МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

Отчет по технологической практике

на тему: “Веб-приложение для управления рабочим временем”

Исполнитель: студент гр. ИП-31

Казутин Павел Николаевич

Руководитель от предприятия:

Гуд Сергей Николаевич

Руководитель: преподаватель

Богданова Наталья Сергеевна

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Подписи членов комиссии \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2021

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc75172395)

[1 ОБЩИЕ СВЕДЕНИЯ О ПРЕДПРИЯТИИ 4](#_Toc75172396)

[1.1 История организации 4](#_Toc75172397)

[1.1 Охрана труда и техника безопасности на рабочем месте 6](#_Toc75172398)

[1.3 Технологии, используемые на предприятии, программное обеспечение на предприятии 10](#_Toc75172399)

[2 ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ 11](#_Toc75172400)

[2.1 Постановка задания 11](#_Toc75172401)

[2.2 Обзор используемых технологий 12](#_Toc75172402)

[2.3 Структура приложения 17](#_Toc75172403)

[2.4 Описание интерфейса пользователя 22](#_Toc75172404)

[ЗАКЛЮЧЕНИЕ 23](#_Toc75172405)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 24](#_Toc75172406)

[Приложение А 25](#_Toc75172407)

# ВВЕДЕНИЕ

Ускоряющийся ритм жизни современного общества требует от всех чёткого и продуманного плана на следующий день, месяц, квартал и т. д. Для планирования своего времени существует множество решений разной степени удобства и, как следствие, полезности.

Центральным звеном такой системы должен быть календарь, на котором пользователь может видеть в удобном для него виде. Так же, удобно видеть в собственном графике рабочие проекты и задачи, на которые этот проект разбивается.

Так же, реалии требуют перехода многих прикладных приложений из «нативного» вида в вид веб-приложений.

Цели технологической практики: закрепление, расширение, углубление и систематизация теоретических знаний, а также приобретение навыков проектирования и конструирования информационных систем.

Задачи технологической практики:

* развитие и закрепление практических навыков выполнения анализа предметной области;
* приобретение практического опыта проектирования программных систем;
* развитие и закрепление практических навыков использования языков и инструментальных средств моделирования при проектировании системы;

‑‑ развитие и закрепление практических навыков создания программных систем с использованием современных сред разработки, поддерживающих возможность командной работы, контроля проекта и версий системы.

При прохождении технологической практики в организации необходимо выполнить индивидуальное задание и выполнить поставленные задачи, связанные с созданием автоматизированной системы.

# 1 ОБЩИЕ СВЕДЕНИЯ О ПРЕДПРИЯТИИ

## **1.1 История организации**

EPAM Systems – американская ИТ-компания, основанная в 1993 году. Крупнейший мировой производитель заказного программного обеспечения, специалист по консалтингу, резидент Белорусского парка высоких технологий.

Компания EPAM была основана в 1993 годудвумя одноклассниками Аркадием Добкиными Леонидом Лознером. Название компании происходило от «Effective Programming for America». Первые офисы были открыты в США и Беларуси. Позже были открыты центральный североамериканский офис в Лоренсвилле, США, штат Нью-Джерси и центральный европейский офис в Будапеште, Венгрия, а также офисы по обслуживанию клиентов в Австрии, Австралии, Армении, Болгарии, Белоруссии, Великобритании, Германии, Индии, Ирландии, Казахстане, Канаде, Китае, Мексике, Нидерландах, ОАЭ, Польше, России, Сингапуре, Украине, Чехии, Швеции, Швейцарии.

ИООО «ЭПАМ Системз» выполняет следующие виды работ:

– 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;

– обучение специалистов на базе университетских лабораторий; онлайн.

Перечислим крупные приобретения и поглощения компании.

25 января 2012 года объявила о начале подготовки к IPO(Initial Public Offering – первичное публичное размещение акций) на Нью-Йоркской фондовой бирже. Как говорилось в сообщении компании, крупнейший пакет акций EPAM Systems могут продать фонды, аффилированные с инвесткомпанией Siguler Guff & Co: после IPO они уменьшат долю в EPAM Systems с 52,5 % до 41,2 %. Основатель и генеральный директор Аркадий Добкин рассчитывал продать 2 % акций компании. Всего акционеры EPAM Systems предполагали разместить до 14 % акций компании. Ещё 3,6 % акций допэмиссии, как планировалось, продаст сама компания, и объём размещения составит 17,7 % на $133,2 млн.

IPO состоялось 8 февраля и было оценено аналитиками как неудачное.. В ходе размещения было продано 6 млн акций (14,7 % увеличенного капитала) за $72 млн, или $12 за бумагу (при этом ранее EPAM объявляла ценовой коридор в $16-18 за бумагу). 33 % проданных акций — дополнительная эмиссия. В соответствии с оценкой на IPO стоимость всей компании составила $488 млн. По мнению экспертов, проблема оказалась не в бизнесе EPAM, а в непростой ситуации на рынках, особенно в Европе.

Всего через 2 года, в июне 2014, капитализация компании выросла более чем в четыре раза и составила $2,14 млрд. Капитализация EPAM в мае 2019 составила $9.39 млрд.

В марте 2004 года EPAM приобрела компанию Fathom Technology в Венгрии, а в сентябре 2006 VDI в России, образовав единую компанию под именем EPAM Systems со штатом сотрудников в 2200 человек.

В 2012 году компания совершает ряд приобретений на северо-американском рынке, в числе которых канадская компания Thoughtcorp и крупный поставщик услуг по разработке цифровых стратегий и организации многоканального взаимодействия Empathy Lab.

В 2014 году EPAM приобрела китайскую ИТ-компанию Jointech (Joint Technology Development Limited), за счёт чего, как следует из пресс-релиза EPAM, расширила свои возможности в Азиатско-Тихоокеанском регионе и купила американского поставщика услуг для здравоохранения и медико-биологического сектора GGA Software Services.

В 2015 EPAM Systems поглотил американские компании: [NavigationArts](https://en.wikipedia.org/wiki/NavigationArts), специализирующуюся на цифровом консалтинге и дизайне, а также Alliance Global Services, которая специализируется на выпуске ПО и решений для автоматизированного тестирования. В связи с этим приобретением руководство EPAM Systems пересмотрело прогноз по выручке в сторону увеличения, ожидая её на уровне не ниже 905 млн долларов в 2015 году против 730 млн годом ранее.

В 2016 году EPAM поглотила китайскую компанию Dextrys, со штатом в 400 сотрудников.

В 2018 году EPAM поглотила американскую компанию Continuum, со штатом более 150 человек, а также цифровое агентство TH\_NK, входящее в топ-100 цифровых агентств Великобритании.

В 2018 году EPAM запустил InfoNgen 7.0, аналитическую платформу с элементами машинного обучения для исследования информации и проведения конкурентного анализа данных из более чем 200 000 многоязычных веб-источников, а также Telescope AI, масштабируемую модульную платформу на базе искусственного интеллекта, которая помогает бизнесу получить всестороннее представление о внутренних процессах организации.

Компания запустила EPAM SolutionsHub – библиотеку программных продуктов, акселераторов и OpenSource-решений, а также Open Source Contributor Index – инструмент, который оценивает вклад коммерческих компаний в развитие решений с открытым исходным кодом.

В данный момент EPAM насчитывает 42 тысячи работников, из которых 10 – в Республике Беларусь.

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

1.2 Охрана труда

Охрана труда – система законодательных актов, социальноэкономических, организационных, технических, гигиенических и лечебно-профилактических мероприятий и средств, обеспечивающих безопасность, сохранение здоровья и работоспособности человека в процессе труда. Научно-технический прогресс внес серьезные изменения в условия производственной 5 деятельности работников умственного труда. Их труд стал более интенсивным, напряженным, требующим значительных затрат умственной, эмоциональной и физической энергии. Это потребовало комплексного решения проблем эргономики, гигиены и организации труда, регламентации режимов труда и отдыха. Рабочее место – это часть пространства, в котором инженер осуществляет трудовую деятельность, и проводит большую часть рабочего времени.

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.

Общие требования охраны труда:

– к работе программистом допускаются лица не моложе 18 лет, имеющие соответствующую выполняемой работе квалификацию, прошедшие вводный и первичный на рабочем месте инструктажи по охране труда, обученные безопасности труда при работе с персональным компьютером;

– для выполнения работ на персональном компьютере программист должен изучить Инструкцию по эксплуатации персонального компьютера, на котором работник выполняет работы, пройти инструктаж по электробезопасности и получить I группу;

– программист, выполняющий работу на персональном компьютере, независимо от квалификации и стажа работы, не реже одного раза в шесть месяцев должен проходить повторный инструктаж по безопасности труда; в случае нарушения требований безопасности труда, при перерыве в работе более чем на 60 календарных дней программист должен пройти внеплановый инструктаж;

– программист, не прошедший инструктажи по охране труда и не имеющий I группы по электробезопасности, к самостоятельной работе не допускается;

– программист, показавший неудовлетворительные навыки и знания требований безопасности при работе на персональном компьютере, к самостоятельной работе не допускается;

– программист, допущенный к постоянной работе на персональном компьютере, перед поступлением на работу и в дальнейшем периодически (не реже 1 раза в год) должен проходить медицинские осмотры;

– программист, допущенный к самостоятельной работе, должен знать: правила эксплуатации и требования безопасности при работе с персональным компьютером, способы рациональной организации рабочего места, санитарно-гигиенические требования к условиям труда, опасные и вредные производственные факторы, которые могут оказывать неблагоприятное воздействие на программиста;

– программист, направленный для участия в несвойственных его профессии работах, должен пройти целевой инструктаж по безопасному выполнению предстоящих работ;

– программисту запрещается пользоваться инструментом, приспособлениями и оборудованием, безопасному обращению с которыми он не обучен. Требование охраны труда перед началом работы;

– перед началом работы программисту следует рационально организовать свое рабочее место;

– программист должен знать о том, что, если в помещении расположены несколько персональных компьютеров, то для обеспечения безопасности расстояние между ними должно быть не менее 1,5 м;

– не рекомендуется располагать монитор экраном к окну;

– для того, чтобы в процессе работы не возникало перенапряжение зрительного анализатора, программисту следует проверить, чтобы на клавиатуре и экране монитора не было бликов света;

– для повышения контрастности изображения перед началом работы программист должен очистить экран монитора от пыли, которая интенсивно оседает на нем под воздействием зарядов статического электричества;

– программист должен убрать с рабочего места все лишние предметы, не используемые в работе;

– перед включением персонального компьютера программисту следует визуально проверить исправность электропроводки, вилки, розетки, а также электрических подсоединений между собой всех устройств, входящих в комплект персонального компьютера;

– перед началом выполнения работы программист должен проверить исправность персонального компьютера и подготовить его к работе.

Требования охраны труда в аварийных ситуаиях:

– при обнаружении каких-либо неполадок в работе персонального компьютера программист должен прекратить работу, выключить компьютер и сообщить об этом непосредственному руководителю для организации ремонта;

– программисту не следует самому устранять технические неполадки персонального компьютера;

– программист не должен производить работу при снятом корпусе компьютера;

– при несчастном случае, отравлении, внезапном заболевании необходимо немедленно оказать первую помощь пострадавшему, вызвать врача или помочь доставить пострадавшего к врачу, а затем сообщить руководителю о случившемся;

– программист должен уметь оказывать первую помощь при ранениях; при этом он должен знать, что всякая рана легко может загрязниться микробами, находящимися на ранящем предмете, коже пострадавшего, а также в пыли, на руках оказывающего помощь и на грязном перевязочном материале.

Оказывая первую помощь при ранении, необходимо соблюдать следующие правила:

– нельзя промывать рану водой или даже каким-либо лекарственным препаратом, засыпать порошком и смазывать мазями, так как это препятствует заживлению раны, вызывает нагноение и способствует занесению в нее грязи с поверхности кожи;

– нужно осторожно снять грязь с кожи вокруг раны, очищая ее от краев наружу, чтобы не загрязнять рану; очищенный участок кожи нужно смазать йодом и наложить повязку;

– для оказания первой помощи при ранении необходимо вскрыть имеющийся в аптечке перевязочный пакет;

– при наложении перевязочного материала не следует касаться руками той его части, которая должна быть наложена непосредственно на рану; если перевязочного пакета почему-либо не оказалось, то для перевязки можно использовать чистый платок, чистую ткань и т.п.; накладывать вату непосредственно на рану нельзя;

– на то место ткани, которое накладывается непосредственно на рану, нужно накапать несколько капель йода, чтобы получить пятно размером больше раны, а затем положить ткань на рану; оказывающий помощь должен вымыть руки или смазать пальцы йодом; прикасаться к самой ране даже вымытыми руками не допускается;

– первая помощь пострадавшему должна быть оказана немедленно и непосредственно на месте происшествия, сразу же после устранения причины, вызвавшей травму, используя медикаменты и перевязочные материалы, которые должны храниться в аптечке;

– аптечка должна быть укомплектована перевязочными материалами и медикаментами, у которых не истек срок реализации; аптечка должна находиться на видном и доступном месте;

– если пострадавший находится в бессознательном состоянии, но с сохранившимся устойчивым дыханием и пульсом, его следует удобно уложить, расстегнуть одежду, создать приток свежего воздуха, дать понюхать нашатырный спирт, обрызгать водой и обеспечить полный покой;

– если пострадавший плохо дышит (очень редко и судорожно), ему следует делать искусственное дыхание и массаж сердца; при отсутствии у пострадавшего признаков жизни (дыхания и пульса) нельзя считать его мертвым, искусственное дыхание следует производить непрерывно как до, так 8 и после прибытия врача; вопрос о бесцельности дальнейшего проведения искусственного дыхания решает врач;

– при обнаружении пожара или признаков горения (задымление, запах гари, повышение температуры и т.п.) необходимо немедленно уведомить об этом пожарную охрану по телефону 101;

– до прибытия пожарной охраны нужно принять меры по эвакуации людей, имущества и приступить к тушению пожара;

– при возгорании персонального компьютера программист должен отключить его от источника тока и приступить к тушению своими силами; при этом следует помнить, что для тушения установок, находящихся под напряжением, применяют углекислотные или порошковые огнетушители. Большое значение имеет также характер работы.

В частности, при организации рабочего места программиста должны быть соблюдены следующие основные условия:

– Оптимальное размещение оборудования, входящего в состав рабочего места;

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

– Необходимо естественное и искусственное освещение для выполнения поставленных задач;

– Уровень акустического шума не должен превышать допустимого значения. Главными элементами рабочего места программиста являются письменный стол и кресло. Основным рабочим положением является положение сидя.

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

Компания предоставляет сотрудникам современные технологии и оборудования для комфортной работы. Программное обеспечение представлено широким спектром программ, необходимых для предоставления услуг.

Для общения на проектах сотрудники часто пользуются современными средствами видеосвязи, такими как Skype, однако компания предпочитает к использованию Microsoft Teams*.*

Microsoft Teams – корпоративная платформа, объединяющая в рабочем пространстве чат, встречи, заметки и вложения. Чаты – коллективы могут объединяться в групповых или личных переписках, обсуждая те или иные вопросы, пересылать различные файлы, обмениваться стикерами и т. д. В целом это очень напоминает функцию мессенджера.

В компании используются следующие технологии:

* Microsoft Technologies Division: asp.net mvc, .net, xamarin, wcf, wpf, nhibernate, javascript, azure;
* ­Cloud and DevOps: octopus, terraform, python, teamcity, Zabbix, Salt, Kafka, Docker, Chef, Kubernetes, Google cloud, Tfs, AWS, Azure, Cassandra, Bamboo, Vmware.
* Enterprise Applications: Java, SAP Cloud Platform, SAPUI5/ Fiori, ABAP, CDS, OData, Apex, Lightning, Visualforce, .NET, MS SQL Server;

# ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

## **Постановка задания**

Разрабатываемое веб-приложение должно иметь следующие функции:

* просмотр своих планов на день;
* просмотр заданий проектов, в которых пользователь состоит;
* просмотр и возможность отправки сообщений другим пользователям;
* создание проектов;
* добавление заданий проекта;
* добавление пользователей в проект;
* регистрация пользователей, в т.ч. через соц. сети;

В ходе анализа предметной области было решено использовать единственную роль «пользователь», что позволит создать более удобное приложение для каждого пользователя.

Для реализации указанного функционала был выбран объектно-ориентированный язык программирования C#, предоставляющий широкие возможности программисту. Технология для создания пользовательского интерфейса – ASP.Net Core MVC -- технология от компании Microsoft, предназначенная для создания различного рода веб-приложений: от небольших веб-сайтов до крупных веб-порталов и веб-сервисов. Используемая СУБД – PostgreSQL.

## **Обзор используемых технологий**

Для создания веб-приложения были использованы следующие технологии:

* язык программирования C# версии 9.0;
* платформа .Net Core 5;
* платформа ASP.Net Core 5 MVC;
* СУБД PostgreSQL 12.

Опишем каждую из приведённых выше технологий.

C# (произносится как "си шарп") — современный [2] объектно-ориентированный и типобезопасный язык программирования. C# позволяет разработчикам создавать множество типов безопасных и надежных приложений, работающих в экосистеме .NET. C# относится к широко известному семейству языков C, и покажется хорошо знакомым любому, кто работал с C, C++, Java или JavaScript.

C# — это объектно- и компонентно-ориентированный язык программирования. C# предоставляет языковые конструкции для непосредственной поддержки такой концепции работы. Благодаря этому C# подходит для создания и применения программных компонентов. С момента создания язык C# обогатился функциями для поддержки новых рабочих нагрузок и современными рекомендациями по разработке ПО.

Вот лишь несколько функций языка C#, которые позволяют создавать надежные и устойчивые приложения.

Сборка мусора автоматически освобождает память, занятую недоступными неиспользуемыми объектами. Типы, допускающие значение null, обеспечивают защиту от переменных, которые не ссылаются на выделенные объекты. Обработка исключений предоставляет структурированный и расширяемый подход к обнаружению ошибок и восстановлению после них. Лямбда-выражения поддерживают приемы функционального программирования. Синтаксис LINQ создает общий шаблон для работы с данными из любого источника. Поддержка языков для асинхронных операций предоставляет синтаксис для создания распределенных систем.

В C# действует единая система типов. Все типы C#, включая типы-примитивы, такие как int и double, наследуют от одного корневого типа object. Все типы используют общий набор операций, а значения любого типа можно хранить, передавать и обрабатывать схожим образом. Более того, C# поддерживает как определяемые пользователями ссылочные типы, так и типы значений. C# позволяет динамически выделять объекты и хранить упрощенные структуры в стеке. C# поддерживает универсальные методы и типы, обеспечивающие повышенную безопасность типов и производительность. C# предоставляет итераторы, которые позволяют разработчикам классов коллекций определять пользовательские варианты поведения для клиентского кода.

В языке особое внимание уделяется управлению версиями для обеспечения совместимости программ и библиотек при их изменении. Вопросы управления версиями существенно повлияли на такие аспекты разработки C#, как раздельные модификаторы virtual и override, правила разрешения перегрузки методов и поддержка явного объявления членов интерфейса.

Программы C# выполняются в .NET [3], виртуальной системе выполнения, вызывающей общеязыковую среду выполнения (Common Language Runtime, CLR) и набор библиотек классов. Среда CLR — это реализация общеязыковой инфраструктуры языка (Common Language Infrastructure, CLI), являющейся международным стандартом, от корпорации Майкрософт. CLI является основой для создания сред выполнения и разработки, в которых языки и библиотеки прозрачно работают друг с другом.

Исходный код, написанный на языке C#, компилируется в промежуточный язык (Intermediate Language, IL), который соответствует спецификациям CLI. Код на языке IL и ресурсы, в том числе растровые изображения и строки, сохраняются в сборке, обычно с расширением .dll. Сборка содержит манифест с информацией о типах, версии, языке и региональных параметрах для этой сборки.

При выполнении программы C# сборка загружается в среду CLR. Среда CLR выполняет JIT (Just In Time)-компиляцию из кода на языке IL в инструкции машинного языка. Среда CLR также выполняет другие операции, например, автоматическую сборку мусора, обработку исключений и управление ресурсами. Код, выполняемый средой CLR, иногда называют "управляемым кодом", чтобы подчеркнуть отличия этого подхода от "неуправляемого кода", который сразу компилируется в машинный язык для определенной платформы.

Обеспечение взаимодействия между языками является ключевой особенностью .NET. Код IL, созданный компилятором C#, соответствует спецификации общих типов (Common Type System, CTS). Код IL, созданный из кода на C#, может взаимодействовать с кодом, созданным из версий .NET для языков F#, Visual Basic, C++ и любых других из более чем 20 языков, совместимых с CTS. Одна сборка может содержать несколько модулей, написанных на разных языках .NET, и все типы могут ссылаться друг на друга, как если бы они были написаны на одном языке.

В дополнение к службам времени выполнения .NET также включает расширенные библиотеки. Эти библиотеки поддерживают множество различных рабочих нагрузок. Они упорядочены по пространствам имен, которые предоставляют разные полезные возможности: от операций файлового ввода и вывода до управления строками и синтаксического анализа XML (eXtensible Markup Language), от платформ веб-приложений до элементов управления Windows Forms.

ASP.NET Core является кроссплатформенной [4], высокопроизводительной средой с открытым исходным кодом для создания современных облачных приложений, подключенных к Интернету. ASP.NET Core позволяет выполнять следующие задачи:

* создавать веб-приложения и службы, приложения Интернета вещей (IoT) и серверные части для мобильных приложений;
* использовать избранные средства разработки в Windows, macOS и Linux;
* выполнять развертывания в облаке или локальной среде;
* запускать в .NET Core.

Миллионы разработчиков использовали и продолжают использовать ASP.NET 4.x для создания веб-приложений. ASP.NET Core — это модификация ASP.NET 4.x с архитектурными изменениями, формирующими более рациональную и более модульную платформу.

ASP.NET Core предоставляет следующие преимущества:

* единое решение для создания пользовательского веб-интерфейса и веб-API;
* разработано для тестируемости;
* Razor Pages упрощает написание кода для сценариев страниц и повышает его эффективность;
* Blazor позволяет использовать в браузере язык C# вместе с JavaScript. совместное использование серверной и клиентской логик приложений, написанных с помощью .NET;
* возможность разработки и запуска в ОС Windows, macOS и Linux;
* открытый исходный код и ориентация на сообщество;
* интеграция современных клиентских платформ и рабочих процессов разработки;
* поддержка размещения служб удаленного вызова процедур (RPC) с помощью gRPC;
* облачная система конфигурации на основе среды;
* встроенное введение зависимостей;
* упрощенный высокопроизводительный модульный конвейер HTTP-запросов;
* инструментарий, упрощающий процесс современной веб-разработки.

ASP.NET Core MVC (Model View Controller) представляет собой упрощенную, эффективно тестируемую платформу с открытым исходным кодом, оптимизированную для использования с ASP.NET Core.

ASP.NET Core MVC предоставляет основанный на шаблонах способ создания динамических веб-сайтов с четким разделением задач. Она обеспечивает полный контроль разметки, поддерживает согласованную с TDD (Test Driven Development) разработку и использует новейшие веб-стандарты.

Структура архитектуры MVC разделяет приложение на три основных группы компонентов: модели, представлении и контроллеры. Это позволяет реализовать принципы разделения задач. Согласно этой структуре, запросы пользователей направляются в контроллер, который отвечает за работу с моделью для выполнения действий пользователей и получение результатов запросов. Контроллер выбирает представление для отображения пользователю со всеми необходимыми данными модели.

Такое распределение обязанностей позволяет масштабировать приложение в контексте сложности, так как проще писать код, выполнять отладку и тестирование компонента (модели, представления или контроллера) с одним заданием. Гораздо труднее обновлять, тестировать и отлаживать код, зависимости которого находятся в двух или трех этих областях. Например, логика пользовательского интерфейса, как правило, подвергается изменениям чаще, чем бизнес-логика. Если код представления и бизнес-логика объединены в один объект, содержащий бизнес-логику, объект необходимо изменять при каждом обновлении пользовательского интерфейса. Это часто приводит к возникновению ошибок и необходимости повторно тестировать бизнес-логику после каждого незначительного изменения пользовательского интерфейса.

Модель в приложении MVC представляет состояние приложения и бизнес-логику или операций, которые должны в нем выполняться. Бизнес-логика должна быть включена в состав модели вместе с логикой реализации для сохранения состояния приложения. Как правило, строго типизированные представления используют типы ViewModel, предназначенные для хранения данных, отображаемых в этом представлении. Контроллер создает и заполняет эти экземпляры ViewModel из модели.

Представления отвечают за представление содержимого через пользовательский интерфейс. Они используют Razor обработчик представлений для внедрения кода .NET в разметку HTML. Представления должны иметь минимальную логику, которая должна быть связана с представлением содержимого. Если есть необходимость выполнять большую часть логики в представлении для отображения данных из сложной модели, рекомендуется воспользоваться компонентом представления, ViewModel или шаблоном представления, позволяющими упростить представление.

Контроллеры — это компоненты для управления взаимодействием с пользователем, работы с моделью и выбора представления для отображения. В приложении MVC представление служит только для отображения информации. Обработку введенных данных, формирование ответа и взаимодействие с пользователем обеспечивает контроллер. В структуре MVC контроллер является начальной отправной точкой и отвечает за выбор рабочих типов моделей и отображаемых представлений (именно этим объясняется его название — он контролирует, каким образом приложение отвечает на конкретный запрос).

PostgreSQL — свободная объектно-реляционная система управления базами данных (СУБД).

PostgreSQL создана на основе некоммерческой СУБД Postgres, разработанной как open-source проект в Калифорнийском университете в Беркли. К разработке Postgres, начавшейся в 1986 году, имел непосредственное отношение Майкл Стоунбрейкер, руководитель более раннего проекта Ingres, на тот момент уже приобретённого компанией Computer Associates. Название расшифровывалось как «Post Ingres», и при создании Postgres были применены многие ранние наработки.

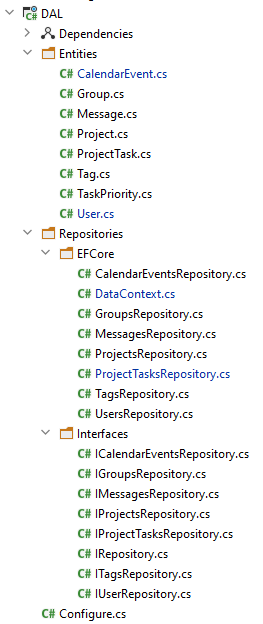
Стоунбрейкер и его студенты разрабатывали новую СУБД в течение восьми лет с 1986 по 1994 год. За этот период в синтаксис были введены процедуры, правила, пользовательские типы и другие компоненты. В 1995 году разработка снова разделилась: Стоунбрейкер использовал полученный опыт в создании коммерческой СУБД Illustra, продвигаемой его собственной одноимённой компанией (приобретённой впоследствии компанией Informix), а его студенты разработали новую версию Postgres — Postgres95, в которой язык запросов POSTQUEL — наследие Ingres — был заменен на SQL.

Разработка Postgres95 была выведена за пределы университета и передана команде энтузиастов. Новая СУБД получила имя, под которым она известна и развивается в текущий момент — PostgreSQL.

## **Структура приложения**

В ходе выполнения задач проектирования было разработано трёхслойное веб-приложение, состоящее из слоёв, описанных ниже.

Первый слой приложения – DAL (Data Access Layer – слой доступа к данным). На этом «этаже» содержатся классы, описывающие предметную область, класс контекста данных, интерфейсы и реализации репозиториев для сущностей предметной области. Структура слоя DAL показана на рисунке 2.1.

Следующий слой – BLL (Business Logic Layer – слой бизнес-логики). На этом слое находятся алгоритмы бизнес-логики – сервисы, DTO (Data Transfer Object – объект обмена данными). Структура BLL показана на рисунке 2.2

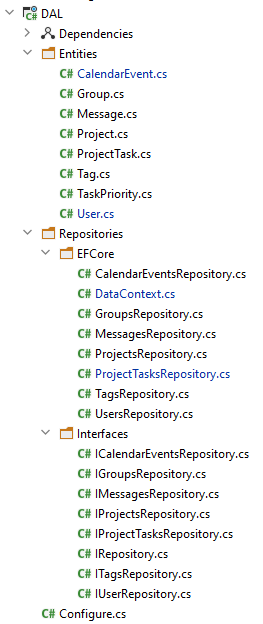


Рисунок 2.2 – Структура DAL

Самый верхний слой – слой пользовательского интерфейса. На нём находятся контроллеры, представления и модели данных. Так же, на этом слое производится авторизация и аутентификация пользователей, реализованная с использование технологии Microsoft Identity. Структура слоя пользовательского интерфейса показана на рисунке 2.3.

Таким образом получается трёхслойное приложение, в котором каждый слой знает только о слое на уровень ниже. Такой подход позволяет создавать легко изменяемые, гибкие системы. Также, для снижения связности приложения используется принцип IoC (Inversion of Control -- инверсия управления), реализуемый через DI (Dependency Injection – внедрение зависимостей). Для DI используется библиотека, разработанная компанией Microsoft.

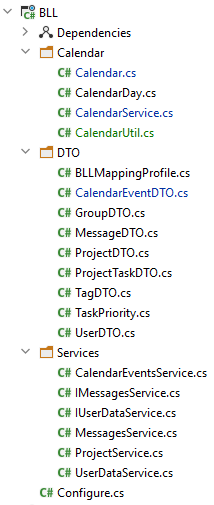
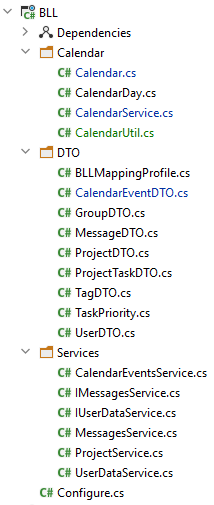


Рисунок 2.3 – Структура BLL

Опишем классы предметной области.

В ходе анализа предметной области, были выделены следующие классы:

* CalendarEvent;
* Group;
* Message;
* Project;
* ProjectTask;
* Tag;
* User.

Опишем каждый.

Класс CalendarEvent представляет собой пользовательское событие в каленаре. Поля класса описаны в таблице 2.1.

Класс Group представляет собой группу пользователей, которых владелец группы может добавить в свой проект. Поля класса описаны в таблице 2.2.

Класс Message представляет собой текстовое сообщение от одного пользователя другому. Поля класса описаны в таблице 2.3.

Класс Project представляет собой некоторый проект. Поля класса описаны в таблице 2.4.

Класс ProjectTask представляет собой задачу в проекте. Поля класса описаны в таблице 2.5.

Класс Tag представляет собой некоторый тэг задачи. Поля класса описаны в таблице 2.6.

Класс User представляет собой личную информацию о пользователе. Поля класса описаны в таблице 2.7.

Таблица 2.1 – Описание класса CalendarEvent

|  |  |  |
| --- | --- | --- |
| **Наименование поля** | **Тип хранимых данных** | **Описание** |
| Id | string | Идентификатор записи |
| OwnerId | string | Идентификатор владельца |
| EventDate | DateTime | Дата и время события |
| Duration | TimeSpan | Продолжительность |
| Comment | string | Пояснение к событию |
| Tags | ICollection <Tag> | Метки |

Таблица 2.2 – Описание класса Group

|  |  |  |
| --- | --- | --- |
| **Наименование поля** | **Тип хранимых данных** | **Описание** |
| Id | int | Идентификатор записи |
| CommandOwner | string | Владелец группы |
| CommandName | string | Имя группы |
| GroupParticipants | ICollection<User> | Участники группы |

Таблица 2.3 – Описание класса Message

|  |  |  |
| --- | --- | --- |
| **Наименование поля** | **Тип хранимых данных** | **Описание** |
| Id | string | Идентификатор записи |
| Sender | string | Отправитель |
| Recipient | string | Получатель |
| Sent | DateTime | Дата отправления |
| MessageBody | string | Тело сообщения |

Таблица 2.4 – Описание класса Project

|  |  |  |
| --- | --- | --- |
| **Наименование поля** | **Тип хранимых данных** | **Описание** |
| Id | int | Идентификатор записи |
| Participants | ICollection<User> | Участники проекта |
| Tasks | ICollection<ProjectTask> | Задачи проекта |
| ProjectOwner | string | Владелей проекта |
| Name | string | Имя проекта |
| ProjectStart | DateTime | Начало проекта |
| ProjectEnd | DateTime | Конец проекта |

Таблица 2.5 – Описание класса ProjectTask

|  |  |  |
| --- | --- | --- |
| **Наименование поля** | **Тип хранимых данных** | **Описание** |
| Id | int | Идентификатор записи |
| ProjectId | int | Идентификатор проекта |
| Participants | ICollection<User> | Исполнители задания |
| TaskName | string | Имя задания |
| Tags | ICollection<Tag> | Метки |
| Priority | TaskPriority | Приоритет |
| TaskStart | DateTime | Начало выполнения |
| TaskEnd | DateTime | Конец выполнения |

Таблица 2.6 – Описание класса Tag

|  |  |  |
| --- | --- | --- |
| **Наименование поля** | **Тип хранимых данных** | **Описание** |
| Id | int | Идентификатор записи |
| Name | string | Имя метки |
| OwnerId | string | Номер владельца |
| TagColor | KnownColor | Цвет метки |
| CalendarEvents | ICollection<CalendarEvent> | События календаря |
| ProjectTasks | ICollection<ProjectTask> | Задачи проектов |

Таблица 2.7 – Описание класса User

|  |  |  |
| --- | --- | --- |
| **Наименование поля** | **Тип хранимых данных** | **Описание** |
| Id | string | Идентификатор записи |
| FullName | string | ФИО |
| Birthday | DateTime | Дата рождения |
| Groups | ICollection<Group> | Группы пользователя |
| InProjects | ICollection<Project> | Проекты пользователя |
| UserProjects | ICollection<Project> | Проекты, созданные пользвателем |
| Tasks | ICollection<ProjectTask> | Задачи проектов |

Диаграмма классов предметной области показана на рисунке 2.4.

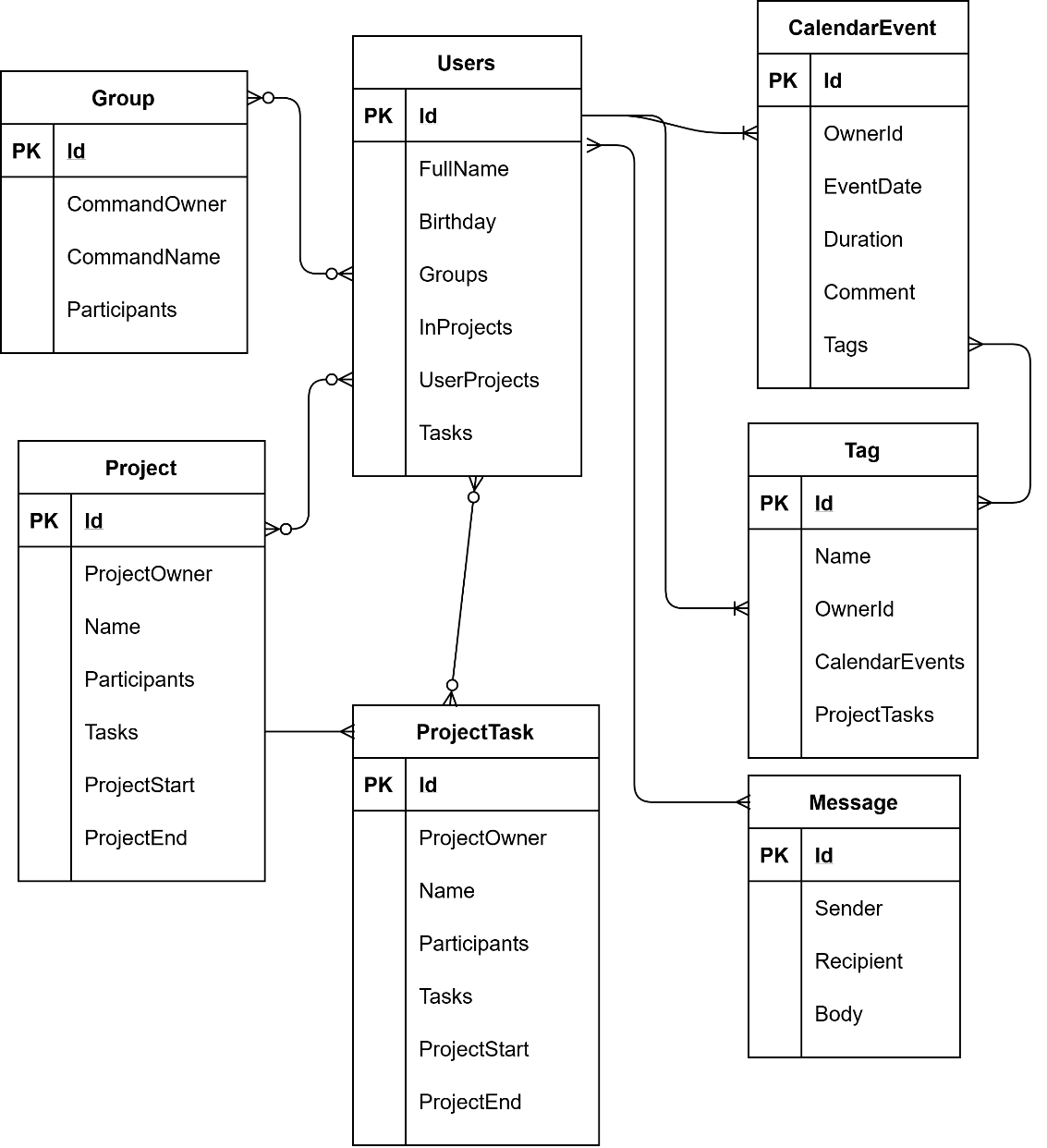


Рисунок 2.4 – Диаграмма классов предметной области

Опишем таблицы, отображающие классы предметной области в базе данных. Полученные таблицы:

* CalendarEvents;
* CalendarEventTag;
* Groups;
* Messages;
* ProjectTasks;
* ProjectTasksTag;
* Tags;
* UserGroup;
* UserProject;
* UserProjectTask;
* Users.

Поля таблиц базы данных описаны в таблицах 2.8 – 2.18. Схема БД (База Данных) на рисунке 2.5.

Таблица 2.8 – Поля таблицы CalendarEvents

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Первичный | Id | TEXT | нет |
| Внешний | OwnerId | TEXT | нет |
|  | EventDate | TEXT | нет |
|  | Duration | TEXT | нет |

Таблица 2.9 – Поля таблицы User

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Первичный | Id | TEXT | нет |
|  | FullName | TEXT | нет |
|  | Birthday | TEXT | нет |

Таблица 2.10 – Поля таблицы Group

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Первичный | Id | INTEGER | нет |
| Внешний | CommandOwner | TEXT | нет |

Таблица 2.11 – Поля таблицы Message

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Первичный | Id | TEXT | нет |
| Внешний | Sender | TEXT | да |
| Внешний | Recipient | TEXT | да |
|  | Sended | TEXT | нет |
|  | MessageBody | TEXT | да |

Таблица 2.12 – Поля таблицы Projects

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Первичный | Id | INTEGER | нет |
| Внешний | ProjectOwner | TEXT | да |
|  | Name | TEXT | да |
|  | ProjectStart | TEXT | нет |
|  | ProjectEnd | TEXT | нет |

Таблица 2.13 – Поля таблицы ProjectTasks

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Первичный | Id | INTEGER | нет |
| Внешний | ProjectId | INTEGER | нет |
| Внешний | TaskName | TEXT | да |
|  | Priority | INTEGER | нет |
|  | TaskStart | TEXT | нет |
|  | TaskEnd | TEXT | нет |

Таблица 2.14 – Поля таблицы ProjectTaskTag

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Первичный | ProjectTasksId | INTEGER | нет |
| Внешний | TagsId | INTEGER | нет |

Таблица 2.15 – Поля таблицы Tags

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Первичный | Id | INTEGER | нет |
| Внешний | OwnerId | TEXT | да |
|  | TagColor | INTEGER | нет |

Таблица 2.16 – Поля таблицы UserGroup

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Первичный | GroupParticipantsId | TEXT | нет |
| Внешний | GroupsId | INTEGER | нет |
| Внешний | GroupParticipantsId | TEXT | нет |

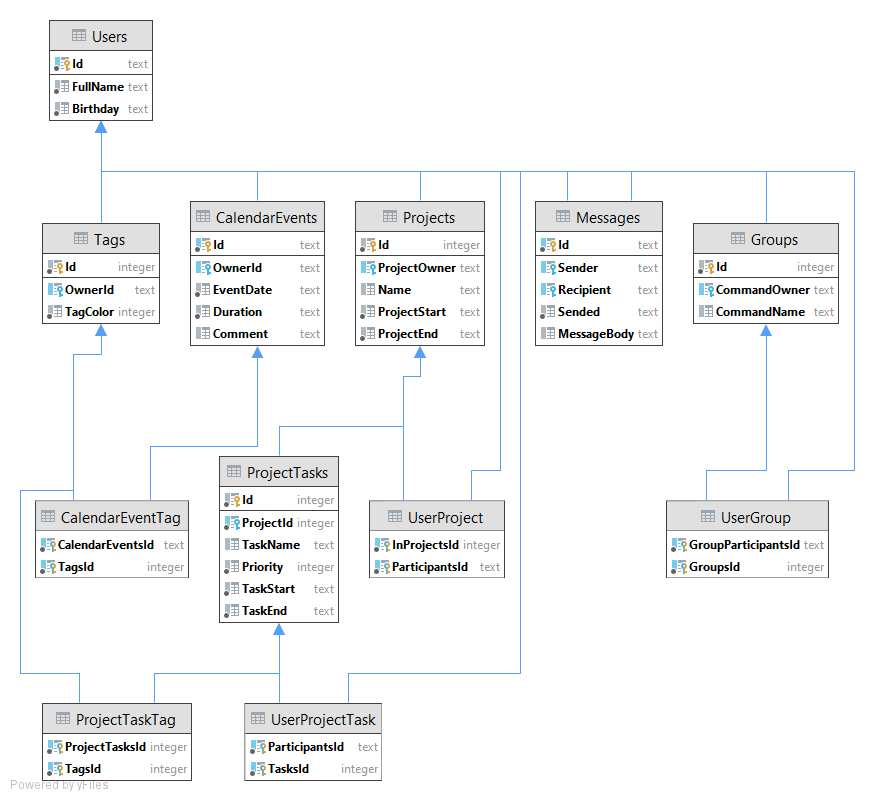


Рисунок 2.5 – Диаграмма базы данных

Таблица 2.17 – Поля таблицы UserProject

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Первичный | InProjectsId | INTEGER | нет |
| Внешний | ParticipantsId | TEXT | нет |

Таблица 2.18 – Поля таблицы UserProjectTask

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Первичный | ParticipantsId | TEXT | нет |
| Внешний | TasksId | INTEGER | нет |

Таблица 2.19 – Поля таблицы CalendarEventTag

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Первичный | CalendarEventId | INTEGER | нет |
| Внешний | TagId | TEXT | нет |

Так как на уровне пользовательского интерфейса для аутентификации и авторизации используется Microsoft Identity, то был создан ещё один контекст данных, в котором хранятся данные о веб-пользователях. Описание таблиц, созданных Identity в таблицах 2.20 – 2.26. Схема БД на рисунке 2.6.

Таблица 2.20 – Поля таблицы AspNetRoleClaims

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Первичный | Id | TEXT | нет |
| Внешний | RoleId | TEXT | нет |
|  | ClaimType | TEXT | нет |
|  | ClaimValue | TEXT | нет |

Таблица 2.21 – Поля таблицы AspNetRoles

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Первичный | Id | TEXT | нет |
|  | Name | TEXT | нет |
|  | NormalizedName | TEXT | нет |
|  | ConcurrencyStamp | TEXT | нет |

Таблица 2.22 – Поля таблицы AspNetUserClaims

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Первичный | Id | TEXT | нет |
| Внешний | RoleId | TEXT | нет |
|  | ClaimType | TEXT | нет |
|  | ClaimValue | TEXT | нет |

Таблица 2.23 – Поля таблицы AspNetUserLogins

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Первичный | LoginProvider | TEXT | нет |

Продолжение таблицы 2.15

|  |  |  |  |
| --- | --- | --- | --- |
| Первичный | ProviderKey | TEXT | нет |
|  | ProviderDisplayName | TEXT | нет |

Таблица 2.24 – Поля таблицы AspNetUserRoles

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Внешний | UserId | TEXT | нет |
| Внешний | RoleId | TEXT | нет |

Таблица 2.25 – Поля таблицы AspNetUserTokens

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Внешний | UserId | TEXT | нет |
| Первичный | LoginProvider | TEXT | нет |
|  | Name | TEXT | нет |
|  | Value | TEXT | нет |

Таблица 2.26 – Поля таблицы AspNetUsers

|  |  |  |  |
| --- | --- | --- | --- |
| **Ключевое поле, тип ключа** | **Наименование поля** | **Тип хранимых данных** | **Может принимать значение NULL** |
| Первичный | Id | TEXT | нет |
|  | Email | TEXT | нет |
|  | EmailConfirmed | TEXT | нет |
|  | PasswordHash | TEXT | нет |
| Внешний | UnderlyingUserId | INT | нет |
|  | UserName | TEXT | нет |
|  | PhoneNumber | TEXT | нет |

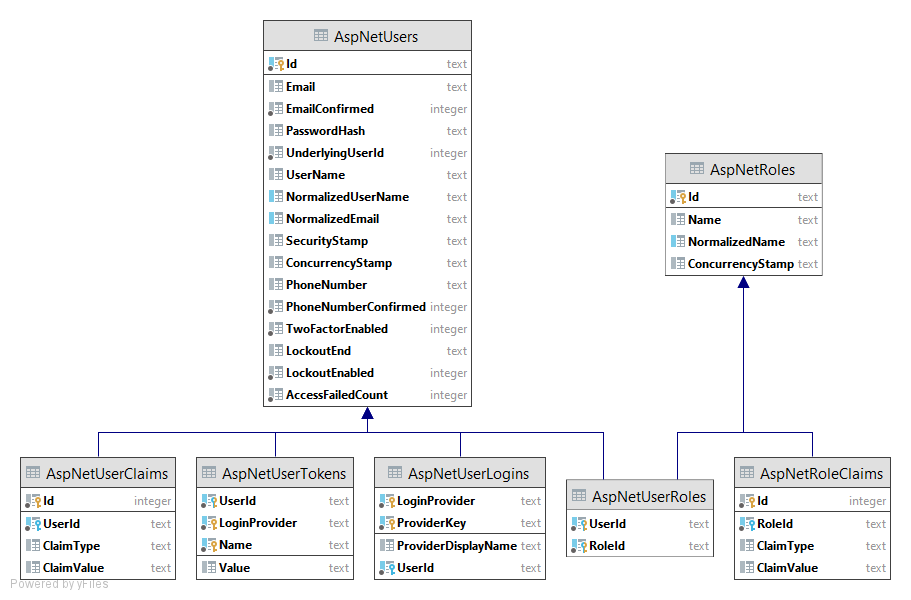
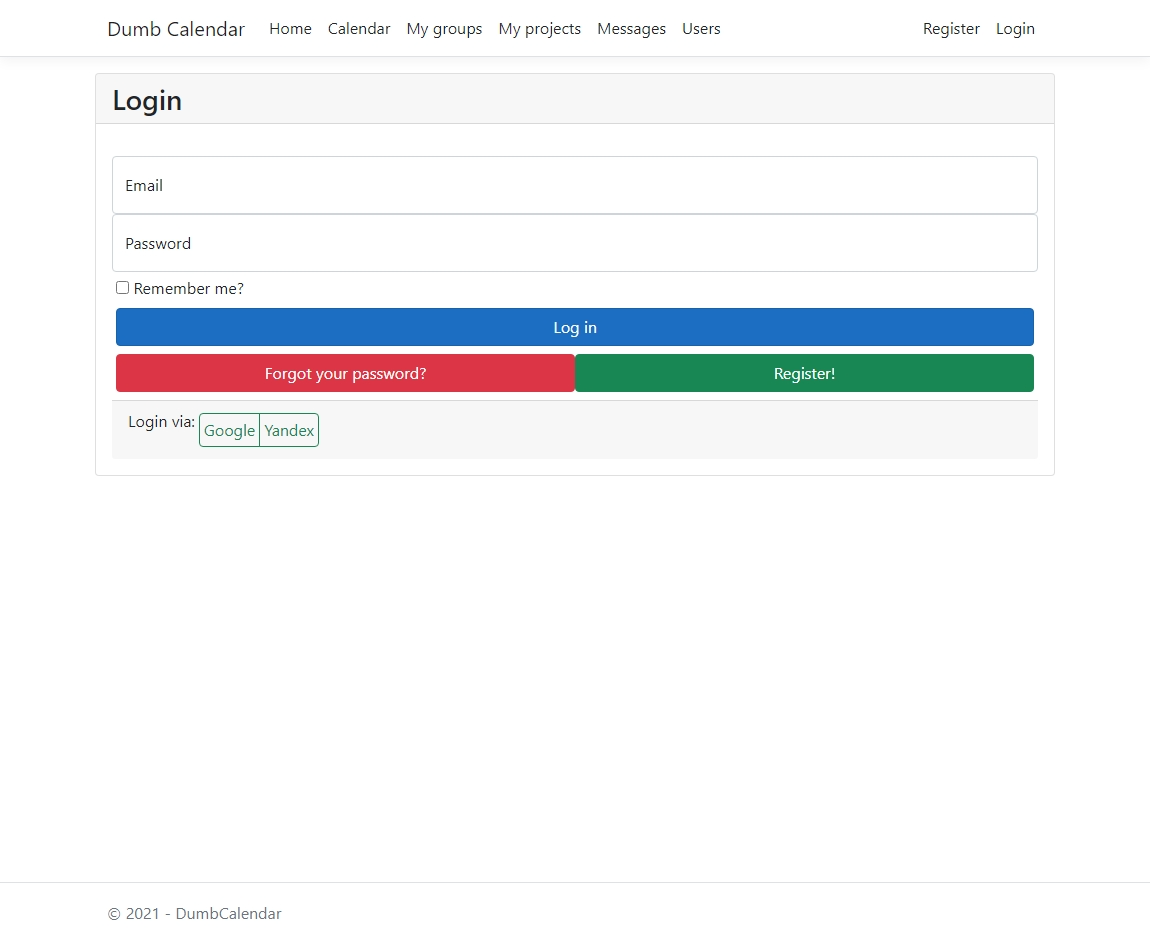
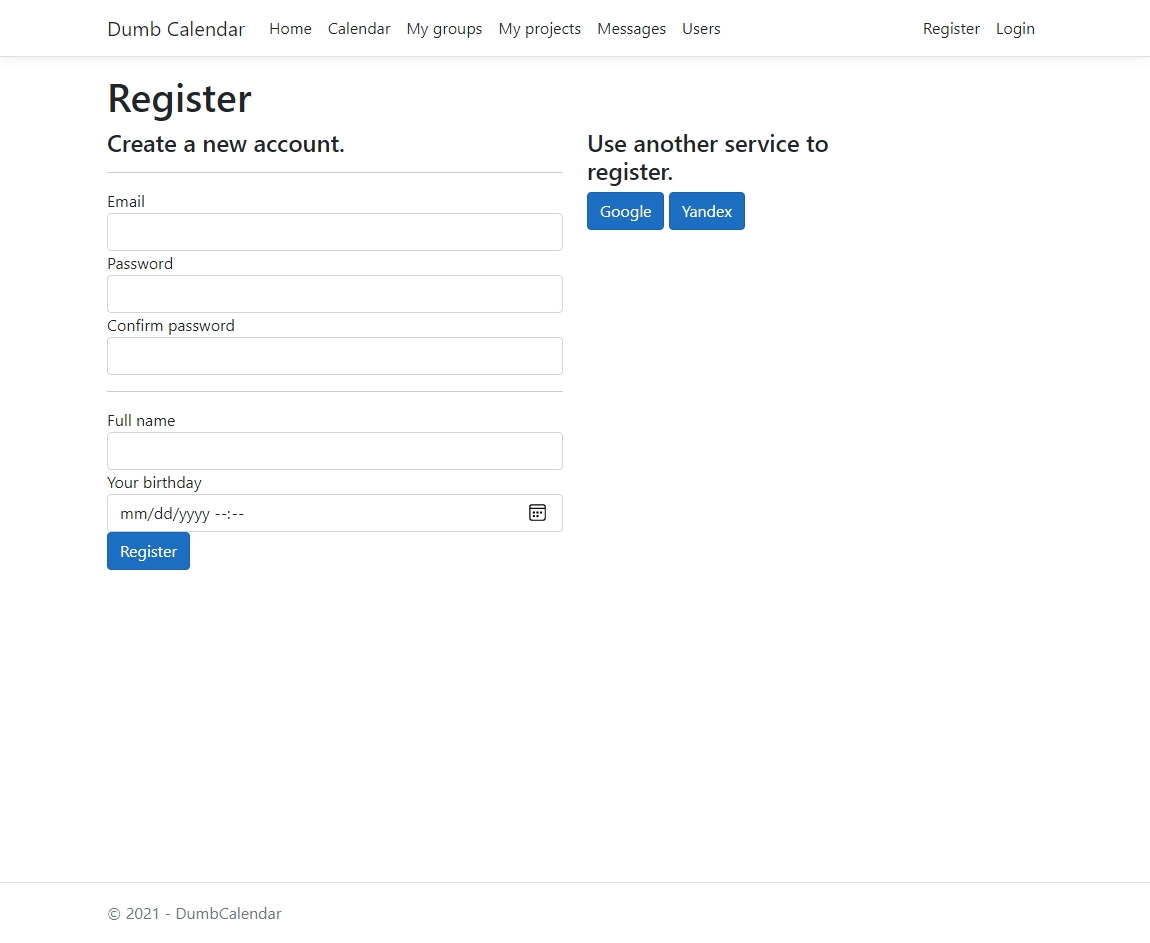
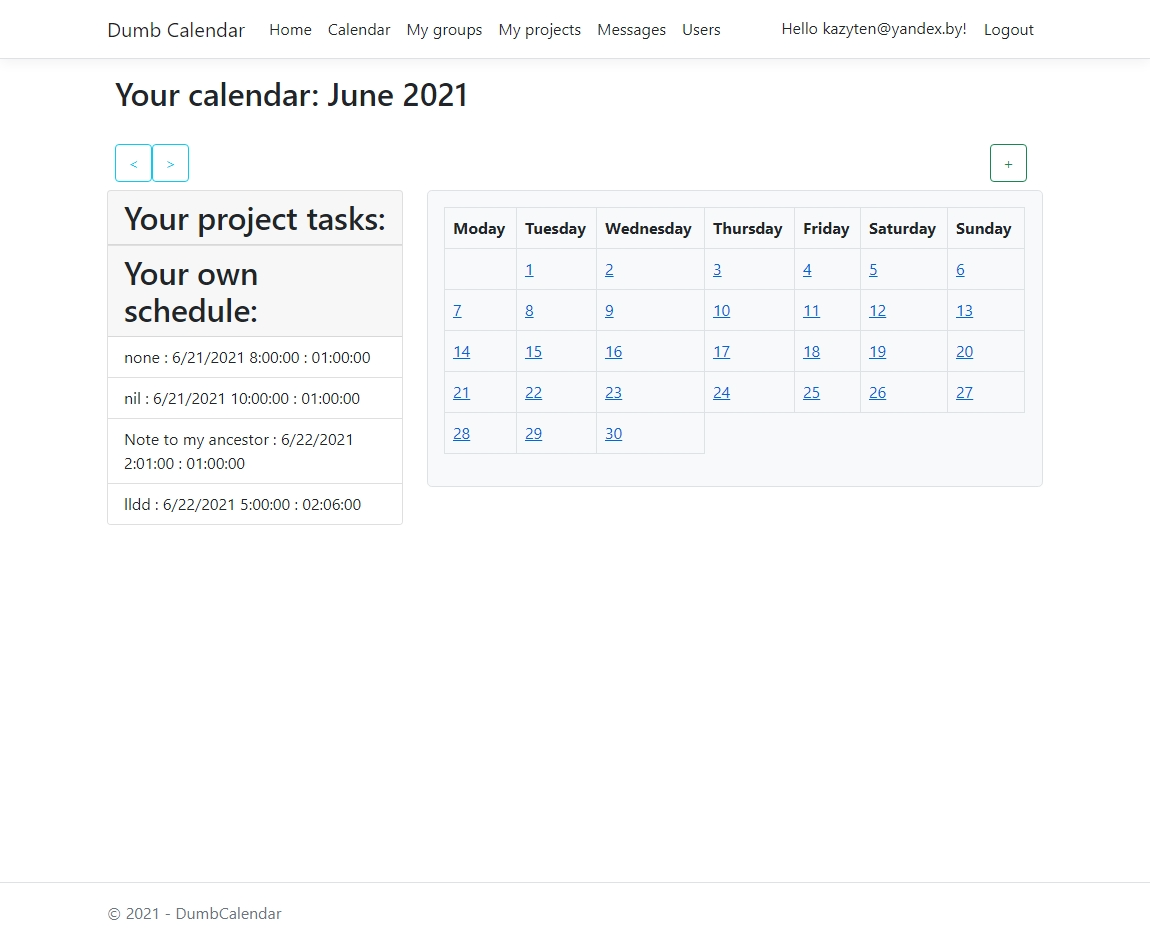


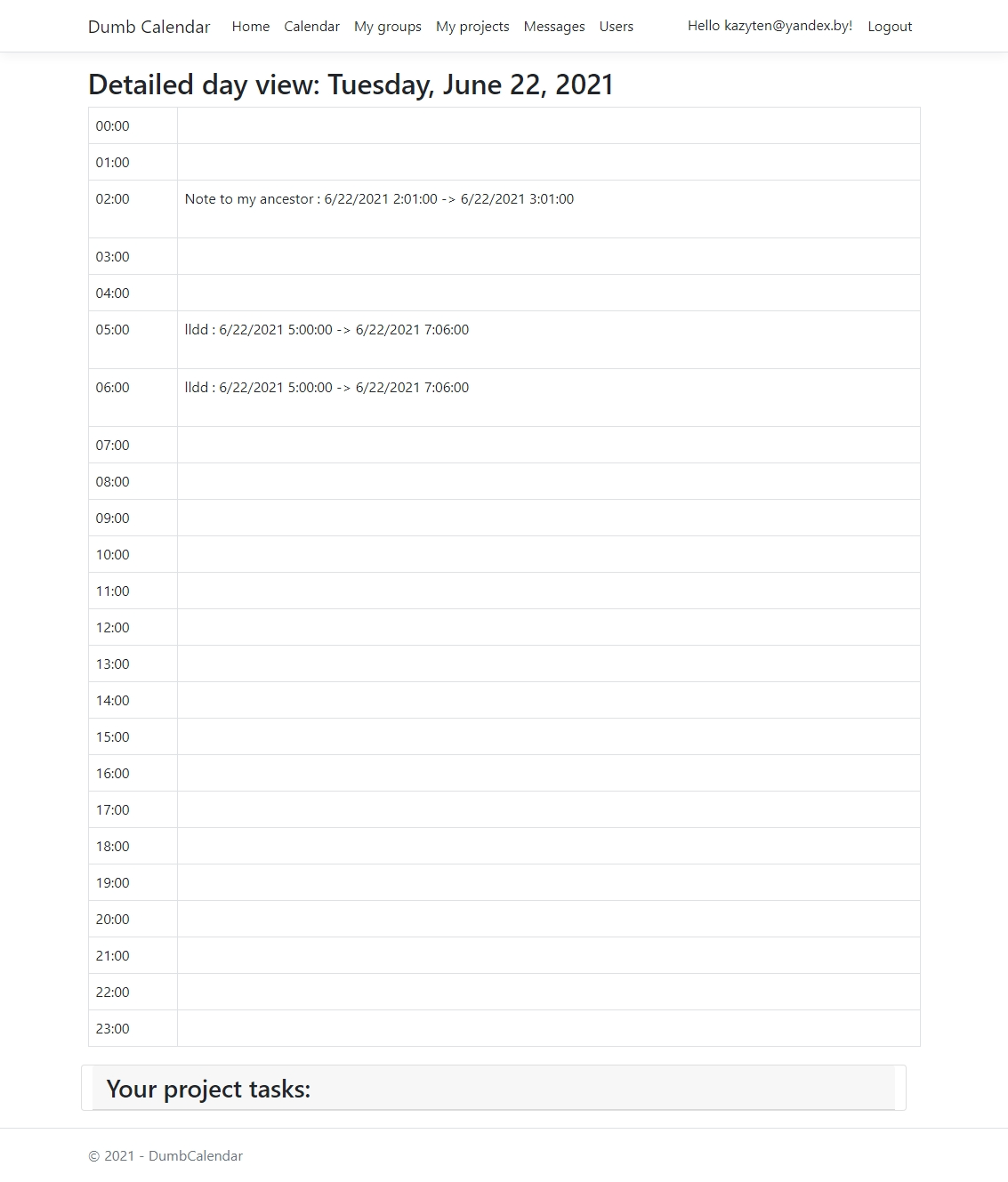
Рисунок 2.6 – Физическая схемы базы данных Asp.Net Core Identiмty

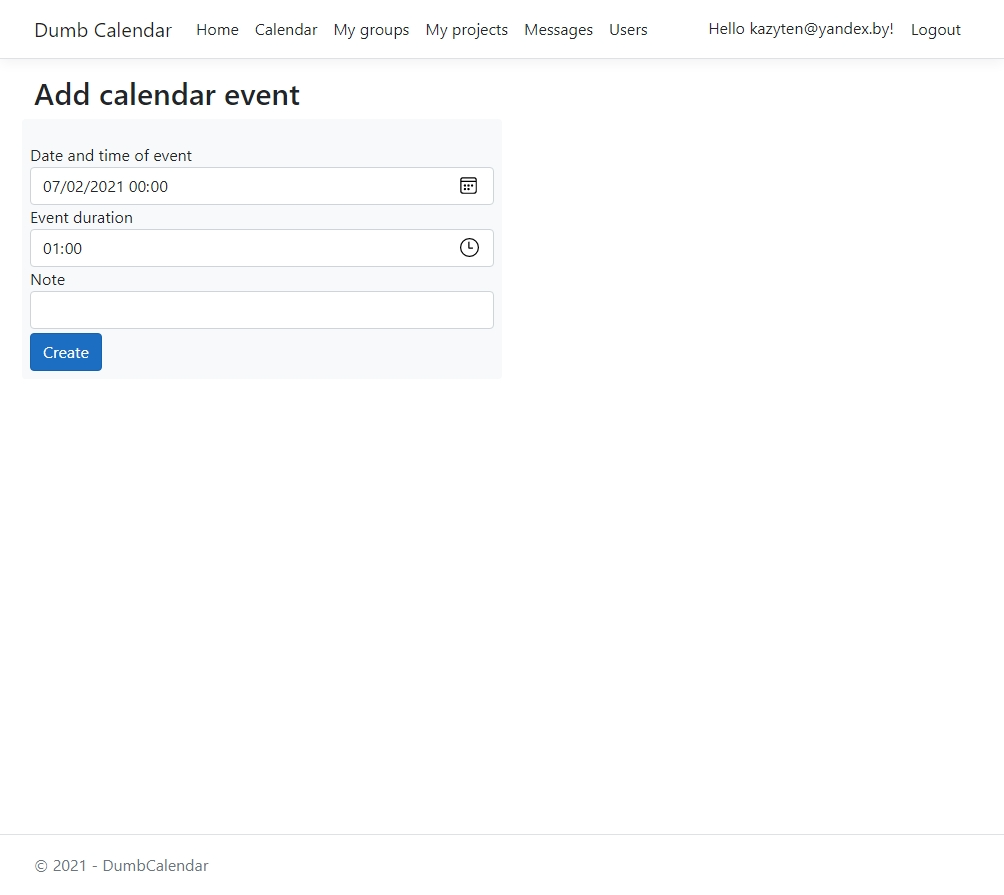
## **Описание интерфейса пользователя**

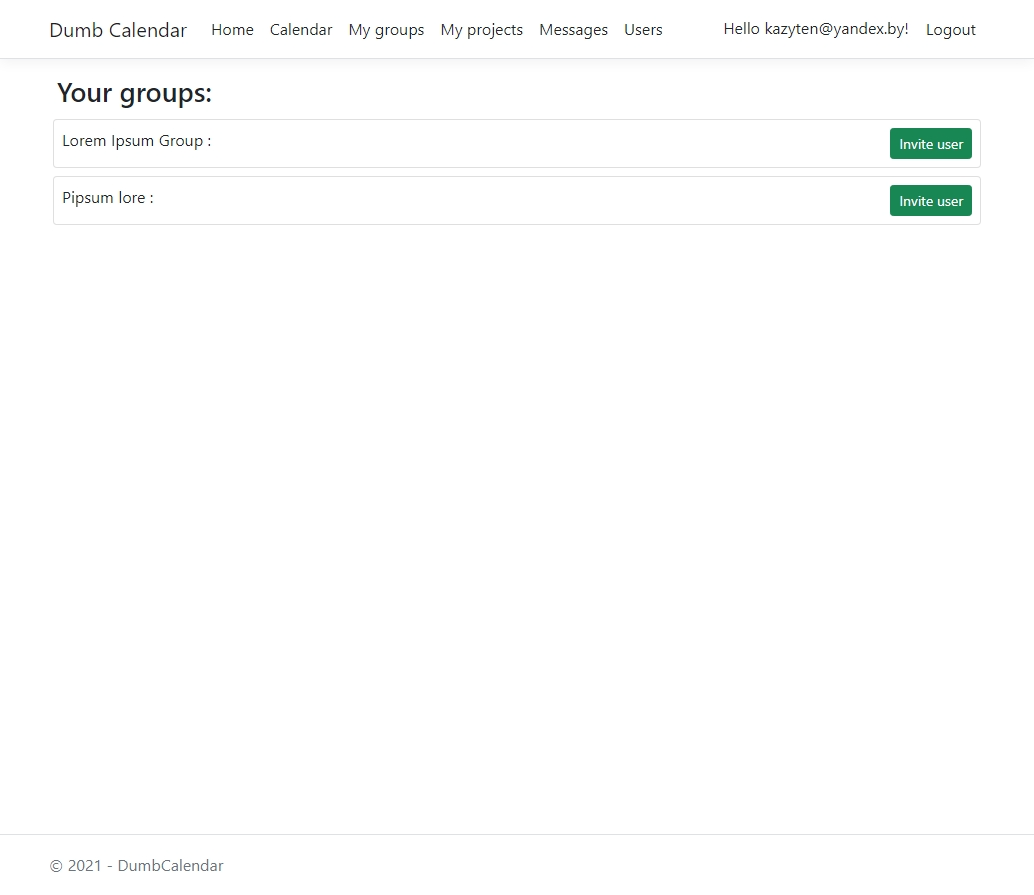












# ЗАКЛЮЧЕНИЕ

В ходе прохождения технологической практики на ИООО «ЕПАМ Системз» я познакомился с его организационной структурой, отделами, рабочими процессами.

Также были закреплены и углублены теоретические и практические знания в области разработки приложений.

При выполнении индивидуального задания была спроектирована база данных, проведён анализ предметной области, спроектирована структура приложение, которое затем было реализовано с использованием объектно-ориентированного языка C#, веб-фреймворка Asp.Net Core MVC, интеллектуальной среды разработки Jetbrains Rider.

Разрабатывая приложение было протестировано, найденные ошибки были исправлены, обработчики исключительных ситуаций были добавлены.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1 Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес; [пер. с англ.: А. Слинкин науч. ред.: Н. Шалаев]. – Санкт-Петербург: Питер, 2014. – 366 с.: ил.

2 Новиков, Б. А. **Основы технологий баз данных: учеб. пособие / Б. А. Новиков, Е. А. Горшкова, Н. Г. Графеева; под ред. Е. В. Рогова. — 2-е изд. — М.: ДМК Пресс, 2020. — 582 с.**

3 Чамберс, Д. ASP.NET Core. Разработка приложений. ASP.NET Core /

Д. Чамберс, Д. Пэкетт, С. Тиммс; [пер. с англ.: Е. Матвеев]. – СПб.: Питер, 2018. — 434 с.: ил.

4 Руководство по ASP.NET Core 5, [METANIT.COM](https://metanit.com/) [Электронный ресурс] - Режим доступа: https://metanit.com/sharp/aspnet5 – Дата доступа: 18.04.2021

5 Симан, М. Внедрение зависимостей в.NET. Внедрение зависимостей / М. Симан – СПб.: Питер, 2014. — 464 с.: ил.

# Приложение А