




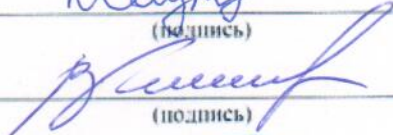

Институт
интеллектуальных кибернетических систем
Кафедра №22 «Кибернетика»

Направление подготовки 09.04.04 Программная инженерия

Пояснительная записка

к научно-исследовательской работе студента на тему:

Разработка сервиса планирования рабочего времени специалиста помогающей профессии с возможностью бронирования времени клиентом на основе алгоритмов распределенных синхронизаций

Группа	M22-514		
Студент	 (подпись)	Саутов Д.С. (ФИО)	
Руководитель	 (подпись)	Климов В.В. (ФИО)	
Научный консультант	 (подпись)	Тихомирова Д.В. (ФИО)	
Оценка руководителя	30 (0-30 баллов)	Оценка консультанта	30 (0-30 баллов)
Итоговая оценка	ECTS		
	(0-100 баллов)		

Комиссия

Председатель	_____ (подпись)	_____ (ФИО)
	_____ (подпись)	_____ (ФИО)
	_____ (подпись)	_____ (ФИО)
	_____ (подпись)	_____ (ФИО)

Москва 2022



КАФЕДРА КИБЕРНЕТИКИ

Институт интеллектуальных кибернетических систем

Задание на НИР

Студенту гр. M22-514

(группа)

Саутову Давиду Сергеевичу

(фио)

ТЕМА УИР

Разработка сервиса планирования рабочего времени специалиста помогающей профессии с возможностью бронирования времени клиентом на основе алгоритмов распределенных синхронизаций

ЗАДАНИЕ

№ п/п	Содержание работы	Форма отчетности	Срок исполнения	Отметка о выполнении Дата, подпись
1.	Аналитическая часть			
1.1.	Анализ предметной области и постановка задачи	Текст РСПЗ	13.10.22	
1.2.	Изучение и анализ рынка имеющихся решений, выделение их преимуществ и недостатков	Список преимуществ и недостатков	13.10.22	
1.3.	Изучение алгоритмов распределенной синхронизации	Сравнительная таблица	20.10.22	
1.4.	<i>Оформление расширенного содержания пояснительной записки (РСПЗ)</i>	Текст РСПЗ	20.10.22	
2.	Теоретическая часть			
2.1.	Выделение основных требований к алгоритму распределенной синхронизации и выбор алгоритма распределенной синхронизации	Раздел ПЗ	27.10.22	
2.2.	Проектирование ER-моделей предметной области	ER-диаграммы	03.11.22	
3.	Инженерная часть			
3.1.	Разработка требований к разрабатываемому программному продукту и выбор инструментов разработки	Раздел ПЗ	10.11.22	
3.2.	Представление моделей разрабатываемого программного продукта	Диаграмма последовательности	17.11.22	

4.	Технологическая и практическая часть			
4.1.	Разработка пользовательского интерфейса	Исполняемые файлы, Исходный код	24.11.22	
4.2.	Разработка программного продукта и его характеристика	Исполняемые файлы, Исходный код	08.12.22	
4.3.	Взаимодействие различных классов пользователей с интерфейсом, а также сценарии использования ПО	Исполняемые файлы, Исходный код	08.12.22	
4.4.	Проверка работоспособности программного продукта и проведение тестирования	Протоколы испытаний	15.12.22	
5.	<i>Оформление пояснительной записки (ПЗ) и иллюстративного материала для доклада.</i>	Текст ПЗ, презентация	15.12.22	

ЛИТЕРАТУРА

1.	Бёрнс Б. Распределенные системы. Паттерны проектирования. / Бёрнс Б. – СПб: Питер, 2021 – 224 с.
2.	Косяков М.С. Введение в распределенные вычисления. / М.С. Косяков – СПб: НИУ ИТМО, 2014 -155с.
3.	Фаулер М. UML. Основы: краткое руководство по стандартному языку объектного моделирования / Мартин Фаулер – Пер. с англ. – СПб: Символ-Плюс, 2011 – 184 с.
4.	Куликов С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов 3-е изд. - Минск: Четыре четверти, 2020 - 312 с.
5.	Хлудова М.В. Синхронизация распределённых приложений реального времени // Информатика, телекоммуникации и управление. / М.В. Хлудова – 2009. – №6. – С. 88–92.
6.	Чамберс Дж. ASP.NET Core Разработка приложений / Джеймс Чамберс, Дэвид Пэккетт, Саймон Тиммс. - СПб.: Питер, 2018. — 464 с.: ил. - (Серия «Для профессионалов»).
7.	Бабичев С. Л. Распределенные системы : Учебное пособие для вузов / Бабичев С. Л., Коньков К. А.. - Москва : Юрайт, 2020. - 507 с
8.	Resca S. Hands-On RESTful Web Services with ASP.NET Core 3: Design production-ready, testable, and flexible RESTful APIs for web applications and microservices / Samuele Resca - Packt Publishing; 1st edition, 2019. – 780 p. – ISBN-13 978-1789537611
9.	Himschoot P. Blazor Revealed: Building Web Applications in .NET / Peter Himschoot - Apress; 1st ed. Edition, 2019. – 285 p. – ISBN-13 978-1484243428
10.	Litvinavicius T. Exploring Blazor: Creating Hosted, Server-side, and Client-side Applications with C# / Taurius Litvinavicius, Apress; 1st ed. Edition, 2019. – 199 p. – ISBN-13 - 978-1484254455
11.	Trivedi J. Building A Web App With Blazor And Asp .Net Core: Create A Single Page App With Blazor Server And Entity Framework Core: Create a Single Page App with ... and Entity Framework Core / Jignesh Trivedi, BPB Publications, 2020. – 220 p. – ISBN-13 - 978-9389845457
12.	Washington M. An Introduction to Building Applications with Blazor: How to get started creating applications using this exciting easy to use Microsoft C# framework / Michael Washington, BlazorHelpWebsite.com, 2019. – 339 p. ISBN - 1688540040
13.	Engström J. Web Development with Blazor: A hands-on guide for .NET developers to build interactive UIs with C# / Jimmy Engström, Packt Publishing Limited, 2021. – 310 p. – ISBN-13 - 978-1800208728

14.	Pattankar M. Mastering ASP.NET Web API: Build powerful HTTP services and make the most of the ASP.NET Core Web API platform / Mithun Pattankar, Malendra Hurbuns, Packt Publishing; 1st edition, 2017. 332 p. – ISBN - 1786463954
15.	Price M. C# 11 and .NET 7 – Modern Cross-Platform Development Fundamentals: Start building websites and services with ASP.NET Core 7, Blazor, and EF Core 7 / Mark J. Price, Packt Publishing; 7th edition, 2022. 818 p. – ISBN-13 - 978-1803237800
16.	RFC 7519. JSON Web Token (JWT). 2015. . [Электронный ресурс]. URL: https://datatracker.ietf.org/doc/html/rfc7519.html (дата обращения 22.12.2022).
17.	Janoky L. V., Levendovszky J., Ekler P. A Novel JWT Revocation Algorithm [Электронный ресурс]. URL: http://real.mtak.hu/113127/1/CSCS2020.pdf (дата обращения: 22.12.2022)
18.	Janoky L. An analysis on the revoking mechanisms for JSON Web Tokens / Janoky L. V., Levendovszky J., Ekler P., International Journal of Distributed Sensor Networks, 14(9), 2018. – 10 p.
19.	Тестирование программного обеспечения. Базовый курс / С. С. Куликов. — 3-е изд. — Минск: Четыре четверти, 2020. — 312 с.
20.	Осипов Д. Технологии проектирования баз данных. / Осипов Д. Л., – М.: ДМК Пресс, 2019. – 498 с.
21.	Free Blazor Components 70+ controls by Radzen [Электронный ресурс]. URL: https://blazor.radzen.com (дата обращения: 22.12.2022)
22.	Overview of ASP.NET Core [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0 (дата обращения: 22.12.2022)
23.	ASP.NET Core Blazor [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/aspnet/core/blazor/?view=aspnetcore-7.0 (дата обращения: 22.12.2022)
24.	Tutorial: Create a web API with ASP.NET Core [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/aspnet/core/tutorials/first-web-api?view=aspnetcore-7.0&tabs=visual-studio (дата обращения: 22.12.2022)
25.	Выбор пользовательского веб-интерфейса ASP.NET Core [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/aspnet/core/tutorials/choose-web-ui?view=aspnetcore-7.0 (дата обращения: 22.12.2022)


Дата выдачи задания:

« 18 » сентября 2022г.

Руководитель

Научный
консультант

Студент





Климов В.В.
(ФИО)

Тихомирова Д.В.
(ФИО)

Сауров Д.С.
(ФИО)

Отчет о проверке на заимствования №1



Автор: Sautov David

Проверяющий: Sautov David

Отчет предоставлен сервисом «Антиплагиат» - <http://users.antiplagiat.ru>

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 1
Начало загрузки: 03.01.2023 13:47:52
Длительность загрузки: 00:00:01
Имя исходного файла:
M22-514_СаутовДС_ПЗ.pdf
Название документа: M22-514_СаутовДС_ПЗ
Размер текста: 82 кБ
Символов в тексте: 83716
Слов в тексте: 10688
Число предложений: 785

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Начало проверки: 03.01.2023 10:47:55
Длительность проверки: 00:00:21
Комментарии: не указано
Модули поиска: Интернет Free



СОВПАДЕНИЯ

0,13%

САМОЦИТИРОВАНИЯ

0%

ЦИТИРОВАНИЯ

0%

ОРИГИНАЛЬНОСТЬ

99,87%

Совпадения — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Самоцитирования — доля фрагментов текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты; общеупотребительные выражения; фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.
Совпадения, самоцитирования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в отчете	Источник	Актуален на	Модуль поиска	Комментарии
[01]	0,13%	руOpenRPA tutorial. Управление WEB приложениями / Хабр https://habr.com	07 Сен 2020	Интернет Free	
[02]	0%	Веб-приложение http://ru.wikipedia.org	20 Авг 2020	Интернет Free	Источник исключен. Причина: Маленький процент пересечения.
[03]	0%	не указано http://dspace.susu.ru	08 Ноя 2018	Интернет Free	Источник исключен. Причина: Маленький процент пересечения.

Еще источников: 2

Еще совпадений: 0%

Реферат

Пояснительная записка содержит:

- 44 страниц (61 с приложением)
- 22 рисунка
- 4 таблицы
- 1 приложение
- 25 используемых источников

Ключевые слова:

Централизованный алгоритм синхронизации – Алгоритм распределенных систем синхронизации, где присутствует один узел, который распределяет ресурсы между другими узлами системы.

Централизованный алгоритм – алгоритм распределенной синхронизации

Веб-API - это программный интерфейс для веб-сервера.

БД – База данных, необходимая для хранения данных. Управляется СУБД.

REST - это программная архитектура, которая определяет условия работы API.

Blazor - Бесплатная веб-платформа с открытым исходным кодом, позволяющая разработчикам создавать веб-приложения с использованием C# и HTML.

Веб-приложение - клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера.

HTTP - Протокол прикладного уровня передачи данных

JWT – JSON Web Token — это открытый стандарт (RFC 7519) для создания токенов доступа, основанный на формате JSON.

Целью данной работы является создание сервисов для планирования рабочего времени специалистов (Сервис А), бронирование помещения, где будет проходить прием (Сервис Б), а также сервис, который будет производить распределение ресурсов сервисов А и Б между клиентами. Распределение ресурсов будет реализована при помощи одного из алгоритмов распределенной синхронизации.

ВВЕДЕНИЕ

В 21 веке большую популярность приобрело создание различных сервисов. Под сервисом считается некоторая сущность, которая выполняет набор задач, за которые несет ответственность данный сервис, а также он должен предоставлять интерфейс для взаимодействия с ним. Сервисам принято противопоставлять монолитную структуру, по сравнению с которой сервисы имеют ряд преимуществ, но также и имеют недостатки. Множество компаний используют сервисы для предоставления своих услуг клиентам. Их использование имеет множество преимуществ, такие как: автоматизации различных процессов и их разделение, доступностью и возможность блочного обновления системы, по сравнению с монолитной структурой. Также возможна ситуация, когда клиенту необходимо воспользоваться услугами нескольких компаний определенным образом, при этом услуги данных компаний имеют некоторую связь. Примером такой ситуации можно считать планирование своего отпуска, где клиенту необходимо заказать билеты на транспортное средство, а также забронировать гостиницу или отель. Стоит отметить, что клиенту приходится рассматривать множество зависимостей, такие как время, месторасположение, продолжительность полета и проживания, что усложняет данную задачу и клиент может что-то не учесть и совершить ошибку, которая потом скажется на его отпуске. В данной работе будет представлен похожий пример, при этом данная проблема, а также большая часть задач на учет зависимостей между двумя сервисами будет возложена на программную составляющую, что облегчит возможность бронирования и заказа услуг для клиентов.

В данной работе будет создано два независимых друг от друга сервиса. Первый из которых позволит искать определенных специалистов из различных областей, а также создавать к ним записи на выбранное время. Второй же сервис позволит бронировать различные помещения, которые находятся в различных локациях. Именно в этих помещениях и будет проходить прием у специалиста. Также в данной работе необходимо разработать систему, которая будет сопоставлять данные из двух различных сервисов, учитывать их зависимости, а также бронировать помещения и время специалиста для получения различных услуг.

Для синхронизации данных и учета всех зависимостей между сервисами предлагается использовать концепции, представленные в алгоритмах распределенной синхронизации. Они позволят определить, какие ресурсы доступны пользователю, а какие заняты, а также позволят избежать ситуации бронирования кабинета на одно и то же время различными клиентами и избежать случая, когда на один временной интервал специалиста записалось несколько клиентов.

В первом разделе пояснительной записки рассматривается аналитическая часть, а именно рассматриваются похожие решения на рынке, их преимущества и недостатки. Будут рассмотрены различные алгоритмы распределенной синхронизации, при этом будут выделены их сильные и слабые стороны, что потом позволит выбрать наилучший из них для синхронизации сервисов.

Во втором разделе пояснительной записки будет рассматриваться теоретическая составляющая. Будет выделены требования к алгоритму синхронизации между двумя сервисами, что позволит выбрать один из алгоритмов распределенной синхронизации из множества тех, что рассматривались в 1 разделе. Также будут разработаны схемы, отображающие устройство сервисов и их взаимодействие друг с другом и пользователем.

В третьем разделе будут рассмотрены и обоснованы инструменты для разработки, языки программирования, а также будут указаны программные требования к ожидаемому продукту (указанные на различных уровнях). Будет представлена архитектура разрабатываемой системы, а также ее компонентов.

В четвертом разделе будет дана краткая характеристика разрабатываемому программному обеспечению и представлены различные классы пользователей, а также сценарии использования данного ПО. По итогам разработки будет проведено тестирование.

1 Анализ предметной области

В данном разделе рассматривается аналитическая составляющая поставленной задачи. В нем рассматриваются и анализируются предметная область, ее составляющие и выделение основной проблемы, и ее решение для решения поставленной задачи. Также в данном разделе рассматривается такое понятие, как распределенная система. Основные задачи, которые она решает, и какие проблемы встречаются при использовании распределенных систем и способы их решения. В данном разделе также сформулированы основные требования к алгоритму синхронизации, а также рассмотрено, можно ли использовать алгоритмы распределенной синхронизации и необходимо ли это для решения поставленной задачи. В конце данного раздела проведено сравнение с выделением преимуществ и недостатков различных алгоритмов распределенной синхронизации.

1.1 Анализ предметной области и постановка задачи

В данном подразделе проводится анализ предметной области. В нем будет расписан из чего она состоит, основные проблемы, которые в ней встречаются и способы их решения. Тут рассматривается такое понятие как распределенная система, сфера их использования и проблемы, которые она позволяет решить. Для решения проблем, которые были выделены в ходе анализ предметной области предлагается использовать алгоритмы распределенной синхронизации. Также будет проведена постановка задачи.

Предметная область содержит в себе два независимых друг друга сервиса, которые необходимо, некоторым образом, синхронизировать, для получения клиентом желаемого результата.

Всего имеется 2 сервиса:

- 1) для записи к специалисту на определенное время и промежуток времени
- 2) для бронирования комнаты в некоторой локации на определенное время, в которой будет проходить встреча со специалистом.

На данном этапе можно встретить несколько проблем:

1. Клиент забронировал комнату в локации А на время t_0 , но необходимые специалисты заняты в момент времени t_0 .
2. Клиент записался к специалисту на время t_0 , но в желаемой локации А нету свободных комнат на время t_0 .
3. Клиент забронировал комнату в локации А на момент времени t , также записался к необходимому специалисту, который тоже не занят во время t , но он находится в локации Б, и он не успевает приехать ко времени t в локацию А.

4. Два клиента хотят записаться к одному и тому же специалисту на время t , специалист сумеет быть только на одной записи. То есть придется решить, кто из клиентов получит право записи к специалисту.
5. Два специалиста хотят забронировать помещение А в локации Б, придется определить кому из клиентов достанется бронь данного помещения.

В ходе анализа данных проблем можно выделить ресурсы, такие как время специалистов и комнаты в локациях, которые можно забронировать на время.

Разделение данных ресурсов между клиентами и избежание вышеперечисленных проблем и будет задачей данной практики.

Для понимания того, как распределить данные ресурсы можно обратить внимание на такой класс систем, как распределенные системы.

Распределенной системой можно назвать набор компьютерных программ, которые используют вычислительные ресурсы различных вычислительных узлов, для достижение поставленной цели. Под распределенной системой обычно понимают какой-нибудь набор из нескольких вычислительных машин, соединённых между собой, некоторым образом, для обмена сообщениями и вычисления какой-то сложной задачи.

Во время работы распределенной системы, могут возникать конфликты по доступу к определенным ресурсам, к примеру один узел может попытаться считать нужные ей данные, которые в текущий момент изменяет другим узлом. Итог данного чтения может нести непредсказуемый результат и приводить к ошибкам. Для решения таких конфликтов по доступу к ресурсам используются алгоритмы распределённой синхронизации.

Распределенные системы и алгоритмы их синхронизации позволяют различному количеству узлов делить между собой общие ресурсы так, чтобы не нарушать работу других узлов в данной системе. Именно это и необходимо для решения нашей задачи.

Для выполнения данной задачи предлагается смоделировать распределенную систему, в которой будут 2 узла, предоставляющие собой сервисы, которые будут делить такие ресурсы, как время специалистов и комнаты в локациях. Также к этим узлам будут обращаться N других узлов - клиенты, которым необходимы данные ресурсы. Для решения конфликтов по доступу к одним и тем же ресурсам будут использоваться методы и алгоритмы синхронизации и распределения ресурсов для распределенных систем.

Подводя итоги, задачей данной работы будет создание 2 сервисов и клиентского приложения (далее сервис С):

Сервис А нужен для записи к специалисту на определенный интервал времени. Будет считаться отдельным узлом в распределенной системе.

Сервис Б нужен для бронирования комнаты на определенный интервал времени. Будет считаться отдельным узлом в распределенной системе.

Также необходимо разработать клиентское приложение, через который клиент и будет взаимодействовать с системой и позволит им записываться к специалистам и бронировать комнату, он будет обращаться к сервисам А и Б и распределять их ресурсы между клиентами на основе алгоритмов распределенной синхронизации. Одновременно может работать N клиентов, при этом каждый клиент будет считаться отдельным узлом в распределённой системе и будет иметь связь с сервисам А и Б при помощи клиентского приложения.

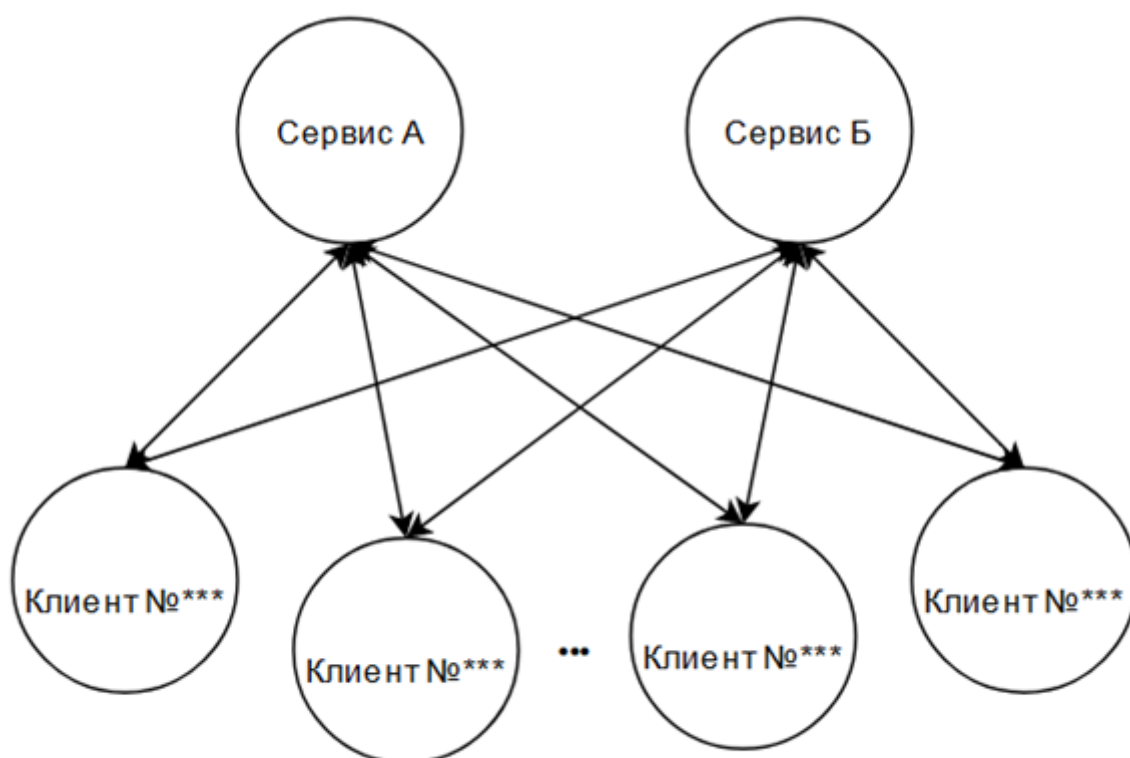


Рис. 1. Представление предметной области в виде распределенной системы

1.2 Изучение и анализ рынка имеющихся решений, выделение их преимуществ и недостатков

В данном подразделе рассматривается некоторая часть рынка, из которой берутся основные аналоги и решения, схожие с системой, которую необходимо реализовать в ходе данной работы. Они анализируются и для данных решений выделяются их сильные и слабые стороны. Именно анализ решений на рынке позволяет определить узкие места в данной сфере, которые в дальнейшем необходимо решить в процессе данной работы.

На рынке в качестве аналогичных систем, которых можно использовать для решения поставленной задачи можно выделить системы бронирования комнат, переговорных и конференц-залов.

Именно эти системы позволяют распределять такие ресурсы, как комнаты, находящиеся в определенных локациях, и время специалистов компании и других ее сотрудников.

Минусом данных систем является то, что в них не учитывается логистика и время пути между локациями для специалистов. Также, в случае если специалистам платят не фиксированную заработную плату каждый месяц, а отдельно, за каждый заказ, в зависимости от его содержания, то эти приложения не позволяют рассчитывать заработную плату сотрудникам.

В качестве рассматриваемых систем бронирования выступили следующие:

- Системы бронирования АЙТЕК
- DURANTE.Space Booking
- Система бронирования LWCOM
- Simple Office

Таблица 1. Рассмотрение системы бронирования АЙТЕК

Система бронирования АЙТЕК	
Плюсы	Минусы
<ul style="list-style-type: none"> • Подбор помещений с учетом количества сотрудников • Возможность отмены брони • Возможность учета расписания специалистов • Интеграция с почтой для клиентов и специалистов • Сбор статистики использования помещений 	<ul style="list-style-type: none"> • Для достижения полной функциональности и раскрытия всех преимуществ системы, необходимо установка и интеграция аппаратных решений в офисе • Система учитывает расписание только в пределах одной локации и не рассчитывает на время в пути между локациями

Таблица 2. Рассмотрение DURANTE.Space Booking

DURANTE.Space Booking	
Плюсы	Минусы

<ul style="list-style-type: none"> • Кроссплатформенность • Интеграция с сервисами Google Workspace и Microsoft Outlook и другие сервисы • Возможность выбора комнат по оборудованию, смотря от запросов клиента • Сбор статистики использования помещений 	<ul style="list-style-type: none"> • Для достижения полной функциональности и раскрытия всех преимуществ системы, необходимо установка и интеграция аппаратных решений в офисе • Система не учитывает расписание сотрудников и не поддерживает наличия разных локаций и работает в пределах одного офиса
--	--

Таблица 3. Рассмотрение системы бронирования LWCOM

Система бронирования LWCOM	
Плюсы	Минусы
<ul style="list-style-type: none"> • Простой и понятный для пользователя интерфейс 	<ul style="list-style-type: none"> • Для достижения полной функциональности и раскрытия всех преимуществ системы, необходимо установка и интеграция аппаратных решений в офисе • Система не учитывает расписание сотрудников и не поддерживает наличия разных локаций и работает в пределах одного офиса

Таблица 4. Рассмотрение Simple Office

Simple Office	
Плюсы	Минусы
<ul style="list-style-type: none"> • Является только программным решением и не требует дополнительных затрат компании на установку аппаратного оборудования 	<ul style="list-style-type: none"> • Система может работать, только с учетом близкого расположения комнат и локаций, так как не учитывает логистику и затраты времени на переход из одной локации в другую

<ul style="list-style-type: none"> • Широкий функционал: учет расписания сотрудников и время их пребывания на работе • Сбор информации и ее анализ, для улучшения бизнес-процессов • Анализ загруженности локаций • Наличия карты офиса, для клиентов, которые плохо ориентируются 	
--	--

1.3 Изучение алгоритмов распределенной синхронизации

В данном разделе рассматривается сфера применения различных алгоритмов распределённой синхронизации. Также выделяются их главные преимущества и недостатки, для решения поставленной задачи. Следует отметить, что на основе анализа, проведенного в данном разделе, потом будет определено, какой из алгоритмов распределённой синхронизации следует использовать.

Алгоритмы распределенной синхронизации необходимы для разделения различных ресурсов в распределенных системах или синхронизации узлов друг с другом. Примером распределенной сети может стать сеть из компьютеров и/или различных устройств. Этим компьютерам необходимо делить между собой такие ресурсы, как каналы связи, по которым передаются сообщения.

Для решения проблемы, по распределению различных ресурсов можно использовать следующим алгоритмы:

1. Алгоритмы с выбором координатора – для распределения ресурсов будет использоваться отдельный процесс/узел, выполняющий функцию распределения ресурсов между другими узлами. Выбор координатора происходит по голосованию.

1.1. Централизованный алгоритм – данный алгоритм является алгоритмом с выбором координатора. В нем также необходим узел-координатор, но его основная роль будет в принятии запросов от других узлов на выделение ресурсов другим узлам системы и определения, выделить запрашиваемый ресурс узлу или нет. Данный алгоритм может использоваться для решения данной задачи.

2. Круговой алгоритм – для работы данного алгоритма необходима перестройки сети в кольцо (логическое или физическое). Его принцип в передаче права на резервирование ресурсов по кольцу следующему узлу. Данный алгоритм не подходит для выполнения

поставленной задачи, так как имеет крайне низкую производительность: каждому клиенту придется стоять в очереди, пока **один** из клиентов выбирает помещение и время.

2.1. Круговой алгоритм с маркером – данный алгоритм похож на круговой алгоритм, но отличается от него тем, что в данном алгоритме по кольцу ходит маркер, который резервируют узлы, для доступа к ресурсам. В случае, если на данный момент узел не резервирует ресурсы. Этот алгоритм имеет более высокую производительность, нежели «круговой алгоритм», но его реализация крайне тяжела, так как количество пользователей в системе может меняться. Из-за этого придется постоянно перестраивать кольцо, на что будет уходить большое количество ресурсов и времени.

3. Алгоритм древовидный маркерный – этот алгоритм работает похожим образом с круговым алгоритмом с маркером, только в данном методе в качестве топологии сети используется бинарное дерево и по нему ходит маркер с учетом ее топологии. Данный алгоритм лучше работает по сравнению с круговым алгоритмом с маркером, но реализация данной топологии будет крайне затратной.

4. Децентрализованный алгоритм на основе временных меток – алгоритм заключается в том, что все узлы сами определяют могут ли они получить доступ к ресурсам или нет. Если узел получает запрос на доступ к критической сети, которую он хочет зарезервировать, то он сравнивает временные метки, у кого время меньше, тот и занимает критическую секцию.

1.4 Вывод

В данном разделе было рассмотрена предметная область, была поставлена задача и определены способы ее достижения, при помощи алгоритмов распределенной синхронизации, которые также были рассмотрены в данном разделе. Также был проведен анализ похожих продуктов на рынке. Это позволило выделить основные преимущества и недостатки рассмотренных систем, а также увидеть огромный спектр возможностей, которые они предлагают. Информация о данных продуктах позволяет выделить основные требования и понять, как должен выглядеть качественный продукт, а также лучше понять предметную область.

1.5 Цели и задачи НИР

Целью работы является разработка клиент-серверного приложения, которое позволяет клиентам бронировать помещения, а также время клиентов. В данной работе приводится решение следующих задач:

- Анализ рынка похожих решений
- Анализ и сравнение основных алгоритмов распределенной синхронизации

- Определение требований к разрабатываемому продукту
- Проектирование системы
- Разработка и тестирование системы

2 Определение основных требований к разрабатываемому продукту

В данном разделе уточняется предметная область, а также выбирается формируются основные требования к алгоритму синхронизации. На основе данных требований к алгоритму синхронизации выбираются наиболее подходящие алгоритмы распределенной синхронизации. Потом каждый из них рассматривается по-отдельности, что позволяет понять, какие проблемы или изменения необходимо внести в предметную область для реализации определенного алгоритма распределенной синхронизации. После его выбора, более подробно рассматривается предметная область, а именно строится ER-модели, которые отображают различные части предметной области и связи между ними. Стоит отметить, что еще выделяется классы пользователей и их взаимодействие с предметной областью.

2.1 Выделение основных требований к алгоритму распределенной синхронизации и выбор алгоритма распределенной синхронизации

В данном подразделе сформулированы основные требования к алгоритмы распределенной синхронизации. На основе этих требования выделяются наиболее подходящие под них алгоритмы распределенной синхронизации. В данном случае были выбраны централизованный алгоритм и децентрализованный алгоритм на основе временных меток. Будут рассмотрены возможные проблемы с их реализацией.

Главными требованиями к алгоритму распределенной синхронизации будет его производительность и корректность работы. Необходимо выбрать такой алгоритм, который позволит обслуживать как можно большее количество пользователей одновременно, при этом, который корректно распределил бы общие ресурсы, при этом, чтобы не возникало ситуации, когда несколько разных пользователей могли записаться в одну комнату одновременно или могли бы забронировать специалиста в один и тот же промежуток времени. Также, желательно, чтобы алгоритм был прост в его реализации и не накладывал ограничения на разработку ПО, ведь в случае его простоты будет меньший шанс допустить при его разработке ошибки.

По итогу рассмотрения основных алгоритмов распределенной синхронизации в [разделе 1.3](#), было выделено 2 алгоритма, как потенциальное решение задачи. А именно «Централизованный алгоритм» и «Децентрализованный алгоритм на основе временных меток». Именно данные алгоритмы позволяют получить наибольшую производительность без

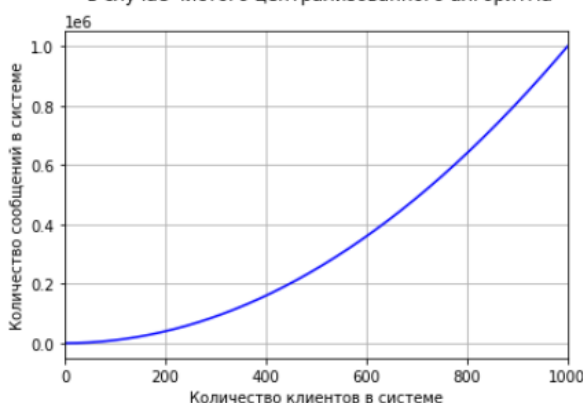
больших затрат вычислительных ресурсов, по сравнению с алгоритмами, где придется моделировать различные топологии и использовать маркер, который также влияет на производительность.

Рассмотрим поподробнее выбранные 2 алгоритма:

- Использование децентрализованного алгоритма на основе временных меток может работать некорректно из-за того, что у разных узлов будет различное время, так как их времена будут не синхронизированы, а как следствие сравнение этих времен не может давать корректный результат. Откуда следует, что необходимо проводить синхронизацию времени на всех клиентских устройствах, которые будут посылать сигналы. Также стоит отметить, что данный алгоритм для применения в нашей задаче необходимо несколько изменить: Сообщения о бронировании ресурсов будет отправляться не всем узлам системы, а лишь только узлам сервиса А и сервиса Б. Ведь в случае, если сообщения о намерении занять ресурс будет отправляться всем узлам, то количество сообщений о запросе на бронирование ресурсов сильно растет (представлено на рис. 2). Далее встает проблема нагрузки на сервисы А и Б, ведь клиент может отправлять некорректные запросы на выделения ресурсов, для исправления данной проблемы необходимо максимально проверить запрос на корректность перед его отправкой. Отсюда вытекает еще 1 проблема: нагрузка клиентского устройства для предотвращения возможности отправки некорректных запросов и их проверку. Так как характеристики клиентских устройств неизвестны, то данной ситуации необходимо избегать. Избежать данную ситуацию поможет использование централизованного алгоритма, где будет отдельный узел, рассчитанный на данную нагрузку, который будет не только проверять корректность, но и определять возможность выделения ресурсов определенному узлу.
- Для использования централизованного алгоритма, необходимо определить по какому принципу централизованный алгоритм будет определять какому из узлов предоставлять ресурс. В случае если на один ресурс претендует один узел, то все тривиально. В случае, когда их 2 и более, то для определения какому устройству отдать ресурс будет использоваться очередность прихода запросов, а также концепция транзакций. Каждый запрос выполняется по мере своего прибытия в центральный узел, при условии, что запрос, который пришел позже, не может начать выполняться раньше, чем тот, который пришел за некоторое время до данного. Транзакция, в свою очередь бронирует определенные ресурсы

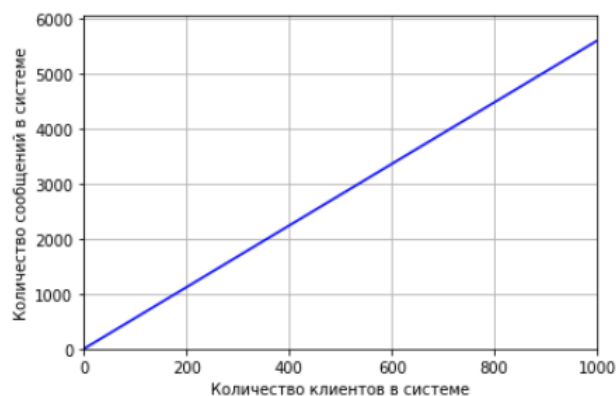
на время исполнения алгоритма. За определение того, какому запросу будет удовлетворено в выделение своего типа ресурса (специалист и помещение) будут отвечать сервисы А и Б соответственно, так как в рассмотрении предметной области были сказано, что данные сервисы должны быть независимыми и уже работоспособными для выделения своего ресурса пользователям. Сервисы будут посылать результаты в узел-координатор, который на основе результатов, будет принимать финальное решение, по поводу того, выделять ресурсы под запрос или нет, в случае успеха транзакции для выделения ресурсов закрываются, и пользователь уведомляется об успехе, в противном случае транзакции откатываются, и пользователь уведомляется об отказе. Этот вариант является оптимальным, так как не требует написания сложного программного обеспечения, для достижения поставленной цели, а также позволяет выполнить концепцию того, что сервис А и Б являются независимыми друг от друга и полностью работоспособными сервисами.

График зависимости кол-во сообщений в системе от клиентов в случае чистого централизованного алгоритма



а)

График зависимости кол-во сообщений в системе от клиентов в случае чистого децентрализованного алгоритма



б)

Рис. 2. График зависимости кол-во сообщений в системе от клиентов в случае
а) чистого децентрализованного алгоритма б) централизованного алгоритма.

Для графика чисто децентрализованного алгоритма использовалась следующая формула:

X = кол-во клиентов, $Y = (\sum_X x - 1) + x = x(x - 1) + x$, где под $\sum_X x$ подразумевается рассылка запроса о бронировании клиентом другим клиентам, с запросом на занятие данных ресурсов.

Для графика централизованного алгоритма использовалась следующая формула: X = кол-во клиентов. Для выделения ресурса клиенту, необходимо отправить запрос в узел-координатор (1), затем узел-координатор проверяет запрос на корректность. Затем запрос

отправляется в сервис А (2) (для записи к специалисту) и в сервис Б (3) (для бронирования помещения). Сервисы выделяют (или нет) ресурсы на данный запрос и отправляют ответ в узел-координатор (4,5). Узел-координатор сравнивает ответы, в случае успеха транзакция закрывается, иначе откатывается. Об результате сообщается пользователю (6).

И того получаем, что если **каждый из пользователей пытается записаться** к специалисту в определенное помещение, то для выполнения этого действия необходимо 6 запросов. Тогда получим следующую формулу для количество сообщений от кол-во пользователе в системе: $Y = 6x$

Предположим, что 10% запросов будут некорректны и отсеяны на этапе проверки запросов, то есть на них будет потрачено всего 2 запрос от клиента к узлу координатору, а также ответ от узла-координатора к клиенту : $X = \text{кол-во клиентов}$, $Y = 6 * X * 0.9 + 2 * X * 0.1$.

Далее мною было проведено моделирование очереди на запись для определения кол-во сообщений в различных ситуациях: имеется 1000 клиентов и t записей, которые клиенты могут занять. Допустим, что с вероятностью 10% запись, созданная пользователем некорректна. Затем проверим, что пользователь записался на запись, которую на момент записи не занял другой клиент. В случае если выбранная запись клиента занята, то тогда он с вероятностью в 2/3 попытается записаться снова. Итого получили следующие результаты по кол-во запросов в системе.

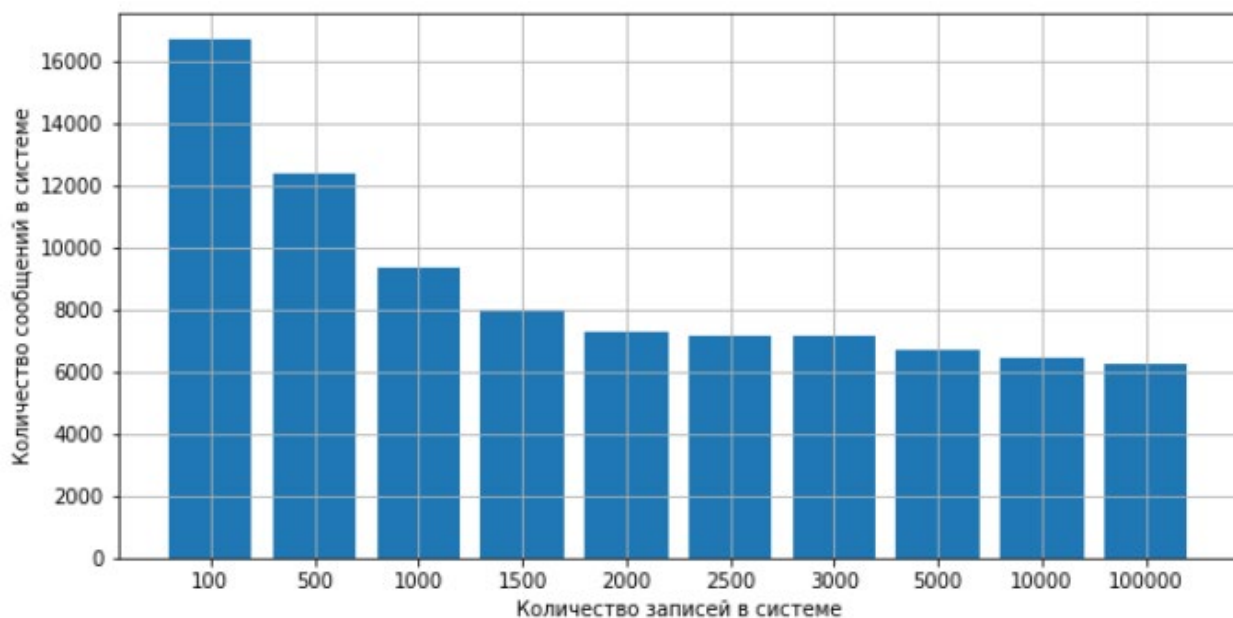


Рис. 3 Моделирование количество запросов в системе при фиксированном кол-ве пользователей (1000) и ограниченном количестве записей.

По итогу рассмотрения и выбора из данных алгоритмов, было решено использовать «Централизованный алгоритм», так как он меньше нагружает клиентское устройство, а также он имеет меньшую нагрузку на сеть, по сравнению с децентрализованным алгоритмом.

2.2 Проектирование ER-моделей предметной области

В данном подразделе рассматривается предметная область в виде ER-моделей. Это позволит более конкретно определить связи и зависимости между ее составляющими, выделить классы пользователей, а также их взаимодействие с другими составляющими предметной области. Также будет представлена ER-диаграмма, которая показывает все возможные воздействия и результаты разрабатываемом продукте.

Так как централизованный алгоритм предполагает использование узла-координатора, который будет позволять бронировать помещения и записываться к специалистам, то наше отображение предметной области необходимо преобразовать под использование централизованного алгоритма.

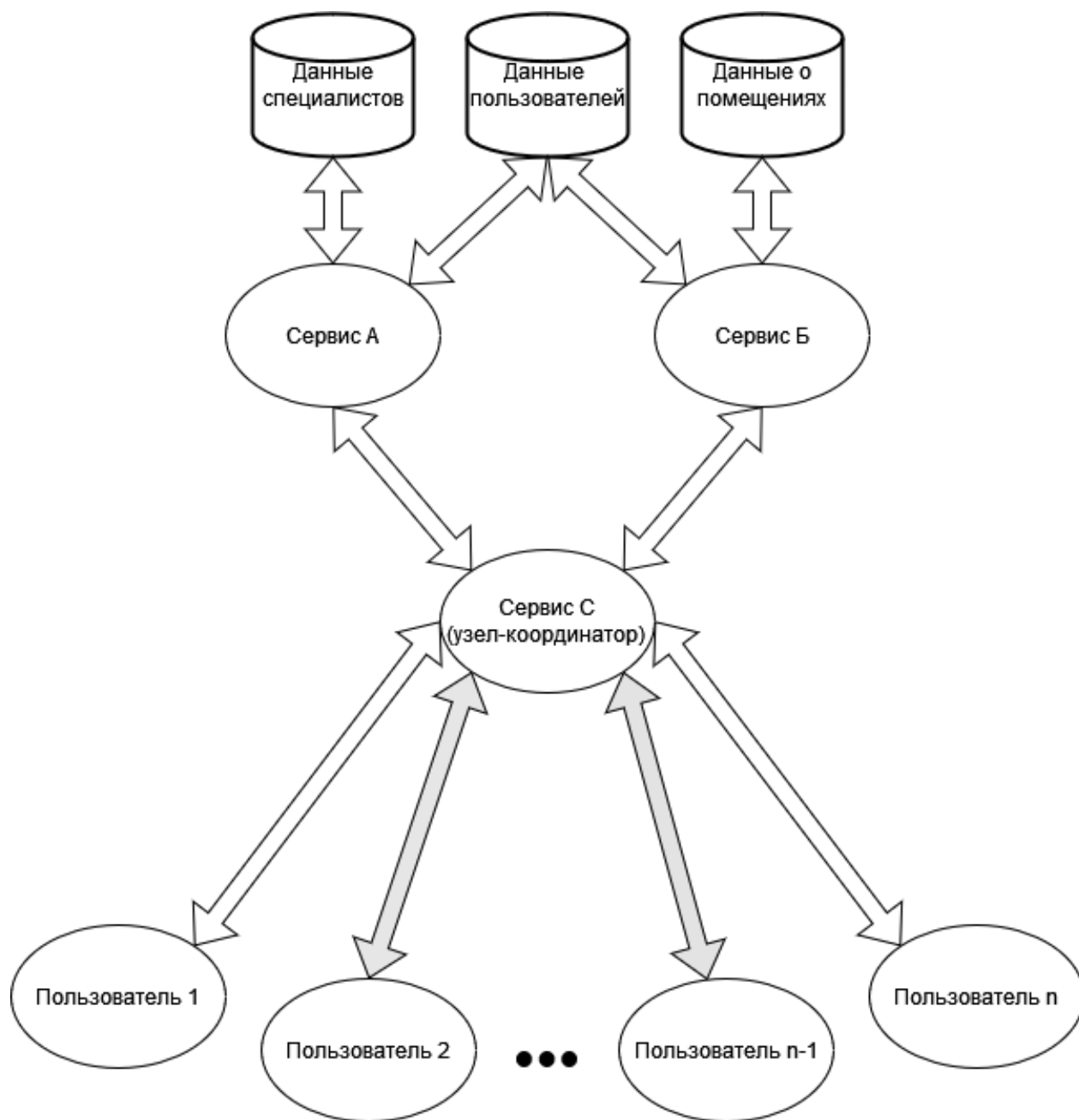


Рис. 4. Представление предметной области с использованием централизованного алгоритма распределенной синхронизации

На рис. 4 можно наблюдать, что сервис А и сервис Б имеют собственные данные, в которых будет храниться необходимая им информация. Также сервисы имеют общую информацию о пользователях. Стоит отметить, что данные сервисы планируется сделать самодостаточными, то есть через сервис А можно записаться к специалисту (без необходимости помещения), а через сервис Б можно бронировать помещение (без необходимости специалиста). Сервис С выступает в качестве связующего, который отправляет запросы к сервисам А и Б, а также получает от них ответы, проверяя возможность одновременного бронирования помещения и записи к специалисту. В свою очередь, несколько клиентов могут одновременно пытаться записываться. Их заявки будут приходить в сервис С, где они будут выстраиваться в очередь, на основе которой будут посылаться запросы в сервисы А и Б, на проверку **возможности** записи и бронирования, сервисы А и Б посылают ответы на запросы в сервис С, который уже отправляет результат об успехе/неудаче пользователю. В случае если данная возможность присутствует, то запись осуществляется и об успехе уведомляется клиент. В противном случае, клиент не записывается, а информация об этом поступает к клиенту, предлагая ему снова выбрать специалиста, а также помещение.

Также необходимо отметить, что с данной системой, могут работать, не только клиенты, но и специалисты, для изменения своего расписания и получения информации о том, где, когда и какому клиенту они будут предоставлять свои услуги. Все основные взаимодействия пользователей (в число которых входят клиенты и специалисты) с системой представлены на рис. 5.

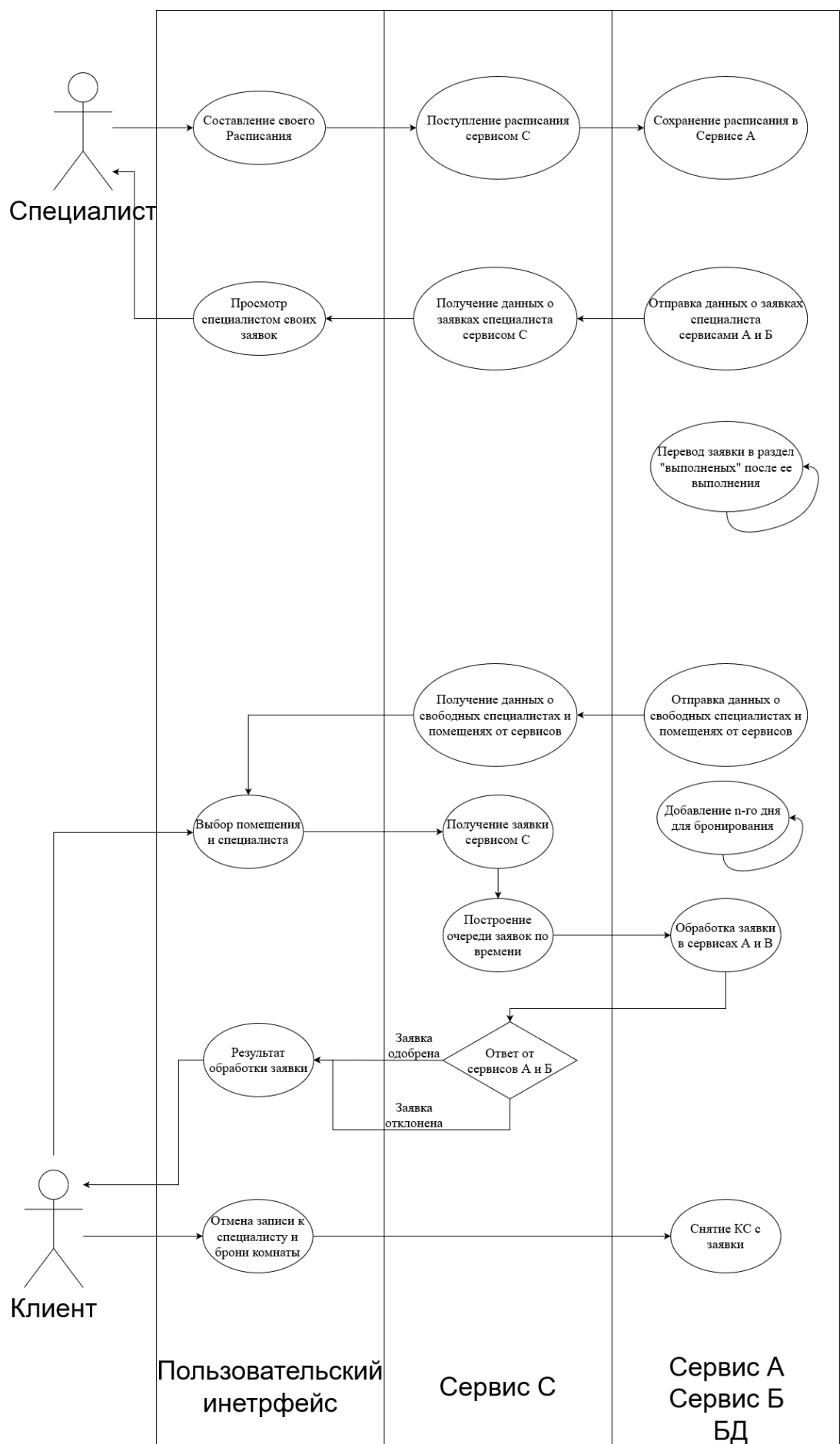


Рис. 5. Взаимодействия сервисов системы с классами пользователей и друг с другом.

На данном рисунке представлены 3 слоя, на которых определяются некоторые из основных функций, которые они должны выполнять по средствам запросов и связей с другими слоями.

2.2.1 Слой «Пользовательский интерфейс»

Данный слой описывает основные функции, которые должны быть представлены пользователям (специалистам и клиентам). Этот слой из себя представляет интерфейс, через который пользователи могут выполнять определенные задачи:

- a. Клиент должен иметь возможность выбирать комнату и специалиста, а также иметь возможность записи на выбранную пару специалист-комната
- b. Клиент должен иметь возможность отменять свои записи, прием которых еще не наступил
- c. Клиент должен иметь возможность просматривать все свои записи
- d. Неавторизованный пользователь должен иметь возможность для регистрации или авторизации
- e. Клиент должен иметь возможность оценить специалиста, по завершению приема
- f. Клиент должен иметь возможность оценить комнату, в которой проходил прием
- g. Специалист должен иметь возможность просмотреть все записи, которые он имеет, а также информацию, где эти записи будут проходить
- h. Специалист должен иметь возможность отменять все записи к нему в случае заболевания или других не предвиденных обстоятельств

2.2.2 Слой «Сервис С»

Данный слой отвечает за прием запросов от пользователей, их обработку и отправку обработанных и приведенных к необходимому формату запросов пользователей.

2.2.3 Слой «Сервис А, Сервис Б и БД»

Данный слой отвечает за прием запросов от сервиса С, их обработки и выдача данных/ответа на данные запросы. Также средствами СУБД необходимо реализовать следующие функции:

- a. Перевод заявок в раздел «**выполненных**» после их выполнения
- b. Добавления возможности записи к специалистам и бронирования комнаты при наступлении следующего дня

2.3 Вывод

В данном разделе был выбран централизованный алгоритм распределенной синхронизации. Модель системы была модифицирована с учетом данного алгоритма. Также было рассмотрено классы пользователей и каким образом они взаимодействуют с системой.

Сама системы была представлена в виде 3 уровней, для каждого из которого были определены его основные функции.

3 Проектирование программного продукта

После более подробного рассмотрения предметной области и выделения ее различных составляющих, уточняются свойства каждого компонента системы, на основе которых выбираются инструменты и методы для создания системы. Среди всех инструментов, подходящих для данной задачи, выбирается те, которые является наиболее удобными и позволяют реализации весь функционал продукта, а также его составляющих частей. На основе выбранных инструментов представляется архитектура системы, а также определяется, каким образом и как будет выполнена каждый компонент архитектуры.

3.1 Разработка требований к разрабатываемому программному продукту и выбор инструментов разработки

В данном разделе определяются основные свойства и устройство компонентов системы. Также в данном разделе уделяется внимание БД, и тем сущностям, которые она будет хранить, а также их свойствам. Затем на основе главных свойств и требований к системе и ее частям приводятся наборы из инструментов, которые позволят реализовать поставленную задачу. Среди данных наборов выбирается наиболее подходящий.

Ранее были определены следующие компоненты системы:

- Сервис А – для записи к специалистам
- Сервис Б – для бронирования помещений
- Сервис С – в качестве центрального узла регулятора, который будет на основе данных из сервисов А и Б решать, записать пользователя (на пару специалист – помещение) или нет
- Пользовательский интерфейс – интерфейс, доступный пользователю для отправки запросов в сервис С
- БД – база данных, которая будет управляться СУБД и хранить всю необходимую для сервисов А и Б информацию

Основные требования к [сервисам А и Б](#), [сервису С](#), а также к [пользовательскому интерфейсу](#) были представлены в разделе 2.2.

Рассмотрим структуру базы данных, в которой будет храниться информация о пользователях, а также информация, необходимая для работоспособности сервисов А и сервиса Б. На основе данного рассмотрения, мы сумеем выделить необходимую сущности, их свойства и связи между ними, благодаря чему в последствии сумеем построить модель БД и в дальнейшем ее разработать.

Введем письменное описание БД.

Данная база данных должна хранить информацию о пользователях, в число которых входят специалист и клиент. Хранить необходимую о них информацию, такую как:

- Фамилия
- Имя
- Отчество
- Номер телефона для связи
- Электронная почта для уведомлений о записи или ее отмене

Со стороны сервиса А имеются несколько сущностей, которые требуют более детального рассмотрения и определения, какую информацию о них необходимо хранить. Данными сущностями являются: **специалист**, который наследуется от пользователя, **записи** к специалистам от пользователей, по завершению которых пользователь может поставить оценку, а также расписание, которое имеют специалисты.

- **Расписание** будет представлено в виде одной недели, где специалист сумеет выбрать в какие дни недели сколько он будет работать.
- **Запись к специалисту**, будет представлять из себя таблицу, которая хранит в себе информацию о записи пользователя к специалисту. В нее будет входить идентификатор пользователя, идентификатор специалиста, время начала записи, время ее завершения, а также оценка от пользователя данному специалисту.
- **Специалист** будет хранить в себе такую информацию, как его специальность, степень, стаж, а также краткое описание этого специалиста.

Со стороны сервиса Б, будет храниться информация об **области (здании)**, а также о **комнатах**, которые принадлежат областям (зданиям).

Область (здание) будет хранить информацию об местоположении области, а именно об имени области, ее адресе и времени, когда она доступна.

Комнаты, будут ссылаться на область, к которой они принадлежат. Также комнаты будут хранить информацию о своем номере и другую поясняющую для них информацию.

В разделе 3.2 на рис. 6. представлена ER-диаграмма базы данных, основные сущности и требования которой были описаны выше.

Перейдем к **выбору средств разработки** для создания сервисов, а также клиентского приложения.

Основным свойством сервисов должно являться их доступность и адаптивность под различные задачи, так как в данной работе сервис по записи к специалисту и бронирование помещений является всего лишь примером.

1. Для этих целей может подойти вариант реализации вышеперечисленных сервисов при помощи RESTful web-сервисов (на основе web-API). К web-сервисам будут общаться и работать при помощи HTTP запросов. Главным преимуществом web-сервисов является его гибкость в работоспособности с различными системами и приложениями, так как он может передавать данные в различных форматах (в основном **JSON**, XML, HTML) и для доступа, к которому нужен всего лишь доступ в сети к данному web-API. Каждый из web-сервисов будет выполнять обязанности определенного сервиса (А, Б). В свою очередь, web-приложение будет реализовывать функционал сервиса С, а для получения всей необходимой информации будет считывать данные, введенные пользователями, а также отправлять запросы в созданные web-API для сервисов А и Б, получая от них всю необходимую информацию для дальнейшей работы. В качестве пользовательского интерфейса могут выступать различные браузеры, которые будут отрисовывать страницу сайта, которая была получена от web-приложения. Преимуществом данного подхода является возможность работы на обширном количестве устройств, начиная с мобильных телефонов и заканчивая ПК.
2. Вторым вариантом является использование платформы WCF, которая используется для создания приложений, ориентированных на службы, и выполняет обмен данных на основе протокола SOAP. Ее преимуществом заключается в ее безопасности, устойчивости. Данная платформа также имеет ряд недостатков, такие как ограничение на то, в каком формате могут передаваться данные (только XML, что вносит свои ограничения). В качестве сервисов А, Б и С будут выступать серверные приложения, а в качестве пользовательского интерфейса будет выступать пользовательское приложение, которое будет обмениваться информацией с сервером С. Данный вариант не имеет пользуетесь большим успехом в различных решениях, в следствии ограничений, которые вводятся использованием протокола SOAP.

По итогу рассмотрения представленных двух вариантов было решено использовать фреймворк **ASP.NET Core 7.0**, для разработки web-приложения и web-API. Для создания которых будет использоваться язык **C#**. Для работы доступа и работы с СУБД будет использоваться **Entity Framework Core**, который позволяет работать с различными СУБД,

что является крайне гибким вариантом. В качестве СУБД будет выступать **SQL Server 2019 Express**, функционал которого будет расширен при помощи инструментов **Windows**.

3.2 Представление моделей разрабатываемого программного продукта

На основе требований к разрабатываемому продукту, выведенных в прошлом пункте, в данном подразделе представляется архитектура системы и структура базы данных. Архитектура системы показывает основные компоненты, из которых будет состоять система, а также их взаимодействия друг с другом.

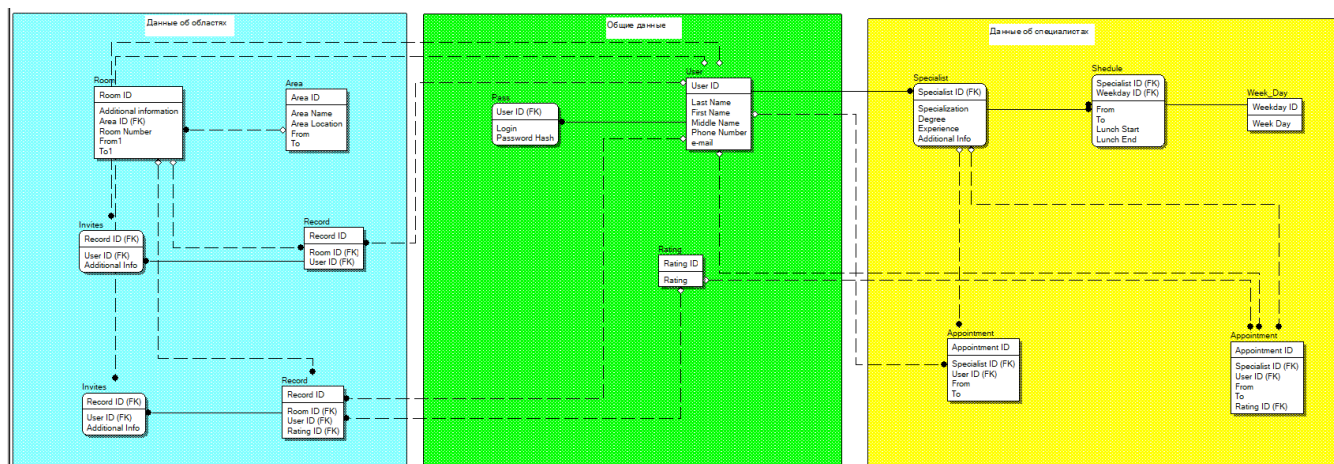


Рис. 6. ER-диаграмма базы данных

Данная диаграмма отображает три различных областей, которые разделены между 2 сервисами. По середине: «Общие данные», зеленого цвета, содержит и хранит информацию о пользователях, их паролях, а также общую шкалу оценок для 2 сервисов. Вся информация из «общего раздела» доступна обоим сервисам. Голубая область слева, под названием «Данные об областях»

Рассмотрим архитектуру разрабатываемой системы. Она представлена на рис. 7.

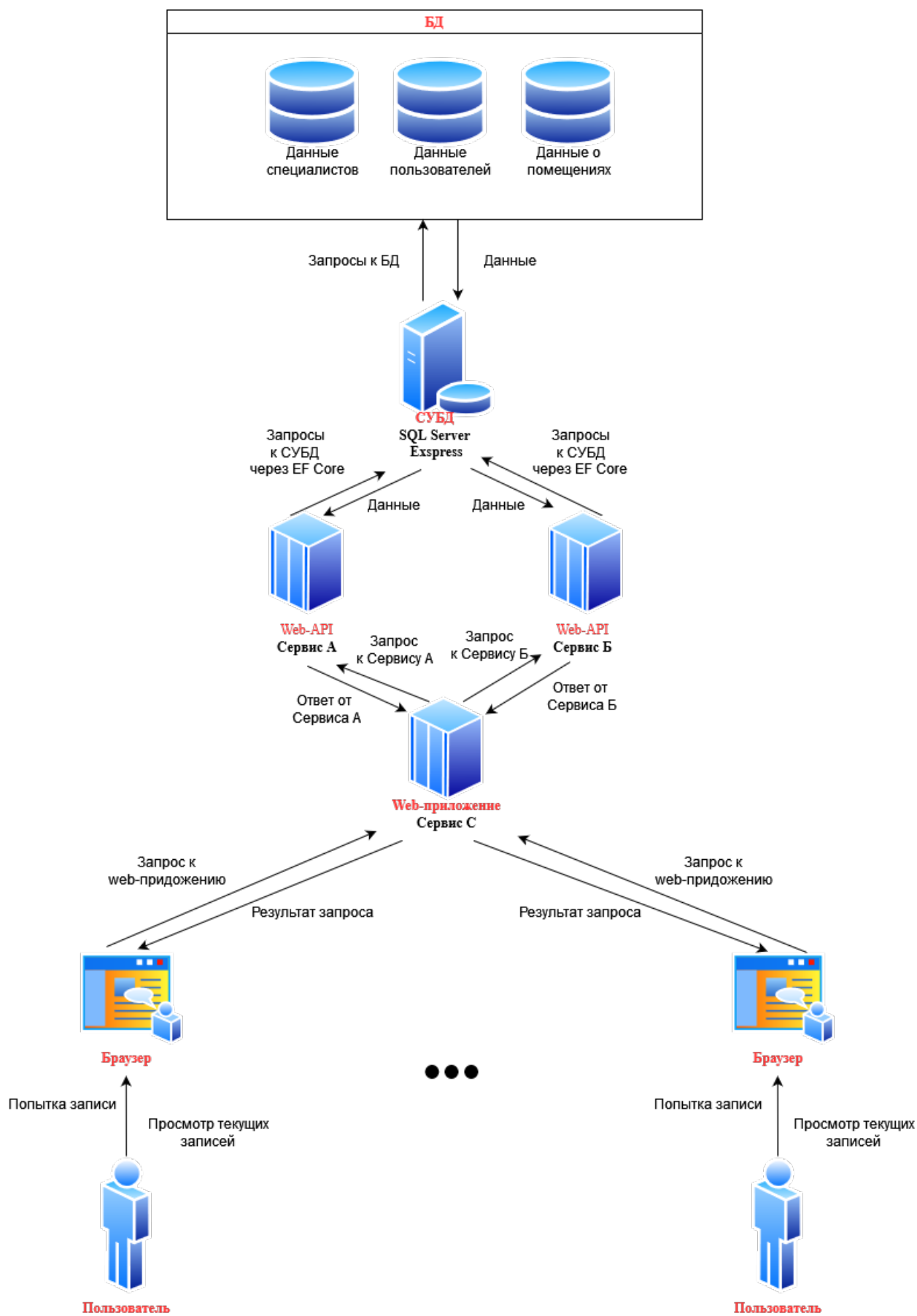


Рис. 7. Архитектура разрабатываемой системы

На рис. 7 можно увидеть полную архитектуру разрабатываемого приложения. Рассмотрим компоненты представленной архитектуры и опишем какие из выбранных инструментов разработки будут использоваться для ее реализации.

СУБД: в качестве СУБД будет использоваться SQL Server Developer 2022

Web-API: для разработки будет использоваться фреймворк ASP.NET Core, под платформой .NET 7.0. В качестве среды разработки будет использоваться Visual Studio 2022. В качестве шаблона будет использоваться «Веб-API ASP.NET Core (Майкрософт)».

Web-приложение: для разработки будет использоваться фреймворк ASP.NET Core, под платформой .NET 7.0. В качестве среды разработки будет использоваться Visual Studio 2022. В качестве шаблона будет использоваться «Веб-приложение ASP.NET Core (Майкрософт)».

Браузер: В качестве браузера – может использоваться любой из актуальных на данный момент браузеров или более ранние версии данных браузеров. Firefox версии 107.0.1, Google Chrome версии 108, Яндекс.Браузер версии 22.1.0.2510.

Обмен информацией между компонентами архитектуры будет осуществляться благодаря HTTP-запросам.

3.3 Вывод

В данном разделе было определен набор инструментов для разработки системы, в соответствии с которыми была определена архитектура системы и основные требования к этим компонентам. Были определены сущности, их свойства и связи между ними в БД, которая необходима для разрабатываемой системы.

4 Разработка и тестирование системы

В данном разделе производится разработка самого программного продукта на основе созданных нами требований и проверка работоспособности как каждого блока, так и целой системы в целом. После создания программного продукта следует его характеристика, которая позволит лучше понять работоспособность созданного продукта. Также в данном разделе производится разработка пользовательского интерфейса, после чего приводятся различные примеры взаимодействия с интерфейсом и рассмотрение различных ситуаций и примеров. Стоит отметить, что в данном разделе производится поиск и обработка исключительных ситуаций, которые могут возникнуть в процессе работы. Еще по итогу разработки выделяются особенности разработанного программного обеспечения и его сравнения с аналогами на рынке. Выделяются преимущества данного продукта, а также определяется дальнейшие пути его развития.

4.1 Разработка программного продукта и его характеристика, а также разработка пользовательского интерфейса

В данном подразделе показывается процесс разработки основных частей программного продукта. Выделяются различные его составляющие, которые затем реализуются. Также к ним приводится краткая характеристика, описывающая за какой функционал отвечает данная часть программы. По итогу разработки всех составляющих происходит сравнение с аналогами на рынке, выделяются преимущества разработанного продукта.

4.1.1 Разработка БД

База данных была написана на основе сущностей, введённых в [разделе 3.1](#).

Весь код скрипта для создания БД приведен в [Приложении 1](#).

По итогу выполнения данного скрипта в SMSS (SQL Server Management Studio) была создана БД, на основе которой была создана диаграмма БД, указанная на рис. 8.

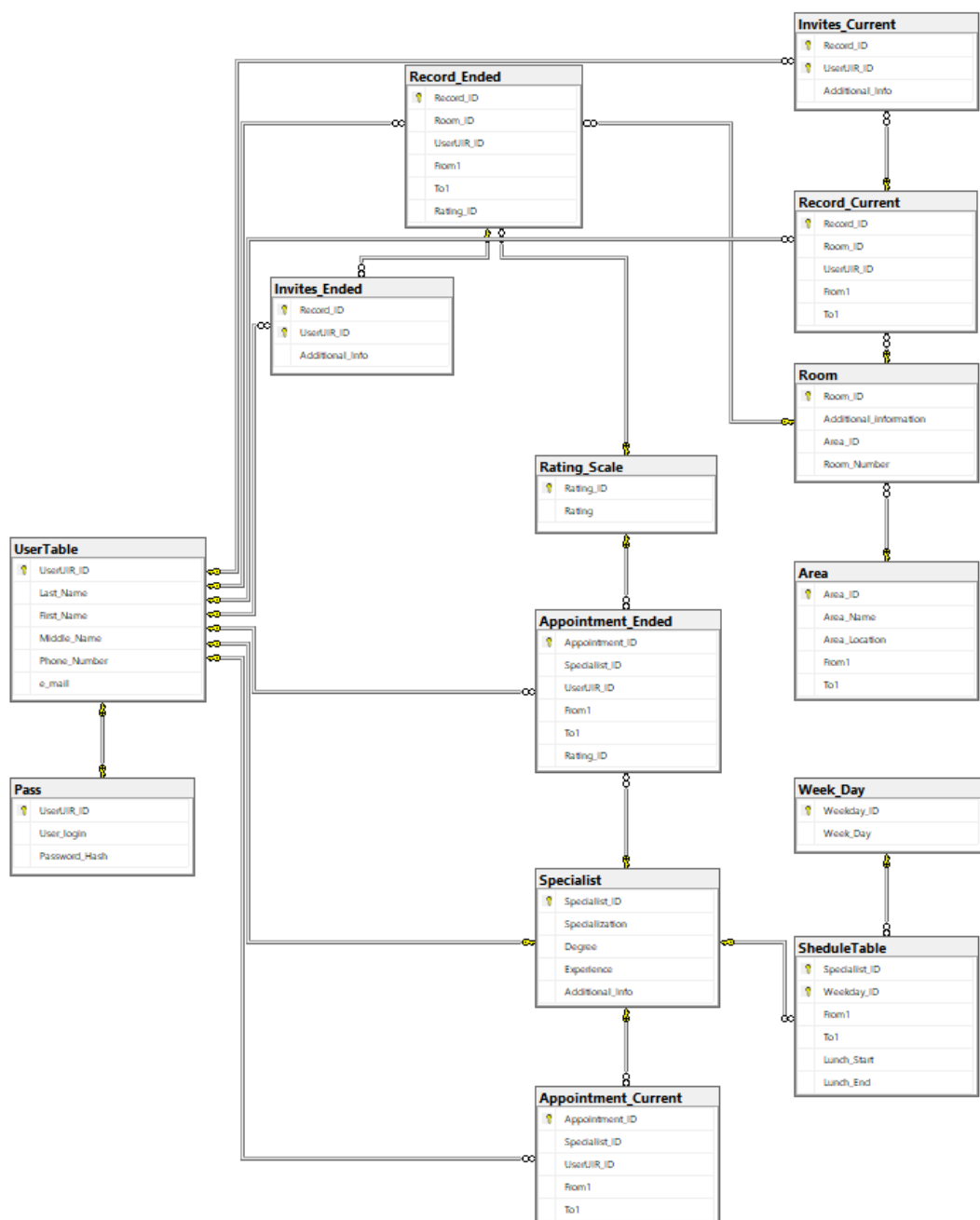


Рис. 8. Диаграмма БД

Также в данную базу данных были добавлены события, которые отрабатывают каждые 5 минут и переводят уже прошедшие и закончившиеся записи из таблиц Record_Current||Appointment_Current||Invites_Current в таблицы Record_Ended||Appointment_Ended ||Invites_Ended соответственно.

4.1.2 Разработка веб-API для сервиса А

При разработке данного компонента использовался шаблон «Веб-API ASP.NET Core (Майкрософт)».

По умолчанию он включает файл launchSettings.json, который содержит необходимую информацию для запуска веб-API, файл appsettings.json, в который была добавлена строка

подключения к созданной БД, а также файл Program.cs, которая конфигурирует и запускает веб-API.

К проекту были подключены следующие пакеты NuGet:

- Microsoft.AspNetCore.Diagnostic.EntityFrameworkCore – необходим для включения возможности получения вложенных данных при помощи EF Core
- Microsoft.AspNetCore.Mvc.NewtonsoftJson – необходим для включения возможности получения вложенных данных при помощи EF Core
- Microsoft.AspNetCore.OpenApi
- Microsoft.EntityFrameworkCore.Relational – необходим для генерации моделей БД при использовании подхода DataBaseFirst
- Microsoft.EntityFrameworkCore.SqlServer – необходим для генерации моделей БД при использовании подхода DataBaseFirst
- Microsoft.EntityFrameworkCore.Tools – необходим для генерации моделей БД при использовании подхода DataBaseFirst
- Microsoft.VisualStudio.Web.CodeGeneration.Design – необходим для генерации шаблонов контроллера
- Swashbuckle.AspNetCore – нужен для работы Swagger

Применение подхода DatabaseFirst

После подключения данных пакетов и написания необходимых строчек кода воспользуемся инструментом **Scaffold-DbContext**, который позволит сгенерировать все необходимые модели на основе существующей БД. Для этого в консоль диспетчера пакетов Visual Studio команду указанную на Рис. 9.

```
PM> Scaffold-DbContext "Data Source=DESKTOP-6V02F50\SQLEXPRESS;Initial
Catalog=UIR_DB;Integrated Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnet
Failover=False" Microsoft.EntityFrameworkCore.SqlServer -force -OutputDir Models
Build started...
Build succeeded.
```

Рис. 9. Команда для создания моделей на основе БД

По выполнению этой команды в указанную папку (в нашем случае это папка «**Models**»).

Выбрать только те модели, которые необходимы для работоспособности сервиса А, мы получим папку «**Models**» со следующими моделями, которые показаны на Рис. 10.

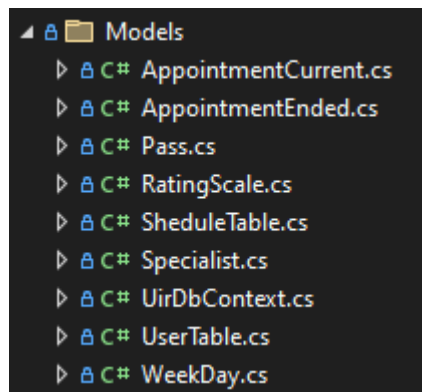


Рис. 10. Модели, которые получились по итогу генерации моделей и контекста БД при использовании подхода DatabaseFirst

Далее необходимо создать контроллеры, которые позволят отправлять запросы в наше веб-API и получать некий результат, работая и обрабатывая входные данные, а также данные от Entity Framework Core.

4.1.2.1 Контроллер UserTablesController

Данный контроллер позволит авторизовываться или проходить регистрацию пользователям, получать список всех пользователей, где это необходимо (к примеру: для приглашения в забронированную комнату), а также изменять данные о пользователе, если он поменял данные своего профиля.

Для регистрации пользователя будет использоваться **POST** запрос с экземпляром модели UserTable, который будет добавлять в БД нового пользователя с включенным в него моделью «Pass» через EF Core. Данный запрос позволяет сразу добавить к пользователю логин и пароль, а также если заполнено поле Specialist, то созданный пользователь становится специалистом.

Для получения всех пользователей будет использоваться GET запрос без входных данных, который будет возвращать лист из пользователей только общедоступную информацию (не логин и пароль). Будет использоваться для вывода списка пользователей, при желании их пригласить в комнату.

Для получения информации о пользователе будет использоваться GET запрос с целочисленной переменной, в качестве входной переменной, которая обозначает уникальный идентификатор пользователя, информацию о котором нам необходима. Данный запрос будет возвращать данные о пользователе. Будет использоваться при переходе на страницу пользователя в системе.

Для получения информации о пользователе будет использоваться PUT запрос с целочисленной переменной (ID пользователя) и экземпляром модели UserTable (обновленные данные пользователя). Данный запрос обновит данные о пользователе. Будет использоваться при изменении данных пользователя в его профиле.

4.1.2.2 Контроллер Passes

В данном контроллере будет реализован лишь один HTTP-запрос, который позволит обновить пользователю его логин и пароль.

Для обновления логина и пароля будет использоваться PUT-запрос. В качестве входных переменных будет использоваться целочисленная переменная (ID пользователя), а также экземпляр Pass, который будет хранить в себе новый логин и пароль пользователя.

В дальнейшем я не буду так подробно описывать все контроллеры, а лишь скажу какие задачи они помогают реализовать и для чего они используются, не вдаваясь в подробности каждого HTTP-запроса.

4.1.2.3 Контроллер SpecialistsController

Данный контроллер позволяет получать информацию о специалистах в системе. При необходимости он может выдать более конкретную информацию о специалисте. Также специалист может менять информацию о самом себе.

4.1.2.4 Контроллер SheduleTablesController

Данный контроллер позволяет получать специалистам свое расписание, добавлять новые дни недели в него, изменять его, а также удалять в случае необходимости (*новое расписание вступит в силу через 7 суток после его добавления/обновления/удаления*)

4.1.2.5 Контроллер AppointmentCurrentsController

Данный контроллер позволяет пользователям создавать и удалять записи к специалистам, а также просматривать все записи пользователя и специалиста. При добавлении записи, проверяется расписание специалиста, все прочие записи данного специалиста на то, чтобы проверить не будет ли пересечений при записи.

4.1.2.6 Контроллер AppointmentEndedsController

Данный контроллер позволяет посмотреть пользователю все записи, которые он посещал. Все посещенные записи можно оценить. Также можно получить среднюю оценку специалиста, по всем записям, которые он посетил.

По итогу разработки данного веб-API мы можем послать множество различных HTTP-запросов, для получения необходимой нам информации. Список всех созданных HTTP-запросов для каждого контроллера приведен на Рис. 11

AppointmentCurrents	
GET	/api/AppointmentCurrents/{id}
DELETE	/api/AppointmentCurrents/{id}
GET	/api/AppointmentCurrents/User/{id}
GET	/api/AppointmentCurrents/Specialist/{id}
DELETE	/api/AppointmentCurrents/Specialist/{id}
POST	/api/AppointmentCurrents
AppointmentEndeds	
GET	/api/AppointmentEndeds/User/{id}
PUT	/api/AppointmentEndeds/User/{id}
GET	/api/AppointmentEndeds/Rating/{id}
Passes	
PUT	/api/Passes/{id}
SheduleTables	
GET	/api/SheduleTables/{id}
PUT	/api/SheduleTables/{id}
DELETE	/api/SheduleTables/{id}
POST	/api/SheduleTables
Specialists	
GET	/api/Specialists
GET	/api/Specialists/{id}
PUT	/api/Specialists/{id}
UserTables	
GET	/api/UserTables
POST	/api/UserTables
GET	/api/UserTables/{id}
PUT	/api/UserTables/{id}

Рис. 11. Список всех HTTP-запросов для веб-API сервиса А.

4.1.2.7 Использование JWT авторизации

Также стоит отметить, что для работы с большинством запросов данного веб-API необходимо проходить авторизацию при помощи JWT (Json Web Token). Это позволит пресечь несанкционированный доступ к данному веб-API со стороны мошенников. Также чтобы в случае получения злоумышленником JWT токена, пресечь его долговременный доступ к системе, будут использоваться JWT токены, совместно с Refresh токенами, которые по истечению короткого промежутка времени, сделают JWT-токен недействительным, а при использовании Refresh-токена, совместно с JWT, пользователь будет получать новый, обновленный JWT-токен, который позволит продолжать работу пользователя без повторного прохождения процесса аутентификации.

4.1.3 Разработка веб-API для сервиса Б

При разработке данного компонента использовался шаблон «Веб-API ASP.NET Core (Майкрософт)».

По умолчанию он включает файл `launchSettings.json`, который содержит необходимую информацию для запуска веб-API, файл `appsettings.json`, в который была добавлена строка подключения к созданной БД, а также файл `Program.cs`, которая конфигурирует и запускает веб-API.

К проекту были подключены следующие пакеты NuGet, такие же как и в веб-API для сервиса А:

- `Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore` – необходим для включения возможности получения вложенных данных при помощи EF Core
- `Microsoft.AspNetCore.Mvc.NewtonsoftJson` – необходим для включения возможности получения вложенных данных при помощи EF Core
- `Microsoft.AspNetCore.OpenApi`
- `Microsoft.EntityFrameworkCore.Relational` – необходим для генерации моделей БД при использовании подхода `DataBaseFirst`
- `Microsoft.EntityFrameworkCore.SqlServer` – необходим для генерации моделей БД при использовании подхода `DataBaseFirst`
- `Microsoft.EntityFrameworkCore.Tools` – необходим для генерации моделей БД при использовании подхода `DataBaseFirst`
- `Microsoft.VisualStudio.Web.CodeGeneration.Design` – необходим для генерации шаблонов контроллера
- `Swashbuckle.AspNetCore` – нужен для работы Swagger

В данном веб-API также применялся подход `DatabaseFirst`

Так как у нас уже имеется созданная база данных, то на ее основе можно сформировать модели для работы с Entity Framework Core для этого воспользуемся инструментом **Scaffold-DbContext**, который позволит сгенерировать все необходимые модели на основе существующей БД. Для этого в консоль диспетчера пакетов Visual Studio команду аналогичную той, что в [разделе 4.1.2](#). После чего из нее нужно удалить все модели, которые не нужны для работы и логики веб-API для сервиса Б. Также подправить связи и класс контекста БД. По итогу получаем, что в папке Models содержатся следующие модели и контекст БД.

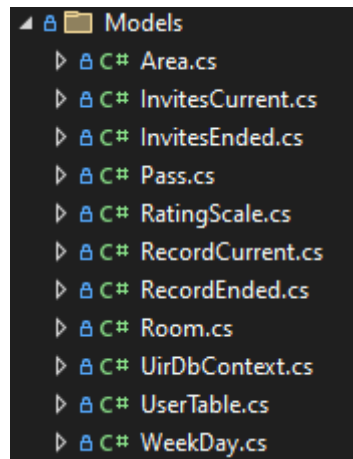


Рис. 12. Модели, которые получились по итогу генерации моделей и контекста БД при использовании подхода DatabaseFirst, а также после выбора только нужных для данного веб-API.

Далее необходимо создать контроллеры, которые позволят отправлять запросы в наше веб-API, обрабатывать входные данные (если они имеются) и после отправлять запрошенные данные, если при обработке не произошло исключений.

Всего планируется 6 контроллеров:

2 для записей, 2 для приглашений, 2 для данных о пользователях

4.1.3.1 Контроллер UserTablesController

Данный контроллер работает также как и [веб-API для сервиса А](#), его отличие лишь в том, что он не отображает, кто из пользователей является специалистом.

4.1.3.2 Контроллер PassesController

Данный контроллер работает также как и [веб-API для сервиса А](#).

4.1.3.3 Контроллер RecordCurrentsController

Этот контроллер отвечает за бронирование помещений пользователей. Он позволяет просмотреть какие комнаты были забронированы пользователем, добавить бронирование на комнату (при этом будет проверяться что пользователь не приглашен и не имеет другое забронированное помещение на данный промежуток времени), убрать бронирование на

комнату до его окончания, а также удалить все бронирования для пользователя, в случае возникновения не предвиденных обстоятельств.

4.1.3.4 Контроллер RecordEndedsController

Этот контроллер позволяет пользователям просматривать историю комнат, которые они бронировали, при этом получая дополнительную информацию, такую как: какие пользователи были приглашены, а еще какая комната бронировалась. Также данный контроллер позволяет оценивать пользователю комнату или менять ее оценку. Еще в возможности данного контроллера входит выдача рейтинга для определенной комнаты, на основе ее оценок от пользователей.

4.1.3.5 Контроллер InvitesCurrentsController

Данный контроллер позволяет приглашать людей к забронированному помещению, просматривать всех приглашенных пользователей, а также удалять определенного пользователя из списка приглашенных. При добавлении пользователя к забронированному помещению проверяется его возможность присутствовать, а именно то, что он не забронировал другое помещение или не приглашен в другую комнату на данный промежуток времени.

4.1.3.6 Контроллер InvitesEndedsController

Данный контроллер позволяет пользователю просматривать все приглашения которые он получал вместе с информацией о самом бронировании и человеком, который его пригласил.

4.1.3.7 Использование JWT авторизации

Данный раздел аналогичен по содержанию разделу [4.1.2.7 Использование JWT авторизации](#). Так как веб-API для сервиса Б тоже имеет доступ к конфиденциальной информации о пользователях, их бронированиях помещений и приглашений от других пользователей.

4.1.4 Разработка веб-приложения

Веб-приложение будет реализовано благодаря фреймворку ASP.NET Core, с использованием платформы Blazor, которая позволяет использовать код на C# и HTML совместно. Использование Blazor повысит продуктивность, позволяя использовать и писать логику страниц в том же файле, где и HTML код.

В разрабатываемом веб приложении будут использоваться все разработанные в сервисах А и Б модели данных, так как мы их будем принимать, отправляя HTTP-запросы. Для повышения безопасности системы, для некоторых запросов необходимо помимо данных, передавать в заголовке JWT (токен), который мы будем получать от API при прохождении авторизации.

Также, для большей надежности приложения, для отправки HTTP-запросов, было решено использовать интерфейс `IHttpClientFactory`, который имеет ряд преимуществ, по сравнению с использованием обычных экземпляров `HttpClient`-ов. Его использование также подразумевает создание отдельных интерфейсов, для получения различных данных.

Перейдем к созданию пользовательского интерфейса. В общем виде, пользовательский интерфейс будет разделен на 3 части: Меню навигации, шапка сайта, а также его тело. Пример пользовательского интерфейса приведен на Рис. 13.

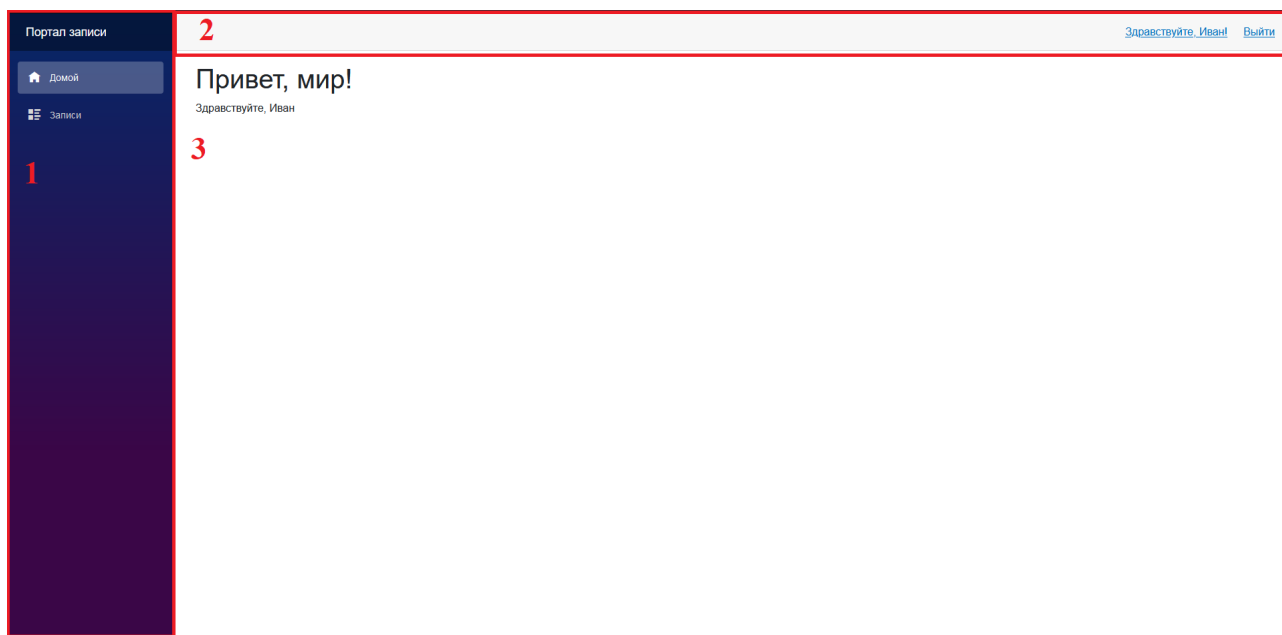


Рис. 13. Структура страницы сайта, где под 1 – раздел меню навигации, 2 – раздел шапки сайта, 3 – раздел тела сайта.

Стоит отметить, что на сайт могут заходить все желающие. При этом может возникать проблема, что неавторизованным пользователям будет доступен интерфейс авторизованного пользователя. Для избегания данной ситуации, было решено использовать Razor-компоненты, а именно `<AuthorizeView>`, который позволяет поделить контент на 2 части: который показывается авторизованным пользователям, а также который показывается неавторизованным пользователям. Пример работы данного компонента приведен на Рис. 14.

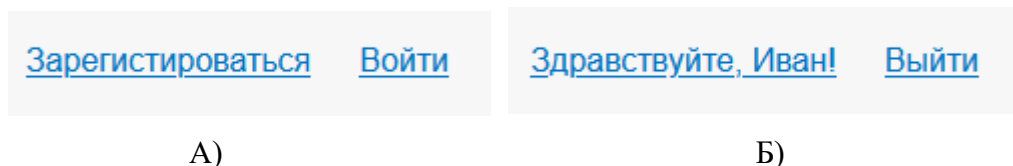
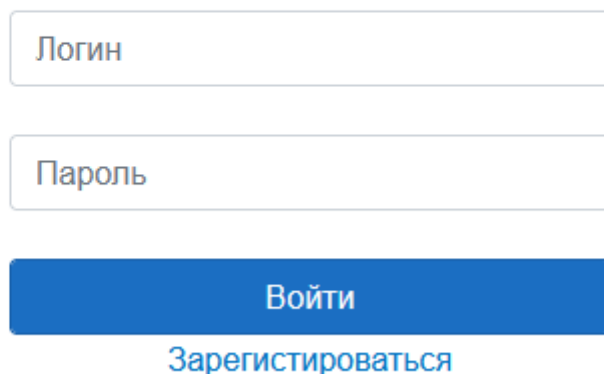


Рис. 14. Вид шапки сайта для неавторизованного пользователя (А) и вид шапки сайта для авторизованного пользователя (Б).

Далее давайте рассмотрим, как будет выглядеть страница авторизации и страница регистрации пользователей. Страница авторизации представлена на Рис. 15. Она не будет содержать ни раздела навигации, ни шапки сайта. Это обусловлено тем, что весь основной

функционал будет доступен пользователю лишь после авторизации, поэтому ему необходимо пройти данный процесс.

Сервис для записи и бронирования помещений



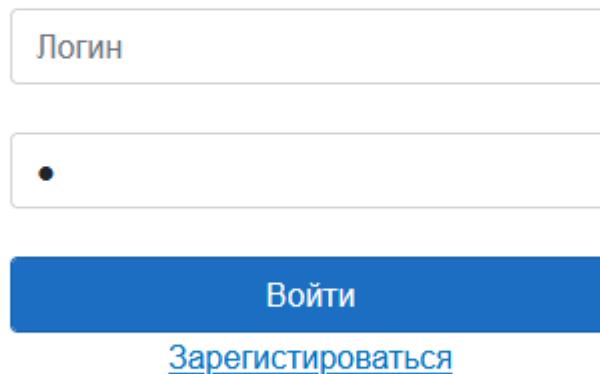
The image shows a user authentication form. It consists of two input fields: the first is labeled 'Логин' (Login) and the second is labeled 'Пароль' (Password). Below these fields is a blue button labeled 'Войти' (Login). Underneath the button is a blue underlined link that says 'Зарегистрироваться' (Register).

Рис. 15. Страница авторизации пользователя

Стоит отметить, что при нажатии на кнопку «Login», перед отправкой данных о пользователе в сервисы, для получения JWT-токенов, необходимо провести валидацию полей на то, что они содержат информацию в необходимом формате, а именно, что они не пустые и имеют определённый размер (логин от 4 до 20 символов, пароль от 6 до 20 символов).

При возникновении соответствующей проблемы при валидации логина и пароля, данные не отправляются в сервисы, но сообщение об ошибке будет показано пользователю, как представлено на Рис. 16.

Сервис для записи и бронирования помещений



The image shows a login form with two input fields. The first field is labeled 'Логин' (Login) and is empty. The second field is for a password, indicated by a single dot, and it has a red error message below it: '• Пароль должен иметь длину от 6 до 20 символов' (Password must be 6 to 20 characters long). Below the password field is a blue button labeled 'Войти' (Login). Below the button is a blue link labeled 'Зарегистрироваться' (Register).

- Необходимо указать логин
- Пароль должен иметь длину от 6 до 20 символов

Рис. 16. Сообщения об ошибке при валидации данных. Для логина не пройдена проверка на пустоту, для пароля не пройдена проверка на длину.

Если валидация данных пройдена успешно, то запрос отправляется в сервисы. В результате работы которых, мы получаем 2 ответа от каждого сервиса (А и Б соответственно). В случае если оба сервиса вернули сообщение об ошибке, то делается вывод о неправильно введённом пароле или логине. В случае если только один из серверов вернул результат об успехе, то пользователя просят попробовать авторизоваться чуть позже, так как скорей всего один из сервисов в данный момент не работает. Пример сообщения об ошибке представлено на Рис. 17.

Сервис для записи и бронирования помещений

4444

•••••

Войти

[Зарегистрироваться](#)

Неправильный логин или пароль!

Рис. 17. Ошибка о неправильном пароле пользователя.

В случае успешной авторизации, пользователь переходит в главную страницу сайта, имея при этом весь функционал, соответствующий авторизованным пользователям.

Также была создана страница для регистрации пользователя, она имеет подобное строение, валидацию полей, перед отправкой запроса в сервисы, а также сообщение об ошибках (в данном случае, будет следующее сообщение об ошибке: «Данный логин не является уникальным и вам необходимо выбрать другой логин»). В случае успешной регистрации, пользователь перенаправляется на страницу «Login», для прохождения процесса авторизации. Пример страницы регистрации приведен на Рис. 18.

Сервис для записи и бронирования помещений

Имя

Фамилия

Отчество

+7 XXX XXX XX XX

e-mail

Логин

Пароль

Подтвердите пароль

Зарегистрироваться

[Уже есть аккаунт? Войдите в него!](#)

Рис. 18. Страница регистрации пользователя

Также была создана страница, которая выводит все активные записи пользователя на данный момент. Каждая запись представлена в виде плитки с необходимой о ней информацией, такой как: **ФИО специалиста**, его **специальность**, **время приема**, **адрес** и **номер помещения**, где будет проходить прием. Пример данных плиток представлен на Рис. 19. Также каждая плитка будет содержать слово «**Отменить**», при нажатии на которое, запись будет отменена.

<div>Записаться</div> <div>Давид Саутов Администратор</div> <div>26.12.2022 10:00:00 - 26.12.2022 11:30:00</div> <div>Адрес: Каширское ш., 31, Корпус А. Комната: 100 Отменить</div>	<div>Давид Саутов Администратор</div> <div>27.12.2022 11:40:00 - 27.12.2022 12:00:00</div> <div>Адрес: Каширское ш., 31, Корпус А. Комната: 100 Отменить</div>	<div>Давид Саутов Администратор</div> <div>28.12.2022 19:10:00 - 28.12.2022 19:40:00</div> <div>Адрес: Каширское ш., 31, Корпус Б. Комната: 101 Отменить</div>	<div>Давид Саутов Администратор</div> <div>29.12.2022 16:00:00 - 29.12.2022 16:30:00</div> <div>Адрес: Каширское ш., 31, Корпус Б. Комната: 101 Отменить</div>
--	--	--	--

Рис. 19. Страница с записями пользователя.

Еще была создана страница с добавлением записей пользователю.

Она содержит компоненты с множественным выбором. Благодаря этим компонентам, пользователь выберет специалиста и помещение, где будет проходить прием.

После выбора специалиста, пользователь сумеет выбрать время записи, которое будет представлено в виде интерактивного ежедневника, где возможные варианты для записи будут указаны в виде голубых плиток. При нажатии на определенную плитку, пользователь пишет желаемое время. В случае если время находится в диапазоне данной плитки, то тогда запрос о записи к специалисту отправляется в Сервис А, запрос о бронировании отправляется в сервис Б, в случае успешного выполнения обоих запросов, пользователь уведомляется об успешной записи и может просмотреть ее на соответствующей странице. В случае, возникновения ошибки, пользователь уведомляется о ней, страница получает обновленные данные и пользователя просят снова выбрать время для приема. Страница для добавления записей пользователю представлена на Рис. 20.

Рис. 20, компонент для выбора специалиста.

4.2 Проверка работоспособности программного продукта и проведение тестирования

В данном подразделе проводится проверка работоспособности как отдельных составляющих программного продукта. Производится обработка исключительных ситуаций при их обнаружении. Также будет проведена обработка результатов, полученных в ходе экспериментов.

В ходе проверки работоспособности сервиса Б было выявлена ошибка, которая заключалась в неправильном виде запроса о бронировании комнат пользователя, совмещенных с приглашениями других людей в эту комнату. Ошибка возникала из-за связи один-к-одному между бронью и приглашениями. Данная ошибка решается переопределением данной связи как один-ко-многим, что позволяет убрать дублирующие записи о брони, для каждого приглашения, при использовании Entity Framework.

Также была обнаружена ошибка, которая позволяет пользователям создавать записи на далёкие даты, по сравнению с текущей. Хотя в требованиях к проекту было указан промежуток в одну неделю максимум. Данная ошибка была решена введением проверки даты с текущей датой при попытке записи.

Перейдем к тестированию общей работоспособности системы. Создадим специалиста. Также создадим 2 пользователей. По итогу был создан специалист с уникальным идентификатором 4. Также было создано два пользователя с уникальными идентификаторами 5 и 6.

Попробуем записать пользователей под uid 5 и 6 на одно и тоже время к специалисту 4, но выберем разные помещения. По итогу получаем, что пользователь с uid 5 сумел записаться к специалисту, а пользователь с uid 6 не сумел. Результаты тестирования приведены на рис. 21 и рис. 22.

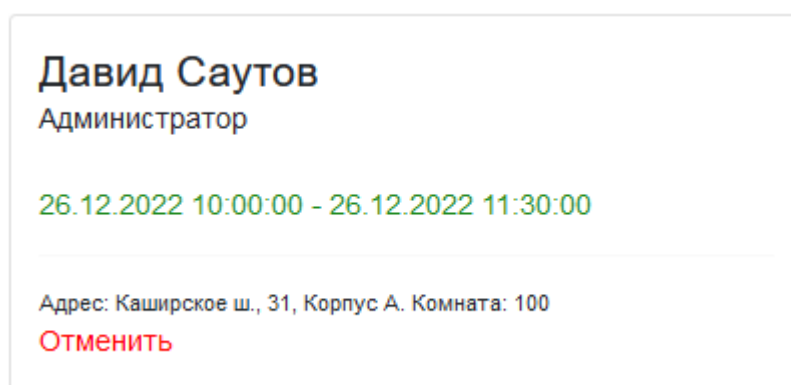
A screenshot of a booking confirmation interface. At the top, the name "Давид Сауров" is displayed in a large, bold, dark blue font, with the title "Администратор" in a smaller, dark blue font below it. Below this, the booking time "26.12.2022 10:00:00 - 26.12.2022 11:30:00" is shown in a green font. A horizontal line separates this from the address "Адрес: Каширское ш., 31, Корпус А. Комната: 100" in a dark blue font. At the bottom, there is a red button with the text "ОТМЕНИТЬ" in white.

Рис. 21. Созданная запись у пользователя с UID 5

Не удалось записаться к специалисту, попробуйте записаться заново

Рис. 22. Отказ в записи пользователю с UID 6

4.3 Вывод

В процессе разработки данной программы использовались современные инструменты, которые позволили создать 2 веб-API для сервиса А и Б, а также веб-приложение, которое выступает не только как сайт, но и узел-координатор при попытке распределения ресурсов предоставляемых этими 2 веб-API. В дальнейшем планируется развить данное приложение, включить в него больший функционал и сделать его более безопасным, а именно планируется внедрить refresh token, который позволит уменьшить срок жизни JWT-токена и предотвратить длительный доступ к приложению злоумышленников при его перехвате. Также планируется добавить другие страницы, которые позволят изменять данные пользователей и специалистов. Еще необходимо реализовать role-based авторизацию для специалистов и обычных пользователей (возможно еще добавлю администратора). В качестве развития предметной области, планируется ввести учет времени для того, чтобы специалист успевал добираться из одной комнаты в другую.

Заключение

В данной работы была рассмотрена проблема синхронизации двух сервисов. Эта проблема была изучена и было предложено несколько подходов для ее устранения. Среди всех способов был выбран способ с централизованным узлом координатором, который на основе ответов от двух сервисов утверждает запись или ее отклоняет. Были выделены основные группы пользователей разрабатываемой системы, их роли и возможности. Также была разработана архитектура приложения и выбраны инструменты для ее реализации. По итогу разработки данной системы была проведено ее тестирование, а также определено как дальше данная система будет дорабатываться и развиваться.

ЛИТЕРАТУРА

1.	Бёрнс Б. Распределенные системы. Паттерны проектирования. / Бёрнс Б. – СПб: Питер, 2021 – 224 с.
2.	Косяков М.С. Введение в распределенные вычисления. / М.С. Косяков – СПб: НИУ ИТМО, 2014 -155с.
3.	Фаулер М. UML. Основы: краткое руководство по стандартному языку объектного моделирования / Мартин Фаулер – Пер. с англ. – СПб: Символ-Плюс, 2011 – 184 с.
4.	Куликов С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов 3-е изд. -Минск: Четыре четверти, 2020 - 312 с.
5.	Хлудова М.В. Синхронизация распределённых приложений реального времени // Информатика, телекоммуникации и управление. / М.В. Хлудова – 2009. – №6. – С. 88–92.
6.	Чамберс Дж. ASP.NET Core Разработка приложений / Джеймс Чамберс, Дэвид Пэккетт, Саймон Тиммс. - СПб.: Питер, 2018. — 464 с.: ил. - (Серия «Для профессионалов»).
7.	Бабичев С. Л. Распределенные системы : Учебное пособие для вузов / Бабичев С. Л., Коньков К. А.. - Москва : Юрайт, 2020. - 507 с
8.	Resca S. Hands-On RESTful Web Services with ASP.NET Core 3: Design production-ready, testable, and flexible RESTful APIs for web applications and microservices / Samuele Resca - Packt Publishing; 1st edition, 2019. – 780 p. – ISBN-13 978-1789537611
9.	Himschoot P. Blazor Revealed: Building Web Applications in .NET / Peter Himschoot - Apress; 1st ed. Edition, 2019. – 285 p. – ISBN-13 978-1484243428
10.	Litvinavicius T. Exploring Blazor: Creating Hosted, Server-side, and Client-side Applications with C# / Taurius Litvinavicius, Apress; 1st ed. Edition, 2019. – 199 p. – ISBN-13 - 978-1484254455
11.	Trivedi J. Building A Web App With Blazor And Asp .Net Core: Create A Single Page App With Blazor Server And Entity Framework Core: Create a Single Page App with ... and Entity Framework Core / Jignesh Trivedi, BPB Publications, 2020. – 220 p. – ISBN-13 - 978-9389845457
12.	Washington M. An Introduction to Building Applications with Blazor: How to get started creating applications using this exciting easy to use Microsoft C# framework / Michael Washington, BlazorHelpWebsite.com, 2019. – 339 p. ISBN - 1688540040

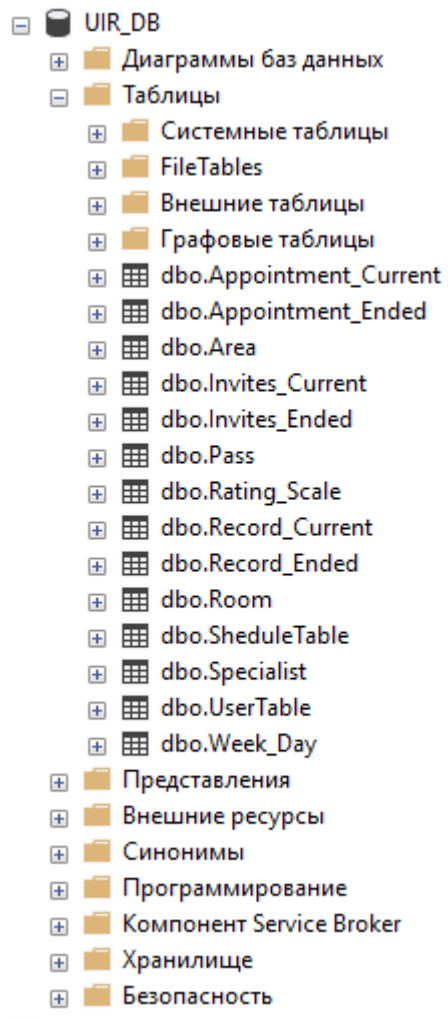
13.	Engström J. Web Development with Blazor: A hands-on guide for .NET developers to build interactive UIs with C# / Jimmy Engström, Packt Publishing Limited, 2021. – 310 p. – ISBN-13 - 978-1800208728
14.	Pattankar M. Mastering ASP.NET Web API: Build powerful HTTP services and make the most of the ASP.NET Core Web API platform / Mithun Pattankar, Malendra Hurbuns, Packt Publishing; 1st edition, 2017. 332 p. – ISBN - 1786463954
15.	Price M. C# 11 and .NET 7 – Modern Cross-Platform Development Fundamentals: Start building websites and services with ASP.NET Core 7, Blazor, and EF Core 7 / Mark J. Price, Packt Publishing; 7th edition, 2022. 818 p. – ISBN-13 - 978-1803237800
16.	RFC 7519. JSON Web Token (JWT). 2015. . [Электронный ресурс]. URL: https://datatracker.ietf.org/doc/html/rfc7519.html (дата обращения 22.12.2022).
17.	Janoky L. V., Levendovszky J., Ekler P. A Novel JWT Revocation Algorithm [Электронный ресурс]. URL: http://real.mtak.hu/113127/1/CSCS2020.pdf (дата обращения: 22.12.2022)
18.	Janoky L. An analysis on the revoking mechanisms for JSON Web Tokens / Janoky L. V., Levendovszky J., Ekler P., International Journal of Distributed Sensor Networks, 14(9), 2018. – 10 p.
19.	Тестирование программного обеспечения. Базовый курс / С. С. Куликов. — 3-е изд. — Минск: Четыре четверти, 2020. — 312 с.
20.	Осипов Д. Технологии проектирования баз данных. / Осипов Д. Л., – М.: ДМК Пресс, 2019. – 498 с.
21.	Free Blazor Components 70+ controls by Radzen [Электронный ресурс]. URL: https://blazor.radzen.com (дата обращения: 22.12.2022)
22.	Overview of ASP.NET Core [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0 (дата обращения: 22.12.2022)
23.	ASP.NET Core Blazor [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/aspnet/core/blazor/?view=aspnetcore-7.0 (дата обращения: 22.12.2022)
24.	Tutorial: Create a web API with ASP.NET Core [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/aspnet/core/tutorials/first-web-api?view=aspnetcore-7.0&tabs=visual-studio (дата обращения: 22.12.2022)

25.	<p>Выбор пользовательского веб-интерфейса ASP.NET Core [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/aspnet/core/tutorials/choose-web-ui?view=aspnetcore-7.0 (дата обращения: 22.12.2022)</p>
-----	--

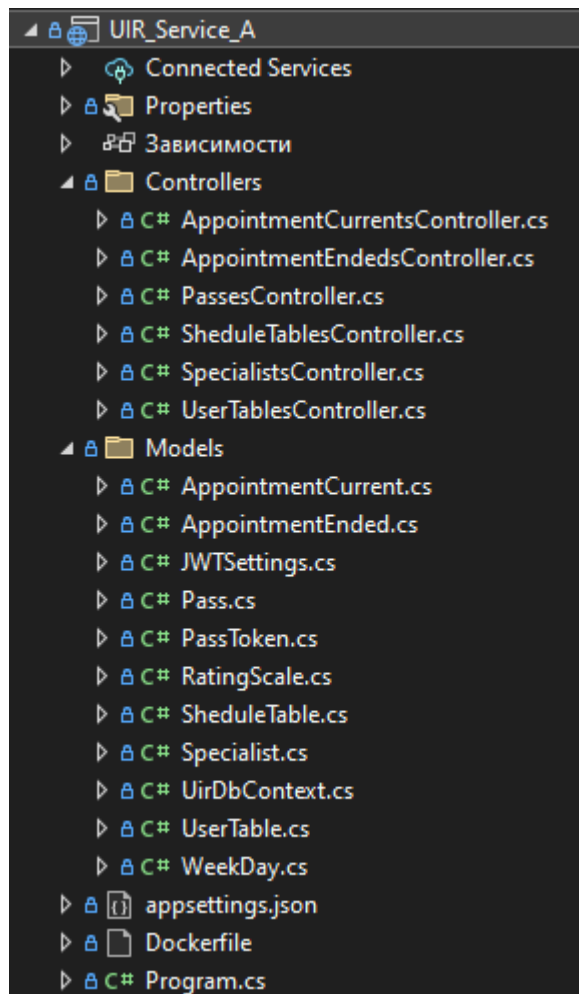
ПРИЛОЖЕНИЕ 1

Модульная спецификация проекта.

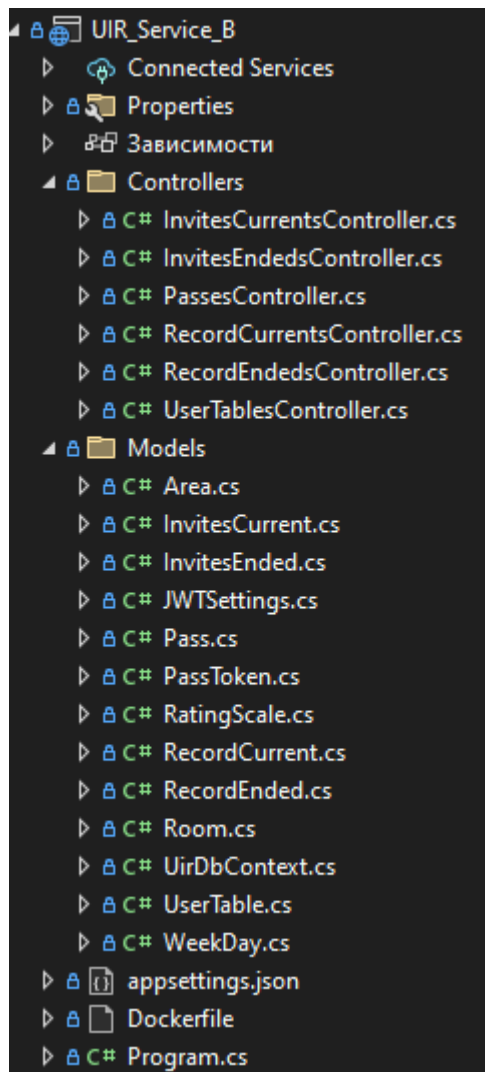
БД имеет следующую структуру:



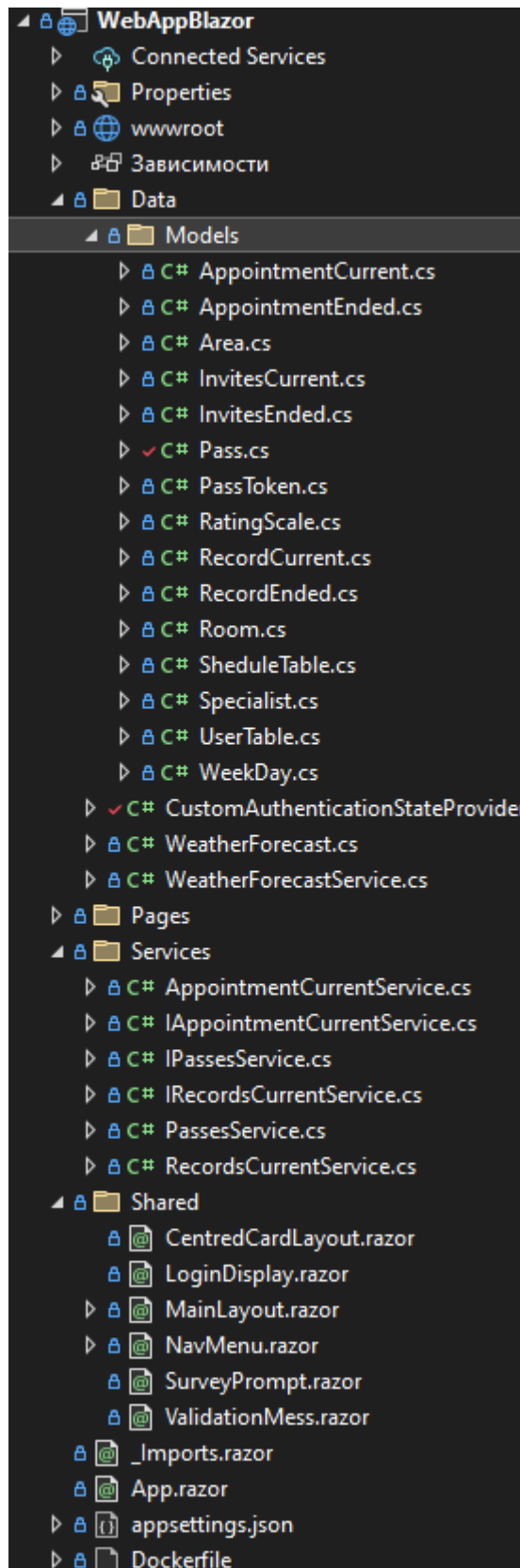
Веб-API для сервиса А имеет следующую структуру:



Веб-АРІ для сервиса Б имеет следующую структуру:



Веб-приложение имеет следующую структуру:



Скрипт для создания БД

```
use UIR_DB;
CREATE TABLE Appointment_Current
(
    Appointment_ID      int IDENTITY(1,1) PRIMARY KEY NOT NULL ,
    Specialist_ID       int NOT NULL ,
    UserUIR_ID          int NOT NULL ,
```

```

        From1          datetime NOT NULL,
        To1            datetime NOT NULL ,
    )
go

```

```

CREATE TABLE Appointment_Ended
(
    Appointment_ID      int PRIMARY KEY NOT NULL ,
    Specialist_ID       int NOT NULL ,
    UserUIR_ID          int NOT NULL ,
    From1              datetime NOT NULL ,
    To1                datetime NOT NULL ,
    Rating_ID          int NULL
)
go

```

```

CREATE TABLE Area
(
    Area_ID             int IDENTITY(1,1) PRIMARY KEY NOT NULL ,
    Area_Name           varchar(200) NOT NULL ,
    Area_Location       varchar(200) NOT NULL ,
    From1              time(0) NOT NULL ,
    To1                time(0) NOT NULL
)
go

```

```

CREATE TABLE Invites_Current
(
    Record_ID           int PRIMARY KEY NOT NULL ,
    UserUIR_ID          int NOT NULL ,
    Additional_Info     varchar(200) NULL
)
go

```

```

CREATE TABLE Invites_Ended
(
    Record_ID           int PRIMARY KEY NOT NULL ,
    UserUIR_ID          int NOT NULL ,
    Additional_Info     varchar(200) NULL
)
go

```

```

CREATE TABLE Pass
(
    UserUIR_ID          int PRIMARY KEY NOT NULL ,
    User_login          varchar(200) NOT NULL ,
    Password_Hash       varchar(2000) NOT NULL
)
go

```

```

CREATE TABLE Rating_Scale
(
    Rating_ID           int IDENTITY(1,1) PRIMARY KEY NOT NULL ,
    Rating              int NOT NULL
)

```

```
)
go
```

```
CREATE TABLE Record_Current
(
    Record_ID          int IDENTITY(1,1) PRIMARY KEY NOT NULL ,
    Room_ID            int NOT NULL ,
    UserUIR_ID         int NOT NULL ,
    From1              datetime NOT NULL ,
    To1                datetime NOT NULL
)
go
```

```
CREATE TABLE Record_Ended
(
    Record_ID          int PRIMARY KEY NOT NULL ,
    Room_ID            int NOT NULL ,
    UserUIR_ID         int NOT NULL ,
    From1              datetime NOT NULL ,
    To1                datetime NOT NULL ,
    Rating_ID          int NULL
)
go
```

```
CREATE TABLE Room
(
    Room_ID            int IDENTITY(1,1) PRIMARY KEY NOT NULL ,
    Additional_information varchar(200) NULL ,
    Area_ID            int NULL ,
    Room_Number        varchar(20) NOT NULL
)
go
```

```
CREATE TABLE SheduleTable
(
    Specialist_ID      int NOT NULL ,
    Weekday_ID         int NOT NULL ,
    From1              datetime NOT NULL ,
    To1                datetime NOT NULL ,
    Lunch_Start        datetime NOT NULL ,
    Lunch_End          datetime NOT NULL ,
    CONSTRAINT PK_Schedule_Specialist_ID_Weekday_ID PRIMARY KEY CLUSTERED (Specialist_ID
ASC,Weekday_ID ASC)
)
go
```

```
CREATE TABLE Specialist
(
    Specialist_ID      int PRIMARY KEY NOT NULL,
    Specialization     varchar(200) NOT NULL ,
    Degree             varchar(200) NOT NULL ,
    Experience         varchar(200) NOT NULL ,
    Additional_Info    varchar(2000) NULL
)
go
```

```
CREATE TABLE UserTable
```

```
(
    UserUIR_ID          int IDENTITY(1,1) PRIMARY KEY NOT NULL,
    Last_Name           varchar(200) NOT NULL ,
    First_Name          varchar(200) NOT NULL ,
    Middle_Name         varchar(200) NULL ,
    Phone_Number        varchar(200) NOT NULL ,
    e_mail              varchar(200) NOT NULL
)
go
```

```
CREATE TABLE Week_Day
```

```
(
    Weekday_ID          int IDENTITY(1,1) PRIMARY KEY NOT NULL ,
    Week_Day            varchar(200) NOT NULL
)
go
```

```
ALTER TABLE Appointment_Current
```

```
ADD CONSTRAINT R_7 FOREIGN KEY (Specialist_ID) REFERENCES Specialist(Specialist_ID)
ON UPDATE CASCADE
ON DELETE CASCADE
```

```
go
```

```
ALTER TABLE Appointment_Current
```

```
ADD CONSTRAINT R_9 FOREIGN KEY (UserUIR_ID) REFERENCES UserTable(UserUIR_ID)
ON UPDATE CASCADE
ON DELETE CASCADE
```

```
go
```

```
ALTER TABLE Appointment_Ended
```

```
ADD CONSTRAINT R_30 FOREIGN KEY (UserUIR_ID) REFERENCES UserTable(UserUIR_ID)
ON UPDATE CASCADE
ON DELETE CASCADE
```

```
go
```

```
ALTER TABLE Appointment_Ended
```

```
ADD CONSTRAINT R_31 FOREIGN KEY (Specialist_ID) REFERENCES Specialist(Specialist_ID)
ON UPDATE CASCADE
ON DELETE CASCADE
```

```
go
```

```
ALTER TABLE Appointment_Ended
```

```
ADD CONSTRAINT R_32 FOREIGN KEY (Rating_ID) REFERENCES Rating_Scale(Rating_ID)
ON UPDATE CASCADE
ON DELETE CASCADE
```


go

```
ALTER TABLE Invites_Current
  ADD CONSTRAINT R_25 FOREIGN KEY (Record_ID) REFERENCES Record_Current(Record_ID)
  ON UPDATE CASCADE
  ON DELETE CASCADE
```

go

```
ALTER TABLE Invites_Current
  ADD CONSTRAINT R_26 FOREIGN KEY (UserUIR_ID) REFERENCES UserTable(UserUIR_ID)
  ON UPDATE CASCADE
  ON DELETE CASCADE
```

go

```
ALTER TABLE Invites_Ended
  ADD CONSTRAINT R_36 FOREIGN KEY (Record_ID) REFERENCES Record_Ended(Record_ID)
  ON UPDATE CASCADE
  ON DELETE CASCADE
```

go

```
ALTER TABLE Invites_Ended
  ADD CONSTRAINT R_37 FOREIGN KEY (UserUIR_ID) REFERENCES UserTable(UserUIR_ID)
  ON UPDATE CASCADE
  ON DELETE CASCADE
```

go

```
ALTER TABLE Pass
  ADD CONSTRAINT R_28 FOREIGN KEY (UserUIR_ID) REFERENCES UserTable(UserUIR_ID)
  ON UPDATE CASCADE
  ON DELETE CASCADE
```

go

```
ALTER TABLE Record_Current
  ADD CONSTRAINT R_17 FOREIGN KEY (Room_ID) REFERENCES Room(Room_ID)
  ON UPDATE CASCADE
  ON DELETE CASCADE
```

go

```
ALTER TABLE Record_Current
  ADD CONSTRAINT R_20 FOREIGN KEY (UserUIR_ID) REFERENCES UserTable(UserUIR_ID)
  ON UPDATE NO ACTION
  ON DELETE NO ACTION
```

go

```

ALTER TABLE Record_Ended
    ADD CONSTRAINT R_33 FOREIGN KEY (Rating_ID) REFERENCES Rating_Scale(Rating_ID)
        ON UPDATE CASCADE
        ON DELETE CASCADE
go

```

```

ALTER TABLE Record_Ended
    ADD CONSTRAINT R_34 FOREIGN KEY (UserUIR_ID) REFERENCES UserTable(UserUIR_ID)
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
go

```

```

ALTER TABLE Record_Ended
    ADD CONSTRAINT R_35 FOREIGN KEY (Room_ID) REFERENCES Room(Room_ID)
        ON UPDATE CASCADE
        ON DELETE CASCADE
go

```

```

ALTER TABLE Room
    ADD CONSTRAINT R_16 FOREIGN KEY (Area_ID) REFERENCES Area(Area_ID)
        ON UPDATE CASCADE
        ON DELETE CASCADE
go

```

```

ALTER TABLE SheduleTable
    ADD CONSTRAINT R_3 FOREIGN KEY (Specialist_ID) REFERENCES Specialist(Specialist_ID)
        ON UPDATE CASCADE
        ON DELETE CASCADE
go

```

```

ALTER TABLE SheduleTable
    ADD CONSTRAINT R_4 FOREIGN KEY (Weekday_ID) REFERENCES Week_Day(Weekday_ID)
        ON UPDATE CASCADE
        ON DELETE CASCADE
go

```

```

ALTER TABLE Specialist
    ADD CONSTRAINT R_2 FOREIGN KEY (Specialist_ID) REFERENCES UserTable(UserUIR_ID)
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
Go

```

Модели

AppointmentCurrent.cs

```
using System;
using System.Collections.Generic;

namespace UIR_WebAPI_1.Models;

public partial class AppointmentCurrent
{
    public int AppointmentId { get; set; }

    public int SpecialistId { get; set; }

    public int UserUirId { get; set; }

    public DateTime From1 { get; set; }

    public DateTime To1 { get; set; }

    public virtual Specialist? Specialist { get; set; }

    public virtual UserTable? UserUir { get; set; }
}
```

AppointmentEnded.cs

```
using System;
using System.Collections.Generic;

namespace UIR_WebAPI_1.Models;

public partial class AppointmentEnded
{
    public int AppointmentId { get; set; }

    public int SpecialistId { get; set; }

    public int UserUirId { get; set; }

    public DateTime From1 { get; set; }

    public DateTime To1 { get; set; }

    public int? RatingId { get; set; }

    public virtual RatingScale? Rating { get; set; }

    public virtual Specialist? Specialist { get; set; }

    public virtual UserTable? UserUir { get; set; }
}
```

И другие файлы, которые имеют схожий формат

Контроллеры

AppointmentCurrentsController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
```

```

using UIR_WebAPI_1.Models;

namespace UIR_WebAPI_1.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class AppointmentCurrentsController : ControllerBase
    {
        private readonly UirDbContext _context;

        public AppointmentCurrentsController(UirDbContext context)
        {
            _context = context;
        }

        // GET: api/AppointmentCurrents/5
        [HttpGet("{id}")]
        public async Task<ActionResult<AppointmentCurrent>> GetAppointmentCurrent(int
id)
        {
            if (_context.AppointmentCurrents == null)
            {
                return NotFound();
            }
            var appointmentCurrents = await
_context.AppointmentCurrents.FindAsync(id);

            if (appointmentCurrents == null)
            {
                return NotFound();
            }

            return appointmentCurrents;
        }

        // GET: api/AppointmentCurrents/User/5
        [HttpGet("User/{id}")]
        public async Task<ActionResult<IEnumerable<AppointmentCurrent>>>
GetAppointmentCurrentUser(int id)
        {
            if (_context.AppointmentCurrents == null)
            {
                return NotFound();
            }
            var appointmentCurrents = await
_context.AppointmentCurrents.Where(ac => ac.UserUirId == id).ToListAsync();

            if (appointmentCurrents == null)
            {
                return NotFound();
            }

            return appointmentCurrents;
        }

        // GET: api/AppointmentCurrents/Specialist/5
        [HttpGet("Specialist/{id}")]
        public async Task<ActionResult<IEnumerable<AppointmentCurrent>>>
GetAppointmentCurrentSpec(int id)
        {
            if (_context.AppointmentCurrents == null)
            {
                return NotFound();
            }
            var appointmentCurrents = await
_context.AppointmentCurrents.Where(ac => ac.SpecialistId == id).ToListAsync();

```

```

        if (appointmentCurrents == null)
        {
            return NotFound();
        }

        return appointmentCurrents;
    }

    // POST: api/AppointmentCurrents
    // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
    [HttpPost]
    public async Task<ActionResult<AppointmentCurrent>>
    PostAppointmentCurrent(AppointmentCurrent appointmentCurrent)
    {
        if (_context.AppointmentCurrents == null)
        {
            return Problem("Entity set 'UirDbContext.AppointmentCurrents' is
null.");
        }
        if (appointmentCurrent.From1.TimeOfDay >=
appointmentCurrent.To1.TimeOfDay)
            return BadRequest(-1);
        var transaction = await
_context.Database.BeginTransactionAsync(System.Data.IsolationLevel.Serializable);
        var SpecView = await _context.Specialists
            .Include(spec => spec.SheduleTables)
            .Include(spec => spec.AppointmentCurrents)
            .Where(spec => spec.SpecialistId == appointmentCurrent.SpecialistId)
            .ToListAsync();
        if (SpecView == null)
            return BadRequest(1);
        foreach (var i in SpecView[0].SheduleTables)
        {
            if (i.WeekdayId == (int)appointmentCurrent.From1.DayOfWeek)
            {
                if (!(i.From1 <= appointmentCurrent.From1.TimeOfDay &&
                    i.LunchStart >= appointmentCurrent.To1.TimeOfDay) ||
                    (i.LunchEnd <= appointmentCurrent.From1.TimeOfDay &&
                    i.To1 >= appointmentCurrent.To1.TimeOfDay))
                {
                    return BadRequest(2);
                }
            }
        }
        foreach (var i in SpecView[0].AppointmentCurrents)
        {
            if (i.From1.Date == appointmentCurrent.From1.Date)
            {
                if (!(i.To1.TimeOfDay <= appointmentCurrent.From1.TimeOfDay ||
                    i.From1.TimeOfDay >= appointmentCurrent.To1.TimeOfDay))
                {
                    return BadRequest(3);
                }
            }
            await _context.AppointmentCurrents.AddAsync(appointmentCurrent);
            await _context.SaveChangesAsync();
            await transaction.CommitAsync();
            return CreatedAtAction(nameof(GetAppointmentCurrent), new { id =
appointmentCurrent.AppointmentId }, appointmentCurrent);
        }

        // DELETE: api/AppointmentCurrents/5
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteAppointmentCurrentUser(int id)
        {
            if (_context.AppointmentCurrents == null)
            {
                return NotFound();
            }

```

```

        }
        var appointmentCurrent = await
_context.AppointmentCurrents.FindAsync(id);
        if (appointmentCurrent == null)
        {
            return NotFound();
        }

        _context.AppointmentCurrents.Remove(appointmentCurrent);
        await _context.SaveChangesAsync();

        return NoContent();
    }

    //Если заболел и не может прийти, то все записи на ближайшую неделю
отменяются
    [HttpDelete("Specialist/{id}")]
    public async Task<IActionResult> DeleteAppointmentCurrentSpec(int id)
    {
        if (_context.AppointmentCurrents == null)
        {
            return NotFound();
        }
        var appointmentCurrents = await
_context.AppointmentCurrents.Where(acs => acs.SpecialistId == id).ToListAsync();
        if (appointmentCurrents == null)
        {
            return NotFound();
        }
        foreach (var i in appointmentCurrents)
            _context.AppointmentCurrents.Remove(i);
        await _context.SaveChangesAsync();
        return NoContent();
    }

    private bool AppointmentCurrentExists(int id)
    {
        return (_context.AppointmentCurrents?.Any(e => e.AppointmentId ==
id)).GetValueOrDefault();
    }
}

```

И другие файлы, которые имеют схожий формат

Прочие файлы

appsettings.json

```

{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "UIR_DB": "Data Source=DESKTOP-6V02F50\\SQLEXPRESS;Initial
Catalog=UIR_DB;Integrated Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;Mult
iSubnetFailover=False"
  }
}

```

Program.cs

```
using Microsoft.EntityFrameworkCore;
using UIR_WebAPI_1.Models;
using System.Resources;
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
//Мб убрать, так как pyрается
builder.Services.AddMvc(opt => opt.EnableEndpointRouting = false)
    .SetCompatibilityVersion(CompatibilityVersion.Latest)
    .AddNewtonsoftJson(opt => opt.SerializerSettings.ReferenceLoopHandling =
ReferenceLoopHandling.Ignore);
builder.Services.AddDbContext<UirDbContext>(opt =>
    opt.UseSqlServer("Name=UIR_DB"));
// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Веб-API Service Б

Модели

InvitesCurrent.cs

```
using System;
using System.Collections.Generic;

namespace UIR_Service_B.Models;

public partial class InvitesCurrent
{
    public int RecordId { get; set; }

    public int UserUirId { get; set; }

    public string? AdditionalInfo { get; set; }

    public virtual RecordCurrent? Record { get; set; }

    public virtual UserTable? UserUir { get; set; }
}
```

```
}
```

И другие файлы, которые имеют схожий формат

Контроллеры

InvitesCurrentsController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using UIR_Service_B.Models;

namespace UIR_Service_B.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class InvitesCurrentsController : ControllerBase
    {
        private readonly UirDbContext _context;

        public InvitesCurrentsController(UirDbContext context)
        {
            _context = context;
        }

        // GET: api/InvitesCurrents/5
        [HttpGet("{id}")]
        public async Task<ActionResult<IEnumerable<InvitesCurrent>>>
GetInvitesCurrent(int id)
        {
            if (_context.InvitesCurrents == null)
            {
                return NotFound();
            }
            var invitesCurrent = await _context.InvitesCurrents.Where(inv =>
inv.RecordId == id).ToListAsync();

            if (invitesCurrent == null)
            {
                return NotFound();
            }

            return invitesCurrent;
        }

        // POST: api/InvitesCurrents
        // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPost]
        public async Task<ActionResult<IEnumerable<InvitesCurrent>>>
PostInvitesCurrent(InvitesCurrent invitesCurrent)
        {
            if (_context.InvitesCurrents == null)
            {
                return Problem("Entity set 'UirDbContext.InvitesCurrents' is null.");
            }
            try
            {
                var rec = await _context.RecordCurrents
                    .SingleAsync(rec => rec.RecordId == invitesCurrent.RecordId);
            }
        }
    }
}
```



```

        if (!(rec.From1.TimeOfDay >= rec.To1.TimeOfDay))
            return BadRequest("From > To");
        var transaction = await
_context.Database.BeginTransactionAsync(System.Data.IsolationLevel.Serializable);
        var userInfo = await _context.UserTables
            .Include(us => us.RecordCurrents)
            .Include(us => us.InvitesCurrents)
            .ThenInclude(inv => inv.Record)
            .SingleAsync(us => us.UserUid == invitesCurrent.UserUid);

        foreach (var i in userInfo.RecordCurrents)
        {
            if (i.From1.Date == rec.From1.Date)
                if (!(i.From1.TimeOfDay >= rec.To1.TimeOfDay &&
                    i.To1.TimeOfDay <= rec.From1.TimeOfDay))
                {
                    return BadRequest("User have a room on this time");
                }
        }
        foreach (var i in userInfo.InvitesCurrents)
        {
            if (i.Record.From1.Date == rec.From1.Date)
                if (!(i.Record.From1.TimeOfDay >= rec.To1.TimeOfDay &&
                    i.Record.To1.TimeOfDay <= rec.From1.TimeOfDay))
                {
                    return BadRequest("User have another invite on this
time");
                }
        }
        _context.InvitesCurrents.Add(invitesCurrent);
        await _context.SaveChangesAsync();
        await transaction.CommitAsync();
    }
    catch (DbUpdateException)
    {
        if (InvitesCurrentExists(invitesCurrent.RecordId))
        {
            return Conflict();
        }
        else
        {
            throw;
        }
    }

    return CreatedAtAction(nameof(GetInvitesCurrent), new { id =
invitesCurrent.RecordId }, invitesCurrent);
}

// DELETE: api/InvitesCurrents/5
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteInvitesCurrent(int recid, int userid)
{
    if (_context.InvitesCurrents == null)
    {
        return NotFound();
    }
    var invitesCurrent = await _context.InvitesCurrents.FindAsync(recid,
userid);
    if (invitesCurrent == null)
    {
        return NotFound();
    }

    _context.InvitesCurrents.Remove(invitesCurrent);
}

```

```

        await _context.SaveChangesAsync();

        return NoContent();
    }

    private bool InvitesCurrentExists(int id)
    {
        return (_context.InvitesCurrents?.Any(e => e.RecordId ==
id)).GetValueOrDefault();
    }
}
}

```

И другие файлы, которые имеют схожий формат

Прочие файлы

appsettings.json

```

{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "UIR_DB": "Data Source=DESKTOP-6V02F50\\SQLEXPRESS;Initial
Catalog=UIR_DB;Integrated Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;Mult
iSubnetFailover=False"
  }
}

```

Program.cs

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Newtonsoft.Json;
using UIR_Service_B.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
builder.Services.AddMvc(opt => opt.EnableEndpointRouting = false)
    .SetCompatibilityVersion(CompatibilityVersion.Latest)
    .AddNewtonsoftJson(opt => opt.SerializerSettings.ReferenceLoopHandling =
ReferenceLoopHandling.Ignore);
builder.Services.AddDbContext<UirDbContext>(opt =>
    opt.UseSqlServer("Name=UIR_DB"));
// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

```

```
app.UseHttpsRedirection();  
app.UseAuthorization();  
app.MapControllers();  
app.Run();
```