисленными технологиями, позволяют реализовать серверную часть комплекса.

**2.4 Выбор средств реализации *Android* приложения**

Для разработки *Android* приложения был выбран язык программирования *Java*, потому, как *Java* – это язык программирования, широко используемый в разработке мобильных приложений на платформе *Android* [6, c. 95]. Пример технологий, используемые в *Java* для *Android*-разработки:

* *Android SDK*: это пакет разработчика, содержащий все необходимые инструменты для создания приложений на *Android*, такие как *Android Studio*, архивы библиотек и документацию;
* *Java Virtual Machine*: это среда выполнения *Java*, которая позволяет исполнять код *Java* на устройстве *Android*;
* *Dalvik Virtual Machine*: это виртуальная машина, используемая *Android* для запуска приложений, написанных на *Java*;
* *Eclipse IDE*: это интегрированная среда разработки, которая позволяет создавать, отлаживать и тестировать приложения;
* *Gradle*: это инструмент сборки проектов, который используется в *Android Studio* для сборки и управления зависимостями в приложении.

Преимущества использования *Java* для *Android*-разработки:

* кроссплатформенность: приложения на данном языкемогут быть запущены на большинстве устройств с операционной системой *Android*, что означает, что разработчикам не нужно создавать приложения для каждого устройства;
* богатая библиотека: *Java* имеет широкий спектр библиотек и фреймворков, что делает разработку приложений проще и быстрее;
* удобство отладки: данный язык имеет инструменты отладки, которыми обладают *Eclipse* и *Android Studio*, которые помогают быстро находить и устранять ошибки в приложениях;
* высокая производительность: приложения на *Java* работают достаточно быстро и плавно на устройствах *Android*;
* безопасность: *Java* имеет множество функций безопасности, которые помогают защитить приложения от взлома и кражи данных;
* поддержка многопоточности: *Java* имеет интегрированную поддержку многопоточности, что позволяет разработчикам создавать приложения, которые могут одновременно выполнять несколько задач;
* крупное сообщество: существует огромное сообщество разработчиков, которые используют *Java* для создания приложений на *Android*, что означает, что всегда можно найти помощь и поддержку в случае возникновения проблем;
* открытый исходный код: язык, как и его платформа является может отличится от других аналогов открытым исходным кодом, что позволяет разработчикам легко получить доступ к исходному коду и вносить необходимые изменения;
* интеграция со сторонними сервисами: используя данный язык, разработчики могут легко интегрировать свои приложения со сторонними сервисами, такими как *Google Maps*, *Facebook* и *Twitter*;
* большое количество готовых решений: существует множество готовых библиотек и фреймворков, которые помогают ускорить разработку приложений на *Java*, некоторые из них, такие как *Retrofit* и *ButterKnife*, позволяют существенно сократить количество кода, написанного вручную.

В целом, использование *Java* для *Android*-разработки позволяет создавать высококачественные приложения с высокой производительностью и безопасностью. Богатый набор инструментов и библиотек, а также поддержка многопоточности и расширяемости, делают процесс разработки более гибким и эффективным. Однако, существует также конкурирующая платформа - *.NET MAUI*, которая использует язык программирования *C#* для создания кроссплатформенных приложений, работающих на разных операционных системах, включая *Android*. Несмотря на то, что *.NET MAUI* предлагает ряд преимуществ, которые могут заинтересовать разработчиков, *Java* остается лучшим выбором для разработки *Android*-приложений по следующим причинам:

* широкая экосистема и богатые инструменты: *Java* имеет широкую экосистему, которая охватывает различные инструменты и библиотеки, позволяющие создавать более качественные и сложные приложения; существуют множество сторонних библиотек, которые можно использовать, чтобы упростить процесс разработки, а также платформы, а с другой стороны, *.NET MAUI*, как новая технология, еще не имеет такой широкой экосистемы и инструментов, как *Java*;
* интеграция с *Android*-платформой: *Android* был создан на основе языка программирования *Java*, поэтому эти две технологии имеют тесную интеграцию друг с другом; это означает, что при использовании данного языка можно легко использовать *Android API* и многие другие компоненты платформы, что упрощает создание приложений для *Android*, c другой стороны, *.NET MAUI* требует дополнительных шагов для интеграции с *Android*, что может увеличить время разработки и сложность проекта;
* сообщество разработчиков и обучающие ресурсы: *Java* является одним из самых популярных языков программирования в мире, поэтому у него есть огромное сообщество разработчиков и множество обучающих ресурсов, которые могут помочь начинающим разработчикам; с другой стороны, *.NET MAUI* имеет более узкое сообщество разработчиков, что ограничивает доступ к опыту и знаниям, которые могут помочь при создании сложных приложений [6, c. 140].

Исходя из данных особенностей, разработка приложения для *Android* была выполнена с помощью языка программирования *Java*.

**2.5 Выбор средств реализации *WEB* приложения**

Для разработки *WEB* интерфейса, выбран язык программирования *JavaScript*, с использованием *React* – это библиотека, которая используется для создания пользовательских интерфейсов. Она была создана *Facebook* и используется для разработки веб-приложений, мобильных приложений и настольных приложений. *React* является одной из самых популярных библиотек для разработки пользовательского интерфейса и обладает рядом преимуществ перед другими инструментами:

* компонентный подход: *React* использует компонентный подход к разработке пользовательского интерфейса, а означает, что каждый компонент может быть переиспользован, что упрощает создание сложных пользовательских интерфейсов;
* декларативный подход: *React* использует декларативный подход к разработке пользовательского интерфейса, что позволяет разработчикам описывать, как должен выглядеть интерфейс, а не описывать каждое действие, которое должно происходить при взаимодействии пользователя с интерфейсом;
* *Virtual DOM*: *React* использует *Virtual DOM*, что позволяет обновлять только необходимые элементы пользовательского интерфейса вместо полной перерисовки страницы; это уменьшает нагрузку на браузер и увеличивает производительность приложения;
* поддержка серверного рендеринга: данная технология позволяет выполнять рендеринг пользовательского интерфейса на сервере, что улучшает производительность и оптимизирует работу с *SEO*;
* использование *JSX*: фреймворк использует *JSX* – язык, который позволяет описывать структуру пользовательского интерфейса внутри *JavaScript* кода;
* большое сообщество: *React* имеет большое сообщество разработчиков, которые создают множество инструментов и библиотек для упрощения разработки;
* использование *Flux* архитектуры: *React* использует *Flux* архитектуру для управления состоянием приложения, что позволяет разработчикам создавать более простые и понятные приложения;
* поддержка мобильных приложений: *React Native* – это фреймворк, основанный на *React*, который позволяет разработчикам создавать мобильные приложения для *IOS* и *Android*.

В целом, использование *React* позволяет разработчикам создавать масштабируемые и гибкие пользовательские интерфейсы, что делает его одним из наиболее популярных инструментов для разработки веб-приложений.

# 2.6 Выбор хранилища данных

Для определения подходящего постоянного хранилища данных необходимо выбрать тип СУБД - *SQL* или *NoSQL*.

Реляционная Система Управления Базами Данных – это тип СУБД, основанный на модели реляционных данных, где данные хранятся в таблицах, которые могут быть связаны друг с другом посредством ключей. *Microsoft SQL Server* – это реляционная СУБД, разработанная *Microsoft*.

Одним из основных преимуществ *MSSQL* является ее масштабируемость. Она может работать с огромными базами данных, поддерживая сотни тысяч одновременных запросов. Кроме того, она обладает высокой производительностью и скоростью выполнения запросов благодаря использованию оптимизации запросов и кэшированию данных.

*MSSQL* также имеет широкий спектр инструментов для управления базами данных и администрирования, таких как *SQL Server Management Studio* (*SSMS*), *SQL Server Data Tools* (*SSDT*) и другие. Она также поддерживает широкий спектр языков программирования, включая *C#*, *Java*, *Python* и другие, что делает ее удобной для разработки и интеграции с различными приложениями.

Другое преимущество *MSSQL* заключается в ее высокой надежности и защищенности. Она обеспечивает функции безопасности, такие как шифрование данных, аудит и контроль доступа, что делает ее надежной для хранения конфиденциальной информации и данных.

Таким образом, *MSSQL* – это мощная и надежная реляционная СУБД, которая обладает широкими возможностями для управления и администрирования баз данных, а также обеспечивает высокую производительность, безопасность и масштабируемость.

Кроме того, *MSSQL* поддерживает транзакционность, что означает, что операции с базой данных могут быть выполнены либо полностью, либо не выполнены вообще, что гарантирует целостность и надежность данных. *MSSQL* также поддерживает репликацию данных, что позволяет создавать резервные копии данных и повышать доступность системы. Кроме того, *MSSQL* является кроссплатформенной СУБД и может работать на различных операционных системах, таких как *Windows*, *Linux* и *Docker* [7, c. 56].

Также реляционные СУБД соответствуют требованиям *ACID* – это сокращение от английских слов *Atomicity*, *Consistency*, *Isolation* и *Durability* (атомарность, согласованность, изолированность и устойчивость). Это концепция, которая описывает свойства транзакций в реляционных базах данных.

Атомарность означает, что транзакция является неделимой операцией. Она может быть либо полностью выполнена, либо отменена, но не может быть выполнена частично. Если одна из операций в транзакции не может быть выполнена, то все операции в транзакции отменяются и база данных возвращается в исходное состояние.

Согласованность гарантирует, что транзакция должна сохранять целостность базы данных. Если транзакция не выполняется, данные должны оставаться в согласованном состоянии.

Изолированность гарантирует, что каждая транзакция должна работать в изолированном режиме, что означает, что операции в одной транзакции не должны быть видны другим транзакциям, пока они не завершат свою работу.

Устойчивость гарантирует, что после успешного завершения транзакции изменения будут сохранены навсегда и не будут потеряны даже в случае сбоя системы.

*ACID* является важным свойством реляционных баз данных, потому что гарантирует надежность и целостность данных. Однако, *ACID* требует значительных затрат на ресурсы, что может снижать производительность системы при обработке больших объемов данных. Некоторые СУБД, такие как *NoSQL*, предлагают более гибкие модели хранения данных, не следующие полностью концепции *ACID*, но обеспечивающие высокую производительность и масштабируемость при обработке больших объемов данных [8].

*NoSQL* – это подход к реализации масштабируемого хранилища информации с гибкой моделью данных, отличающийся от классических реляционных СУБД. В нереляционных базах проблемы масштабируемости и доступности, решаются за счёт атомарности и согласованности данных. *NoSQL* – базы оптимизированы для приложений, которые должны быстро, с низкой временной задержкой обрабатывать большой объем данных с разной структурой. Сам термин *NoSQL* обозначает «не только *SQL*», характеризуя ответвление от традиционного подхода к проектированию баз данных. Изначально так называлась база данных, с открытым исходным кодом, которая хранила все данные как *ASCII*-файлы, а вместо *SQL*-запросов доступа к данным использовались шелловские скрипты. В начале 2000-х годов *Google* построил свою поисковую систему и приложения: *GMail, Maps, Earth* и прочие сервисы, решив проблемы масштабируемости и параллельной обработки больших объёмов данных. Так была создана распределённые файловая и координирующая системы, а также колоночное хранилище, основанное на вычислительной модели *[MapReduce](https://www.bigdataschool.ru/wiki/mapreduce" \t "_blank)*. Все *NoSQL* решения принято делить на 4 типа:

Ключ-значение – вариант хранилища данных, использующий ключ для доступа к значению в рамках большой хэш-таблицы. Такие СУБД применяются для хранения изображений, создания специализированных файловых систем, в качестве кэшей для объектов. Наиболее известными представителями считаются *Oracle NoSQL* *Database, Berkeley DB, Amazon DynamoDB*, которые поддерживают высокую разделяемость, обеспечивая беспрецедентное горизонтальное масштабирование, недостижимое при использовании других типов БД.

Документно-ориентированное – это хранилище, в котором данные, представленные парами ключ-значение, сжимаются в виде полу структурированного документа из тегированных элементов, подобно *JSON, XML, BSON* и другим подобным форматам. Такая модель хорошо подходит для каталогов, пользовательские профилей и систем управления контентом, где каждый документ уникален и изменяется со временем. Самые яркие примеры документно-ориентированных нереляционных баз данных – это *CouchDB, Couc*[*hbase*](https://www.bigdataschool.ru/wiki/hbase)*, MongoDB, eXis* [9].

Колоночное хранилище, которое хранит информацию в виде разреженной матрицы, строки и столбцы которой используются как ключи. В мире *Big Data* к колоночным хранилищам относятся базы типа «семейство столбцов». В таких системах сами значения хранятся в столбцах, представленных в отдельных файлах. Благодаря такой модели данных можно хранить большое количество атрибутов в сжатом виде, что ускоряет выполнение запросов к базе, особенно операции поиска и агрегации данных. Самой известной колоночной базой данных является *Google Big Table*, а также основанные на ней *Apache HBase* и *[Cassandra](https://www.bigdataschool.ru/wiki/cassandra" \t "_blank)*.

Хранилище в виде графов представляют собой сетевую базу, которая использует узлы и рёбра для отображения и хранения данных. Поскольку рёбра графа являются хранимыми, его обход не требует дополнительных вычислений. При этом для нахождения начальной вершины обхода необходимы индексы. Обычно графовые СУБД поддерживают *ACID*-требования и специализированные языки запросов. Примеры графовых баз: *InfoGrid, Neo4j, Amazon Neptune, OrientDB.*

Выбор в пользу реляционной базы данных подходит в тех случаях, когда важна целостность данных, а также устойчивость к аномалиям в соответствии со степенью нормализации, однако в случае если ожидается большое количество запросы на чтение данных, то будет тратиться большое количество ресурсов, как вычислительных, так и временных. Вследствие чего следует рассмотреть использование документных баз данных. В этом случае, выбор такого вида СУБД, позволит существенно ускорить время выполнения запросов на получение данных.

Индивидуальный план по преподавателям имеет хорошо структурированный формат, что идеально подходит для использования реляционной СУБД, такой как *MSSQL*. Основные преимущества использования реляционной СУБД на примере *MSSQL* включают:

* структурированные данные: если у нас есть структурированные данные, которые нужно хранить и обрабатывать, то реляционная СУБД, такая как *MSSQL*, является оптимальным выбором; реляционные базы данных хранят данные в виде таблиц, которые могут быть легко связаны между собой, что облегчает работу с данными;
* большой объем данных: если имеется большой объем данных, который нужно хранить и обрабатывать, то *MSSQL*, может обрабатывать эти данные очень быстро и эффективно, данная СУБД имеет мощные инструменты для оптимизации производительности базы данных, такие как индексы, кластеризация и разделение таблиц;
* сложные запросы: если имеется необходимость выполнять сложные запросы к базе данных, то *MSSQL* предоставляет мощные инструменты для создания этих запросов; *SQL* – это мощный язык запросов, который позволяет пользователям выбирать, отображать и изменять данные по своему усмотрению.
* безопасность: если необходима высокая степень безопасности для ваших данных, то *MSSQL*, предоставляет мощные механизмы безопасности, включая аутентификацию и авторизацию, что позволяет управлять доступом к базе данных и защищать данные от несанкционированного доступа;
* поддержка: если необходима высокая степень поддержки и помощи в решении проблем, то данная база данных имеет обширную документацию и большое сообщество пользователей, что обеспечивает высокую степень поддержки.

Реляционные СУБД, такие как *MSSQL*, хорошо подходят для приложений, где данные имеют структурированный формат, и когда необходимо быстро и эффективно обрабатывать большие объемы данных. *MSSQL* предоставляет мощные инструменты для оптимизации производительности и безопасности, а также имеет обширную документацию и большое сообщество, к которому, в случае необходимости, можно обратиться за поддержкой.

## **3 АРХИТЕКТУРА ПРИЛОЖЕНИЯ**

**3.1 Моделирование базы данных**

Модель базы данных – это абстрактное представление о том, как данные будут организованы и храниться в базе данных. Она определяет структуру базы данных, типы данных, связи между таблицами и другие характеристики, которые определяют, каким образом данные могут быть добавлены, извлечены и изменены в базе данных. Реляционная модель базы данных, которая является наиболее распространенной, представляет данные в виде таблиц, связанных друг с другом по ключам. Модель базы данных, которая отображает информацию об индивидуальных планах преподавателей представлена на рисунке 3.1:

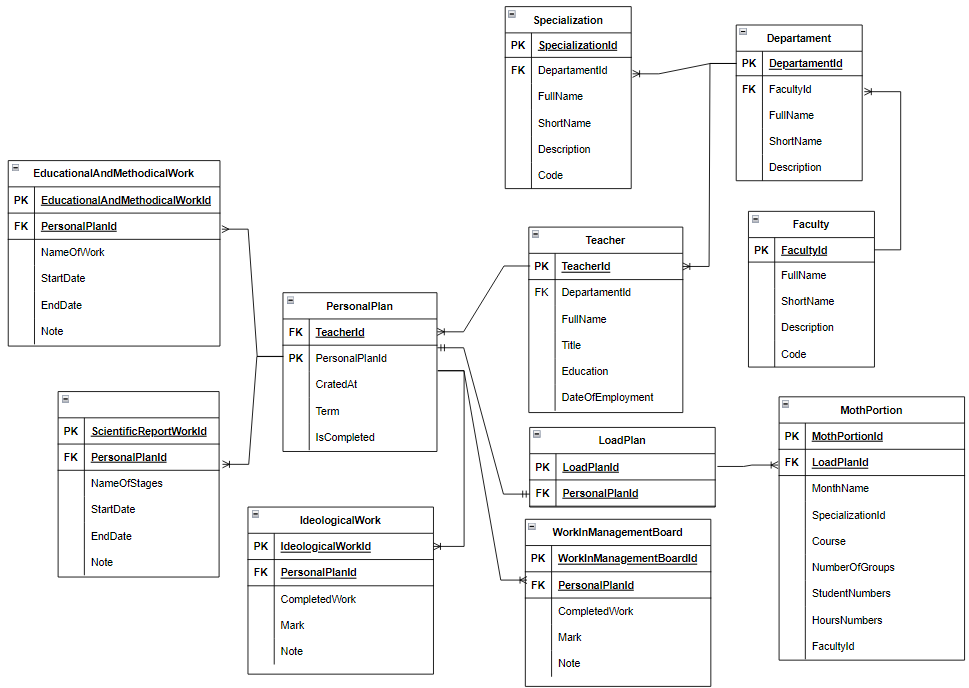


Рисунок 3.1 – Модель базы данных, отображающая информацию об индивидуальных планах преподавателей

Описание таблиц базы данных *MSSQL* приведено в таблице 3.1

Таблица 3.1 – Описание таблиц базы данных *MSSQL*.

|  |  |
| --- | --- |
| Название таблицы | Описание |
| *Specialization* | Хранит информацию о специальности группы |
| *Department* | Хранит информацию о кафедре |
| *Faculty* | Хранит информацию о факультете |
| *Teachers* | Хранит информацию о преподавателях |
| *PersonalPlan* | Хранит информацию об индивидуальных планах преподавателей |
| *EducationalAndMethodicalWork* | Хранит информацию об учебно-методической работе |
| *ScientificReportWork* | Хранит информацию о научно-исследовательской деятельности |
| *IdeologicalWork* | Хранит информацию об идеологической деятельности |
| *WorkInManagementBoard* | Хранит информацию о работе в совете, комиссии, комитете управления |
| *LoadPlan* | Хранит общую информацию о нагрузочном плане преподавателя |
| *MothPortion* | Хранит информацию о нагрузочном плане преподавателя за конкретный месяц |

Описание атрибутов таблицы «*Specialization*» приведено в таблице 3.2.

Таблица 3.2 – Описание атрибутов таблицы «*Specialization*».

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *SpecializationId* | *Int* | Идентификатор специальности | Да | Нет |
| *DeparamentId* | *Int* | Идентификатор  Кафедры | Нет | Да |
| *FullName* | *String* | Название специальности | Нет | Нет |
| *Code* | *String* | Код специальности | Нет | Нет |

Описание атрибутов таблицы «*Deparment*» приведено в таблице 3.3.

Таблица 3.3 – Описание атрибутов таблицы «*Deparment*».

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *DeparmentId* | *Int* | Идентификатор кафедры | Да | Нет |
| *FacultyId* | *Int* | Идентификатор  факультета | Нет | Да |
| *FullName* | *String* | Название кафедры | Нет | Нет |
| *ShortName* | *String* | Сокращенное название кафедры | Нет | Нет |
| *Desctiprtion* | *String* | Описание кафедры | Нет | Нет |

Описание атрибутов таблицы «*Faculty*» приведено в таблице 3.4.

Таблица 3.4 – Описание атрибутов таблицы «*Faculty*».

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *FacultyId* | *Int* | Идентификатор факультета | Да | Нет |
| *Code* | *Int* | Код факультета | Нет | Нет |
| *ShortName* | *String* | Сокращённое название факультета | Нет | Нет |
| *FullName* | *String* | Название факультета | Нет | Нет |
| *Desctiprtion* | *String* | Описание факультета | Нет | Нет |

Описание атрибутов таблицы «*Teacher*» приведено в таблице 3.5.

Таблица 3.5 – Описание атрибутов таблицы «*Teacher*».

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *TeacherId* | *Int* | Идентификатор преподавателя | Да | Нет |
| *DepartmentId* | *Int* | Идентификатор кафедры, к которой относится преподаватель | Нет | Да |
| *FullName* | *String* | ФИО преподавателя | Нет | Нет |
| *Education* | *Int* | Ученая степень | Нет | Нет |
| *Title* | *String* | Должность преподавателя | Нет | Нет |

Описание атрибутов таблицы «*PersonalPlan*» приведено в таблице 3.6.

Таблица 3.6 – Описание атрибутов таблицы «*PersonalPlan*».

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *PersonalPlanId* | *Int* | Идентификатор плана | Да | Нет |
| *TeacherId* | *Int* | Идентификатор преподавателя, к которому относиться данный индивидуальный план | Нет | Да |
| *CreatedAt* | *Datetime* | Дата создания отчета, когда он полностью готов | Нет | Нет |
| *Term* | *Int* | Номер семестра | Нет | Нет |
| *IsCompleted* | *Boolean* | Флаг, который указывает завершенность плана | Нет | Нет |

Описание атрибутов таблицы «*EducationAndMethodicalWork*» приведено в таблице 3.7.

Таблица 3.7 – Описание атрибутов таблицы «*EducationAndMethodicalWork*».

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *EducationAndMethodicalWork Id* | *Int* | Идентификатор методической работы | Да | Нет |
| *PersonalPlanId* | *Int* | Идентификатор индивидуального плана | Нет | Да |
| *NameOfWork* | *String* | Название работы | Нет | Нет |
| *StartDate* | *Date* | Начало работы | Нет | Нет |
| *EndDate* | *Date* | Окончание работы | Нет | Нет |
| *Note* | *String* | Замечания | Нет | Нет |

Описание атрибутов таблицы «*ScientificReportWork*» приведено в таблице 3.8.

Таблица 3.8 – Описание атрибутов таблицы «*ScientificReportWork*».

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *ScientificReportWorkId* | *Int* | Идентификатор научно-докладной деятельности | Да | Нет |
| *PersonalPlanId* | *Int* | Идентификатор индивидуального плана | Нет | Да |
| *NameofStages* | *String* | Название этапа | Нет | Нет |
| *StartDate* | *Date* | Начало работы | Нет | Нет |
| *EndDate* | *Date* | Окончание работы | Нет | Нет |
| *Note* | *String* | Замечания | Нет | Нет |

Описание атрибутов таблицы «*IdeologicalWork*» приведено в таблице 3.9.

Таблица 3.9 – Описание атрибутов таблицы «*IdeologicalWork*».

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *IdeologicalWorkId* | *Int* | Идентификатор идеологической деятельности | Да | Нет |
| *PersonalPlanId* | *Int* | Идентификатор индивидуального плана | Нет | Да |
| *CopletedWork* | *String* | Проведенная работа | Нет | Нет |
| *Mark* | *Int* | Оценка | Нет | Нет |
| *Note* | *String* | Замечания | Нет | Нет |

Описание атрибутов таблицы «*WorkInManagmentBoard*» приведено в таблице 3.10.

Таблица 3.10 – Описание атрибутов таблицы «*WorkInManagmentBoard*».

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *WorkInManagmentBoardId* | *Int* | Идентификатор по работе советов, комиссий, комитетов управления НДРС, работа в студенческих группах | Да | Нет |
| *PersonalPlanId* | *Int* | Идентификатор индивидуального плана | Нет | Да |
| *CopletedWork* | *String* | Проведенная работа | Нет | Нет |
| *Mark* | *Int* | Оценка, за выполненную работу | Нет | Нет |
| *Note* | *String* | Замечания | Нет | Нет |

Описание атрибутов таблицы «*LoadPlan*» приведено в таблице 3.11.

Таблица 3.11 – Описание атрибутов таблицы «*LoadPlan*».

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *LoadPlanId* | *Int* | Идентификатор нагрузки преподавателя | Да | Нет |
| *PersonalPlanId* | *Int* | Идентификатор индивидуального плана | Нет | Да |

Описание атрибутов таблицы «*MothPortion*» приведено в таблице 3.12.

Таблица 3.12 – Описание атрибутов таблицы «*MothPortion*».

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *MothPortionId* | *Int* | Идентификатор нагрузки преподавателя за заданный месяц | Да | Нет |
| *LoadPlanId* | *Int* | Идентификатор нагрузки преподавателя | Нет | Да |
| *MonthName* | *String* | Название месяца | Нет | Нет |
| *SpecializationId* | *Int* | Идентификатор специальности | Нет | Да |
| *Course* | *String* | Название курса | Нет | Нет |
| *NumberOfGroups* | *Int* | Число групп | Нет | Нет |
| *StrudentNumbers* | *Int* | Число студентов | Нет | Нет |
| *HoursNumbers* | *Int* | Число часов | Нет | Нет |
| *FaultyId* | *Int* | Идентификатор факультета | Нет | Да |

Таким образом было приятно решение об использовании реляционной СУБД *MSSQL*, для которой были спроектированы таблицы, содержащие всю необходимую информацию предметной области.

**3.2 Моделирование архитектуры приложения**

Для разработки данной системы был принят подход по разработки микросервисной архитектуры – подход к проектированию и созданию приложений, основанный на разбиении приложения на множество мелких независимых сервисов, каждый из которых выполняет свою задачу. Эти сервисы могут быть разработаны и развернуты независимо друг от друга, и могут взаимодействовать друг с другом через *API*. Каждый сервис отвечает за определенную бизнес-функцию и имеет свою базу данных, свою схему сообщений и интерфейс *API* для общения с другими сервисами. Такой подход позволяет разделить приложение на небольшие блоки, каждый из которых может быть обслуживаем отдельной командой разработчиков.

В качестве архитектурного стиля взаимодействия компонентов системы выбран *REST,* который определяет набор ограничений и правил для создания распределенных систем, которые могут общаться друг с другом с помощью стандартных протоколов, таких как *HTTP* [10, с. 1]. Для того, чтобы система могла считаться *RESTful*, необходимо соблюдение следующих принципов:

* клиент-серверная архитектура: система должна быть построена по клиент-серверной модели, где клиент и сервер могут развиваться независимо друг от друга;
* без состояния: сервер не должен хранить состояние клиента между запросами, все необходимые данные должны передаваться в запросе, и сервер должен отвечать соответствующим образом на каждый запрос, что уменьшает нагрузку на сервер и упрощает масштабирование;
* кэширование: клиенты могут кэшировать ответы сервера, чтобы снизить количество запросов, в свою очередь сервер может определять, какие ответы должны быть кэшированы и на какой срок;
* единообразный интерфейс: интерфейс должен быть единообразным для всех клиентов, включая структуру ресурсов, формат запросов и ответов, используемые *HTTP* методы и коды состояния;
* слои: система может состоять из нескольких слоев, где каждый слой выполняет свои функции, клиент не должен знать о слоях, которые находятся ниже его уровня, а сервер должен выполнять свои функции независимо от того, какие слои находятся ниже него;
* код по требованию: сервер может отправлять код, который клиент может выполнить, данное решение может быть полезно для расширения функциональности клиента без изменения его кода [10, с. 3].

В целом, использование *REST API* позволяет разработчикам создавать гибкие, масштабируемые и безопасные системы, которые легко могут быть интегрированы с другими системами и устройствами.

В качестве архитектуры отдельно взятого компонента системы, была выбрана компонентно-ориентированная архитектура – стиль архитектуры программного обеспечения, в котором система разбивается на отдельные компоненты, каждый из которых является самодостаточным. Компонент в этой архитектуре представляет собой некоторую функциональность или сервис, который может быть использован в различных контекстах. Каждый компонент выполняет определенную функцию и имеет определенный интерфейс, который определяет, как другие компоненты могут взаимодействовать с ним. Обычно компоненты имеют следующие характеристики:

* самодостаточность: компонент должен быть достаточно независимым, чтобы его можно было использовать в разных контекстах;
* интерфейс: компонент должен иметь четко определенный интерфейс, который определяет, как другие компоненты могут использовать его;
* переиспользуемость: компонент можно использовать множество раз, что означает, что его можно использовать в разных системах и приложениях;
* модульность: компонент должен быть по возможности разбит на небольшие модули, чтобы облегчить его сопровождение;
* зависимость: компонент должен иметь минимальную зависимость от других компонентов;
* независимость: компонент должен быть независимым от платформы или технологии [11].

На рисунке 3.2 приведена схема компонентно-ориентированной архитектуры.

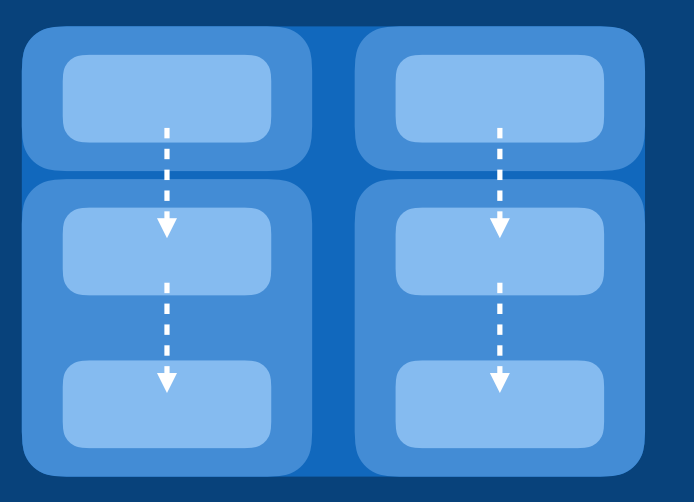


Рисунок 3.2 – Схема компонентно-ориентированной архитектуры

Преимущества компонентно-ориентированной архитектуры:

* повторное использование кода: компоненты могут быть использованы вновь в разных системах и приложениях;
* легкость сопровождения: компоненты являются модульными и независимыми, что упрощает их сопровождение и обновление;
* масштабируемость: систему можно расширять путем добавления новых компонентов не изменения существующих;
* гибкость: компоненты могут быть заменены или обновлены без необходимости изменения других компонентов, что делает систему более гибкой и адаптивной;
* многократное использование тестов: компоненты можно тестировать отдельно, что позволяет повторно использовать тесты для разных систем;
* легкость интеграции: компоненты имеют четкий интерфейс, что упрощает их интеграцию с другими компонентами и системами [11].

Архитектура программного комплекса изображена на рисунке 3.3.

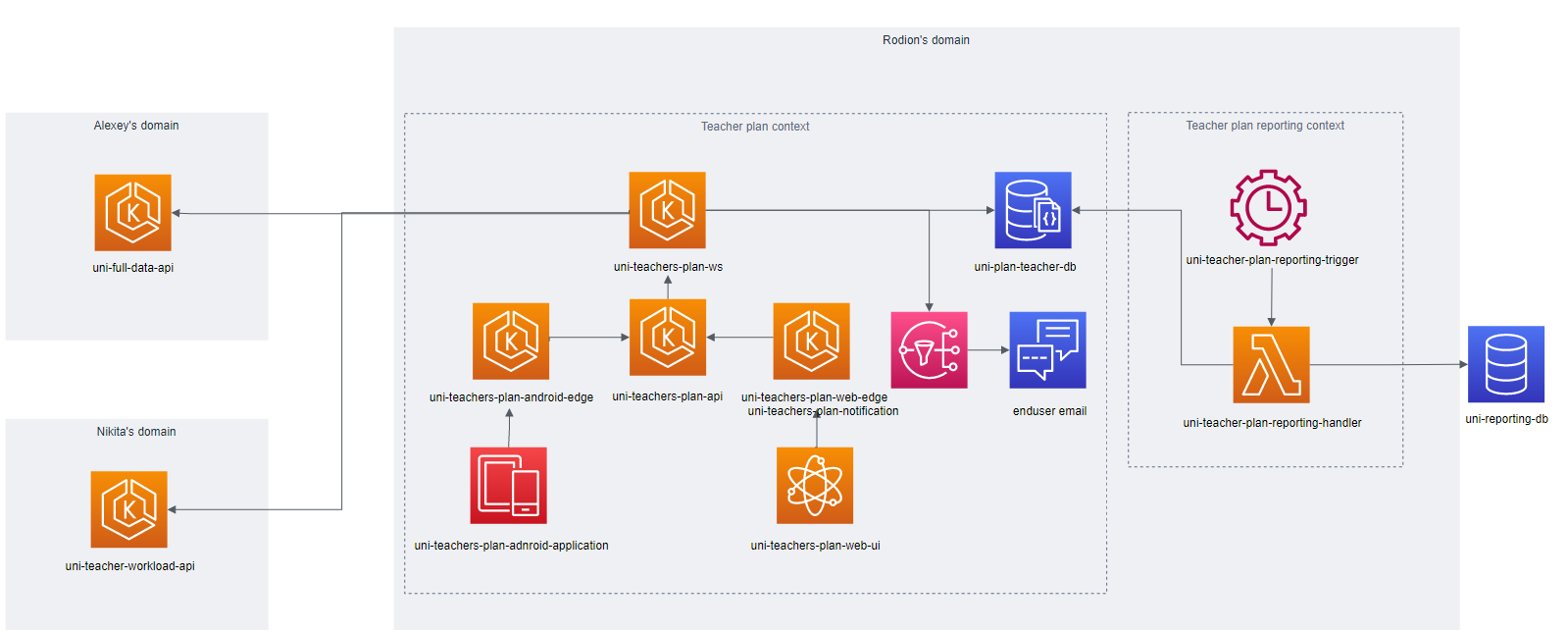
****

Рисунок 3.3 – Общая схема разработанного программного комплекса

По итогам проектирования приложения была разработана микросервисная архитектура, которая соблюдает *REST* принципы. В качестве архитектуры отдельно взятого компонента выбрана компонентно-ориентированная архитектура. Данное архитектурное решение гарантирует простоту расширения системы, повышает отказоустойчивость и облегчает поддержку комплекса.

## **3.3 Структурная схема**

На рисунке 3.4 приведен пример структурной схема приложения.

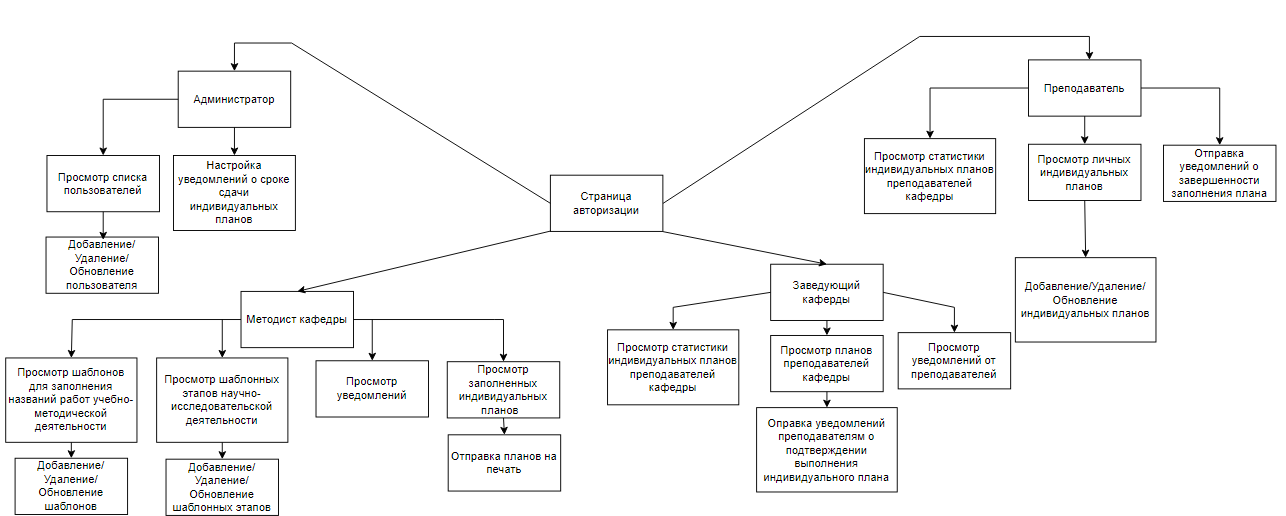


Рисунок 3.4 – Структурная схема приложения

Для создания сложных систем необходимо разрабатывать структуру программы, которая включает в себя описание модулей, их взаимосвязей и используемых данных. Иерархическая схема структуры программы отображает подчиненность модулей, но не учитывает порядок их вызова и функционирование программы.

# 3.4 Функциональная схема

На рисунке 3.5 представлена функциональная схема, составленная для данной системы.

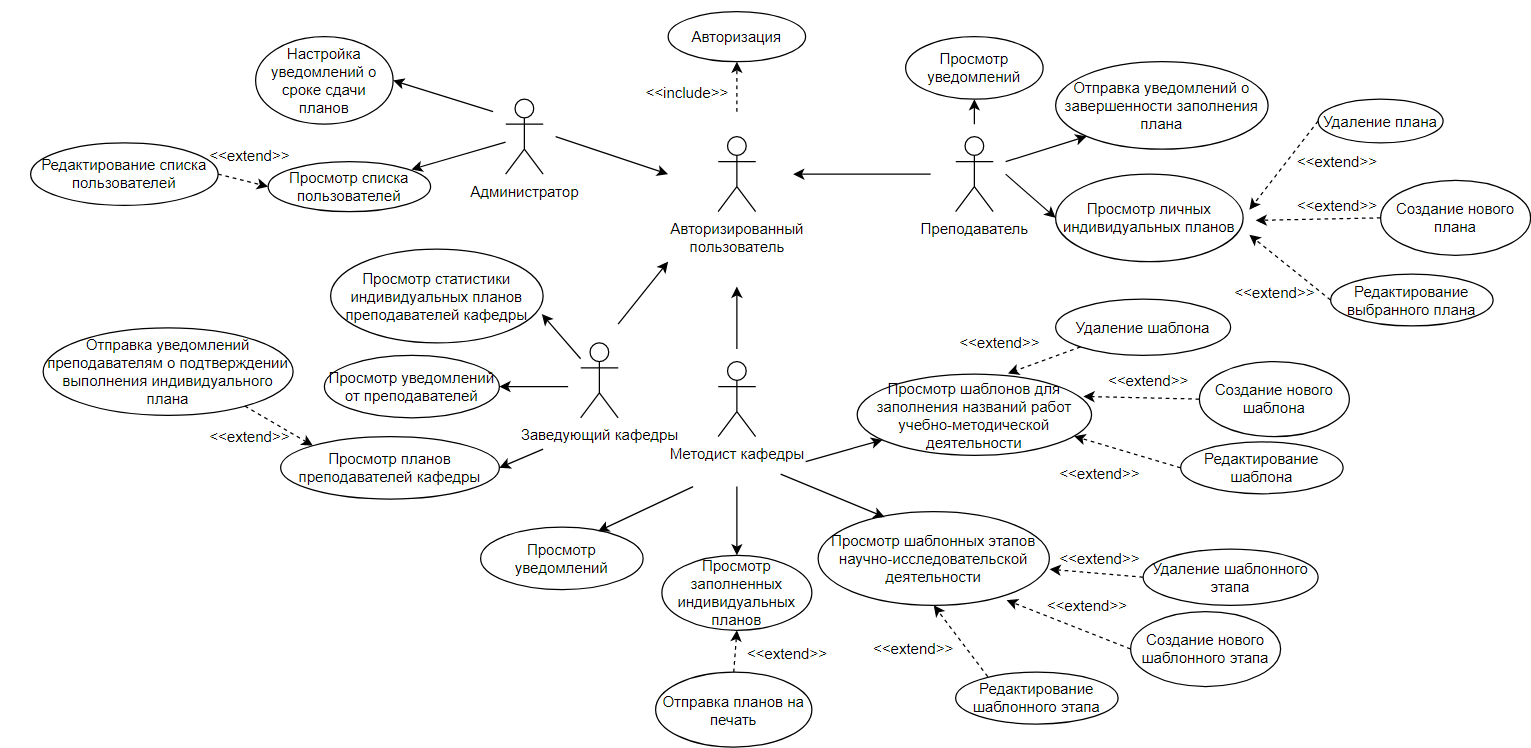


Рисунок 3.5 – Функциональная схема приложения

В системе можно выделить следующих актёров:

* актёр «Администратор» выполняет контроль над списком пользователей и настройками системы;
* актёр «Методист кафедры» выполняет управление шаблонными решениями для индивидуального плана, занимается печатью итоговых планов;
* актёр «Заведующий кафедры» занимается просмотром статистики по планам, отправлением уведомлений преподавателям о подтверждении завершенности плана;
* актёр «Преподаватель» занимается заполнением индивидуального плана.

В системе можно выделить следующие прецеденты:

* прецедент «Авторизация и аутентификация» – прецедент, который является общим для всех актёров и вызывается перед началом работы с приложением;
* прецедент «Просмотр уведомлений» вызывается актёром «Заведующий кафедры», «Преподаватель» и «Методист кафедры» для просмотра сообщений о необходимости заполнения и состояния готовности планов;
* прецедент «Просмотр заполненных индивидуальных планов» вызывается актёром «Методист кафедры» для просмотра полностью заполненных и ободренных планов, расширяется прецедентам «Отправка планов на печать», с целью отправки готовых планов на печать;
* прецедент «Просмотр шаблонных этапов научно-исследовательской деятельности» расширяется прецедентами «Удаление шаблонного этапа», «Создание нового шаблонного этапа» и «Редактирование шаблонного этапа», позволяет просматривать, создавать, удалять и редактировать шаблонные этапы научно-исследовательской деятельности;
* прецедент «Отправка уведомлений о завершенности заполнения плана» вызывается для уведомления актера «Заведующий кафедры», о готовности индивидуального плана;
* прецедент «Просмотр личных индивидуальных планов» вызывается актёром «Преподаватель» расширяется прецедентами «Удаление плана», «Создание нового плана» и «Редактирование выбранного плана», позволяет ему управлять личными индивидуальными планами.

Наличие прецедентов дает ясное определение требований к системе, эффективно отображает коммуникацию компонентов системы, что ускоряет понимания предметной области, для новых разработчиков.

# ЗАКЛЮЧЕНИЕ

Целью данного задания, выданного на предприятии, являлось создание проекта, задача которого – автоматизация ведения индивидуального плана преподавателей.

В результате выполнения работы был создан программный продукт, полностью соответствующий описанным требованиям. Средствами реализации данного программного продукта являются: объектно-ориентированный язык *C#*, *MSSQL Server*, *ASP.NET Core*, *ASP.NET Core Identity*, *Entity Framework Core*, *Docker*, *Java, React*. Также следует отметить то, что данное приложение использует микросервисную архитектуру, где в качестве слоя представления выступает веб приложение, а также андроид приложение.

Преимуществом данного проекта является то, что значительно упрощается задача введения учебных планов для преподавателей, облегчается и становится более комфортной в исполнении, появляется возможность заполнения планов удаленно, что было бы проблематичным без применения средств автоматизации. Также удобным является то, что генерация некоторых планов может происходить автоматически, в удобном для пользователя формате.

Следует отметить ещё и тот факт, что реализованная архитектура приложения позволяет легко и удобно масштабировать данное приложение, что в будущем может сыграть достаточно важную роль.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Epam Systems*: Электронный ресурс. – Электрон. данные. – Режим доступа: <https://www.epam.com/>. – Дата доступа: 27.04.2023.
2. Информационно-программный комплекс «*UNIVER*»: Свободная энциклопедия. – Электрон. данные. – Режим доступа: <https://it.kaznu.kz/?page_id=847>. – Дата доступа: 27.04.2023.
3. Система автоматизации делопроизводства и документооборота «Дело»: Электронная документация. – Электрон. данные. – Режим доступа: <http://www.e-office.by/produkty/elektronnoe-delo>. – Дата доступа: 27.04.2023.
4. АСМО-Документооборот: Электронная документация. – Электрон. данные. – Режим доступа: <https://informatika37.ru/resheniya-asmo/asmo-dokumentooborot/>. – Дата доступа: 27.04.2023.
5. *Mark J. Price*, *C# 10 and .NET 6 – Modern Cross-Platform Development* / *Mark J. Price*; под общ. ред. *Suman Sen*. – М. : Диалог-Мифи, 2021. – 314 с.
6. Гриффитс Дэвид, Программирование для *Android* / Гриффитс Дэвид; под общ. ред. Гриффитс Дон. – М. : СПб, 2018. – 95 с.
7. [*Robin Dewson*](https://link.springer.com/book/10.1007/978-1-4302-3751-8#author-0-0), [*Beginning SQL Server 2012 for Developers*](https://link.springer.com/book/10.1007/978-1-4302-3751-8) / [*Robin Dewson*](https://link.springer.com/book/10.1007/978-1-4302-3751-8#author-0-0); под общ. ред. *Sons*. – М. : Wiley, 2012. – 56 с.
8. Что такое реляционная база данных. – Электронная документация. – Режим доступа: <https://aws.amazon.com/ru/relational-database>. – Дата доступа: 27.04.2023.
9. *MongoDb*: Электронная документация. – Электрон. данные. – Режим доступа: https://www.mongodb.com/docs/. – Дата доступа: 27.04.2023.
10. *Representational State Transfer*: материалы междунар. науч. конф., посвящ. Способы передачи данных по сети интернет, Калифорния, 2000 г. / США. гос. ун-т ; редкол.:  *[Roy Fielding](http://roy.gbiv.com/)* (отв. ред.) [и др.]. – Калифорния, 2000. – 1 с.
11. *Modular Monoliths*: Свободная энциклопедия. – Электрон. данные. – Режим доступа: <https://files.gotocon.com/uploads/slides/conference_12/515/original/gotoberlin2018-modular-monoliths.pdf>. – Дата доступа: 27.04.2023.