

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

Мобильное приложение для театра

Студент _____ Чулков Данила Александрович _____
(подпись, дата) (ФИО полностью)

Руководитель _____ Балашова И. Ю. _____
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В. _____
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ Козлов А.Ю. _____
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А. _____
(подпись) (фамилия, инициалы)

Пенза, 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»

Заведующий кафедрой

_____ А.Ю. Козлов

«__» _____ 2023 г

ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

БАКАЛАВРА

1. Студент Чулков Данила Александрович гр. 19ВП2 факультета
<Фамилия, имя, отчество полностью>

вычислительной техники

2. Тема работы Мобильное приложение для театра

Тема утверждена приказом ПГУ № _____ от " ____ " апреля 2023 года

2. Руководитель работы Балашова Ирина Юрьевна

3. Задание на работу (назначение разработки, исходные данные и т.п.)

1) *Назначение: мобильное приложение для театра с возможностью бронирования билетов.*

2) *Среда выполнения: операционная система Windows 10, смартфон под управлением Android.*

Языковые средства разработки: C#, Xaml, SQL.

Инструментальные средства: MS Visual Studio 2022, MS SQL Server, Draw.io, Postman, Android Emulator

Технологии разработки: ООП, принципы SOLID, шаблоны проектирования, архитектура REST API.

3) Основные функции приложения:

- а) просмотр афиши мероприятий;
- б) просмотр подробной информации о спектакле;
- в) просмотр биографии актеров;
- г) авторизация и регистрация;
- д) бронирование и отмена бронирования билетов.

4. Перечень подлежащих разработке вопросов:

- 1) Анализ предметной области и постановка задачи;
- 2) Проектирование;
- 3) Реализация;
- 4) Контроль качества программного обеспечения;
- 5) План разработки бюджета;
- 6) Оформление пояснительной записки.

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	12.04.2023 – 16.04.2023	
2	Проектирование программных средств	20%	17.04.2023 – 26.04.2023	
3	Реализация программных средств	30%	27.04.2023 – 11.05.2023	
4	Тестирование программных средств	20%	12.05.2023 – 21.05.2023	
5	Оформление пояснительной записки	20%	22.05.2023 – 01.06.2023	

Дата выдачи задания «22» декабря 2022 г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 112 листов, 39 рисунков, 8 таблиц, 17 использованных источников и 4 приложения.

ТЕАТР, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, КЛИЕНТ, СЕРВЕР, БАЗА ДАННЫХ, REST API, СУБД, UML, MVC, СЕРВЕР БАЗЫ ДАННЫХ.

Объект исследования – средства автоматизации рабочих процессов театра.

Цель выпускной квалификационной работы – разработать мобильное приложение, позволяющее пользователям получать актуальную информацию о планирующихся событиях театра.

Технологии разработки – разработка REST API-сервера проводилась на языке программирования C# в среде Microsoft Visual Studio 2022. Клиентское приложение разработано на языке C# с использованием фреймворка Xamarin Forms для мобильных приложений. Для создания базы данных использовался сервер MS SQL Server и СУБД MS SQL Management Studio.

Среда исполнения – смартфон под управлением Android.

Результаты работы – в ходе выполнения выпускной квалификационной работы были созданы клиентская и серверная часть мобильного приложения театра. Все требования к разработке приложения соблюдены.

					ПГУ 09.03.04– 8БР192.22 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Мобильное приложение для театра. Пояснительная записка	Лит.	Лист	Листов
Разраб.		Чулков Д. А.					4	112
Пров.		Балашова И.Ю.				Группа 19ВП2		
Н. Контр.		Такташкин Д.В						
УТВ.								

Содержание

Введение.....	9
1. Анализ мобильных приложений для театра и выделение требований	11
1.1 Анализ предметной области	11
1.2 Обзор продуктов-аналогов.....	11
1.2.1 Сайт кукольного театра города Пензы «Кукольный дом»	12
1.2.2 Сайт Пензенского областного драматического театра.....	13
1.2.3 Мобильное приложение Мариинского театра.....	14
1.2.4 Мобильное приложение «Афиша»	15
1.2.5 Мобильное приложение «Афиша 7»	16
1.3 Сравнительный анализ и критериальная оценка аналогов.....	18
1.4 Исследование вариантов использования	21
1.5 Анализ требований	24
1.6 Постановка задачи	25
2. Проектирование мобильного приложения театра	27
2.1 Архитектура приложения.....	27
2.2 Проектирование структур данных	28
2.2.1 Концептуальная модель базы данных	28
2.2.2 Логическая модель базы данных	30
2.3 Проектирование программных средств.....	31
2.4 Проектирование интерфейса	33
3. Реализация мобильного приложения для театра	39
3.1 Реализация серверной части приложения	39
3.1.1 Реализация базы данных	39
3.1.2 Реализация серверных компонентов	40

4. Контроль качества программного обеспечения	44
4.1 Выбор режима тестирования	44
4.2 Тестирование серверной части мобильного приложения для театра.....	45
4.3 Тестирование клиентской части мобильного приложения для театра	48
4.4 Анализ качества кода.....	49
4.4.1 Измерение удобства поддержки встроенными средствами	
MS Visual Studio	49
4.4.2 Измерение устойчивости и абстрактности кода	50
4.4.3 Главная последовательность	52
4.5 Планирование разработки и оценка бюджета.....	53
Заключение	57
Список использованных источников	59
Приложение А. Глоссарий	61
Приложение Б. Листинг программы	64
Приложение В. Описание сущностей	101
Приложение Г. Результаты тестирования	108

Введение

В последние годы отмечается неизбывный рост интереса к театральному искусству. Исходя из результатов опроса, опубликованного на сайте ВЦИОМ [1], по состоянию на 2022 год 48% населения с той или иной периодичностью посещали театр. Это означает, что популярность театральных представлений среди разных групп населения не убывает, и театр остается популярным местом скрашивания досуга.

Привлечение аудитории является важным процессом, обеспечивающим популярность театра как вида искусства, а также ключевым компонентом увеличения спроса на спектакли конкретного театра. Учитывая этот фактор, можно сделать вывод, что любому современному театру необходимо осуществлять цифровое представление информации, оформленное в виде одного из приложений.

Среди вариантов реализации можно выделить три основные вида: мобильное приложение, компьютерное приложение и web-приложение. Согласно исследованию, опубликованному на ТАСС в 2020 году, 67% россиян используют смартфоны для доступа в Интернет [2]. Использование мобильного приложения обеспечивает театру постоянный, быстрый и недорогой контакт с театральной публикой и создает для зрителя дополнительную ценность в виде удобства. Таким образом, задача разработки мобильного приложения для театра является актуальной.

Целью выпускной квалификационной работы является разработка мобильного приложения для театра.

Объект исследования – средства автоматизации рабочих процессов театра.

Предмет исследования – мобильное приложение для театра.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- провести анализ требований;

- выполнить проектирование программных средств, структур данных и интерфейса;
- реализовать мобильное приложения для театра;
- отладить и протестировать полученное приложение;
- составить план разработки и оценить бюджет разработки.

В работе использованы методы сравнительного анализа, методы проектирования, разработки и тестирования программного обеспечения.

1 Анализ мобильных приложений для театра и выделение требований

1.1 Анализ предметной области

Для проведения анализа требований необходимо предварительно проанализировать предметную область.

Театр представляет собой систему. Основными объектами в ней являются спектакли, актеры, зрители, билеты. Необходимый функционал приложения театра:

- наличие системы авторизации;
- наличие афиши театра, содержащей информацию о предстоящих событиях;
- реализация возможности выбора мест и бронирования билетов;
- наличие карточки спектакля, содержащей подробную информацию о мероприятии;
- наличие списка актеров, участвующих в спектакле.

С точки зрения удобства отображения данных, мобильное приложение является актуальным вариантом. С его помощью пользователь может в любое время и в любом месте посмотреть информацию о готовящихся спектаклях, забронировать билет или прочитать биографию интересующего актера.

Мобильные приложения позволяют театрам быть более доступными, привлекательными и конкурентоспособными, а наличие его также говорит о серьезности организации.

1.2 Обзор продуктов-аналогов

В качестве аналогов разрабатываемого приложения были выбраны следующие сервисы:

- сайт кукольного театра города Пензы «Кукольный дом» (<http://www.puppet-house.ru/>);

- сайт Пензенского областного драматического театра (<https://www.penzateatr.ru/>);
- мобильное приложение Мариинского театра;
- мобильное приложение «Афиша»;
- мобильное приложение «Афиша 7».

Данные решения имеют положительные оценки пользователей и являются популярными. Первые два варианта представляют собой сервисы пензенских театров. Это особенно важно, поскольку необходимо иметь представление о существующих аналогах в своем городе.

1.2.1 Сайт кукольного театра города Пензы «Кукольный дом»

Целью данного web-приложения является предоставление информации о театре и о спектаклях, проводимых в нем. Сайт также позволяет посмотреть актеров и оставить отзыв о спектакле. Интерфейс приложения представлен на рисунке 1.



Рисунок 1 – Интерфейс сайта театра «Кукольный дом»

Преимущества:

- возможность покупки билетов с помощью встроенного сервиса;
- наличие страницы отзывов о спектаклях и других мероприятиях;
- блок новостей, который постоянно обновляется;

- разделение репертуара по возрасту;
- наличие афиши, которая также постоянно обновляется.

Недостатки:

- устаревший интерфейс;
- неполная информация об актерах – нет возможности посмотреть их биографию.

Как видно из проведенного анализа, сайт театра «Кукольный дом» имеет больше преимуществ, чем недостатков, однако наличие устаревшего интерфейса сильно отталкивает новых пользователей.

1.2.2 Сайт Пензенского областного драматического театра

Команда разработчиков «Плюс один» из города Пензы разработала портал для Пензенского областного драматического театра им. Луначарского. Данный сайт имеет огромное количество информации, начиная от блока новостей и заканчивая страницей памяти служителям театра. Интерфейс приложения представлен на рисунке 2.

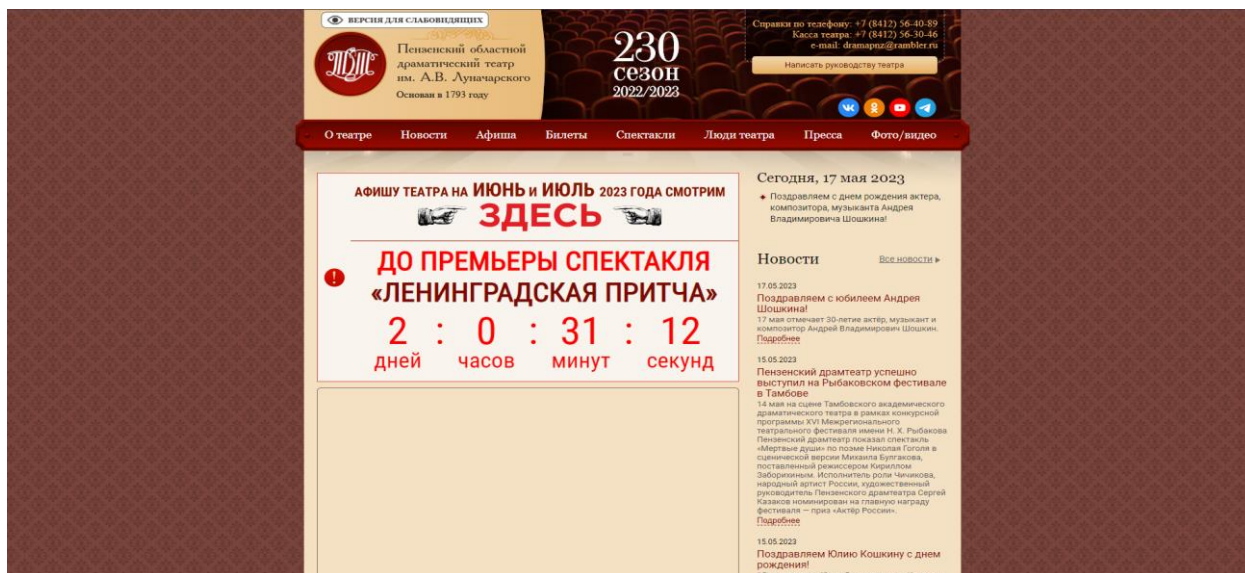


Рисунок 2 – Интерфейс сайта Пензенского областного драматического театра

Преимущества:

- огромная база информации – содержится подробная информация о каждом спектакле и каждом актере;

- наличие версии для слабовидящих с увеличенными элементами интерфейса;
- наличие страницы новостей, которая постоянно обновляется;
- возможность покупки билета;
- наличие галереи фотографий и видеозаписей.

Недостатки:

- огромная база информации также является небольшим недостатком, так как становится сложно найти конкретный данные;
- интерфейс хоть и не является отталкивающим, все равно немного устарел.

Сайт Пензенского областного драматического театра является хорошим примером приложения театра, однако имеет мелкие недостатки.

1.2.3 Мобильное приложение Мариинского театра

Мариинский театр, имеющий здания в Санкт-Петербурге, Владивостоке и Владикавказе, обладает собственным мобильным приложением для операционных систем Android и iOS.

Интерфейс приложения представлен на рисунке 3.

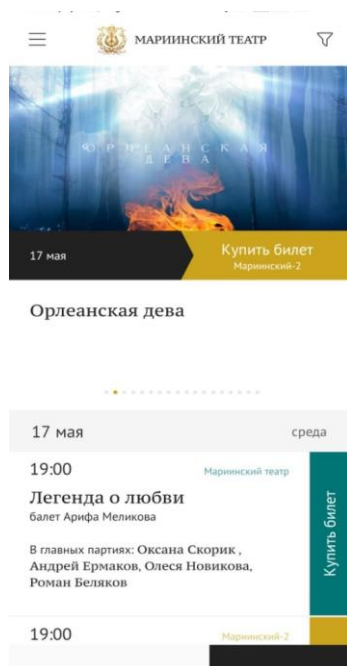


Рисунок 3 – Интерфейс приложения «Мариинский театр»

Преимущества:

- наличие формы авторизации и регистрации;
- подробная информация о каждом спектакле;
- удобное представление информации;
- интуитивно понятный и дружелюбный интерфейс;
- возможность покупки билетов.

Однако следует отметить, что при переходе на страницу биографии актеров открывается сайт театра, что не очень удобно.

Данное приложение является удобным для использования. Во-первых, приятный интерфейс привлекает пользователей. Он не является перегруженным и обладает лишь основной информацией. Во-вторых, приложение работает быстро. Это означает, что запрограммированные алгоритмы были реализованы корректно. Однако тот факт, что для просмотра информации необходимо открывать сайт, является недостатком.

1.2.4 Мобильное приложение «Афиша»

Мобильное приложение «Афиша» является агрегатором мероприятий города. В нем представлена информация о фильмах, спектаклях, заведениях для перекуса и прочем. Интерфейс приложения представлен на рисунке 4.

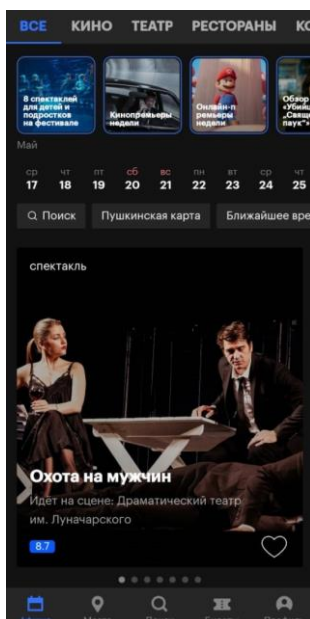


Рисунок 4 – Интерфейс приложения «Афиша»

Преимущества:

- удобная сортировка по дате;
- подробная информация о мероприятиях с оценками пользователей;
- наличие формы авторизации и личного кабинета;
- возможность покупки билета;
- современный интерфейс;
- постоянное обновление данных.

Недостатки:

- перегруженность интерфейса – информации настолько много, что в ней легко запутаться;
- плохая производительность;
- перемешивание различных сфер скрашивания досуга.

Данное приложение имеет красивый современный интерфейс, однако наличие большого количества мероприятий в одном месте отталкивает пользователя. Помимо этого, для комфортного использования приложения необходимо иметь мощное мобильное устройство, иначе пользователю придется ждать определенное количество времени для загрузки информации.

1.2.5 Мобильное приложение «Афиша 7»

Данное приложение также является агрегатором мероприятий выбранного города. В нем представлено множество категорий: кино, спорт, цирк, выставки, театр и прочие. Данные в приложении постоянно обновляются, поэтому пользователь может рассчитывать на актуальность представленной информации. На каждое мероприятие можно купить билет в один из представленных дней. Также в приложении есть фильтр по датам, то есть пользователь может выбрать только интересующие его дни. Интерфейс представлен на рисунке 5.

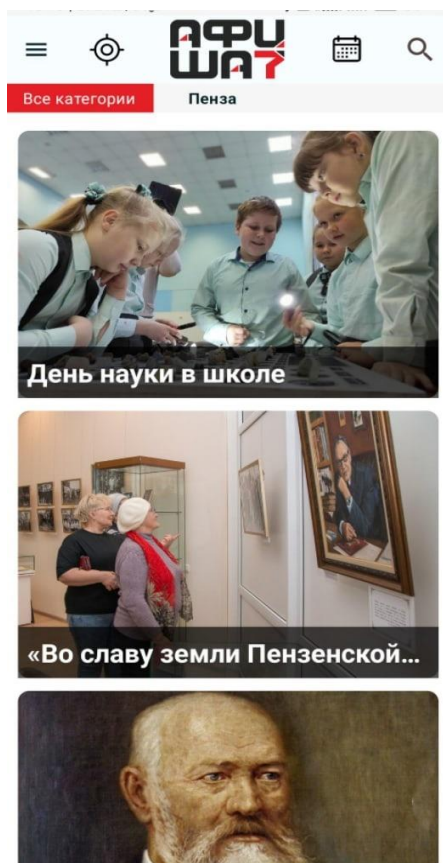


Рисунок 5 – Интерфейс приложения «Афиша 7»

Преимущества:

- большое количество информации;
- дружелюбный интерфейс;
- возможность покупки билетов с использованием стороннего сервиса;
- высокая производительность на слабых устройствах.

Недостатки:

- отсутствие личного кабинета;
- в информацию о спектакле включены все предстоящие мероприятия, из-за чего страница спектакля становится очень длинной;
- отсутствие информации об актерах.

С точки зрения агрегатора, данное приложение является удобным, однако оно не подходит для представления информации о конкретном театре.

1.3 Сравнительный анализ и критериальная оценка аналогов

Для составления объективной оценки оформим таблицу, в которой будут отражены критерии значимости. Данная информация представлена в таблице 1.

Таблица 1 – Выделение критериев значимости

№	Название критерия	Коэффициент значимости
	Функциональные возможности	
1	Просмотр афиши	1
2	Просмотр информации о спектакле (роли, актерский состав, краткое описание)	0.75
3	Возможность бронирования или покупки билета	1
4	Просмотр биографии актеров	0.5
5	Наличие форм регистрации и авторизации	1
	Технические возможности	
6	Возможность комфортного использования на слабом устройстве	0.75
	Необязательные функциональные возможности	
7	Бесплатность продукта	1
	Эргономика	
8	Наличие привлекательного интерфейса	0.75
9	Наличие интуитивно понятного интерфейса	1
10	Отсутствие лишней информации, в которой легко запутаться	0.75

Для оценки приведенных выше критериев используется пятибалльная система оценки, в которой значения изменяются от 0 до 2, где 0 – «полное несоответствие», 1 – «частичное соответствие», 2 – «полное соответствие» требованиям. В таблице 2 приведены более подробные характеристики оценки.

Таблица 2 – Шкала баллов

Балл	Характеристика оценки
0	функция (качество) отсутствует, реализовать ее невозможно
1	похожие, но отличные от требуемых, функции или качества системы находятся в стадии разработки
2	функция или качество безоговорочно реализовано

Для выставления оценок необходимо подробнее рассмотреть каждое приложение.

Оценка проводится по следующей формуле:

$$O_{\text{пп}} = \sum_{i=1}^n (Z_i \cdot K_i),$$

где $O_{пп}$ – интегральная оценка программного продукта;

n – количество критериев сравнения заявленных продуктов согласно таблице критериев;

Z_i – значение выполнения критерия (2 – выполнен, 0 – не выполнен);

K_i – коэффициент значимости критерия сравнения (от 0 до 1).

На основании полученной в ходе анализа информации составим таблицу, в которой будут наглядно видны преимущества и недостатки каждого продукта и общая их оценка. Результат отображен в таблице 3.

Таблица 3 – Критериальная оценка программных продуктов

№	Критерий	Название программного продукта				
		«Кукольный дом»	Пензенский драматический театр	«Мариинский театр»	«Афиша»	«Афиша 7»
1	Просмотр афиши	1	2	2	2	2
2	Просмотр информации о спектакле	1	2	2	2	1
3	Возможность бронирования/покупки билетов	1	1	2	2	2
4	Просмотр биографии актеров	0	2	1	2	0
5	Наличие форм регистрации и авторизации	0	0	2	2	0
6	Возможность комфортного использования на слабом устройстве	2	2	1	0	2
7	Бесплатность продукта	2	2	2	2	2
8	Наличие привлекательного интерфейса	0	1	2	2	1
9	Наличие интуитивно понятного интерфейса	2	2	2	1	2
10	Отсутствие лишней информации, в которой легко запутаться	2	1	2	0	0
	Интегральная оценка	9,75	12,5	14,75	12	11

На основании полученных оценок была построена столбчатая диаграмма, отображенная на рисунке 6.

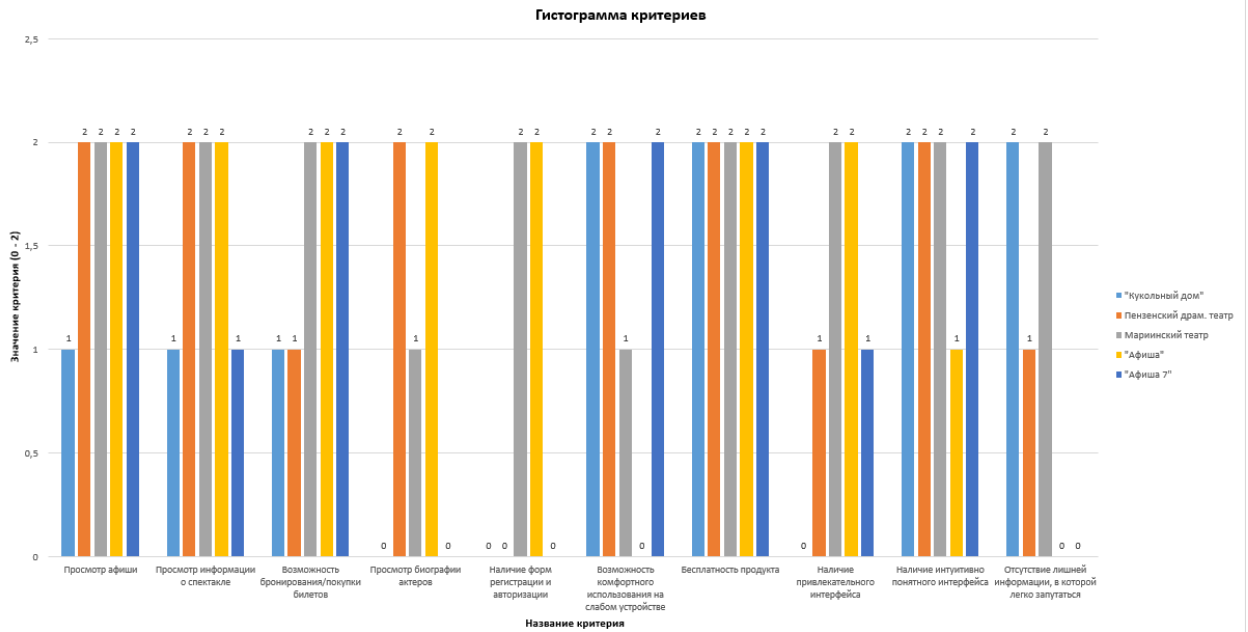


Рисунок 6 – Диаграмма оценок по критериям

Данная диаграмма представляет собой данные в виде разноцветных столбцов. Значение цвета указано в легенде.

На рисунке 7 отображена столбчатая диаграмма итоговых оценок.

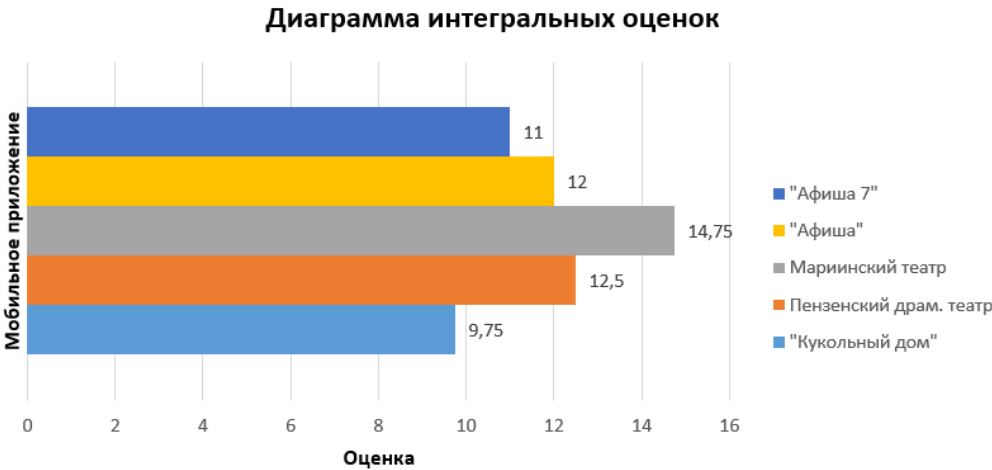


Рисунок 7 – Столбчатая диаграмма интегральных оценок

Подводя итог сравнения, можно сделать вывод, что ни один из сервисов не отвечает всем критериям в полной мере. Сайт кукольного театра отлично отвечает техническим требованиям, однако его дизайн давно не обновлялся, а также он имеет ряд недочетов с точки зрения корректности работы. Приложение «Афиша» удовлетворяет большинству функциональных и

нефункциональных требований, однако имеет множество лишней информации и требователен к мобильному устройству. Наибольшее количество баллов набрало приложение Мариинского театра, однако оно не имеет полную базу данных, и для просмотра информации об актерам приходится переходить на сайт.

1.4 Исследование вариантов использования

Варианты использования UML позволяют анализировать и проектировать систему с точки зрения ее функциональности и взаимодействия с акторами внешней среды [3].

Мобильное приложение предназначено для взаимодействия пользователя с театром в дистанционном формате. Система предоставит возможным посетителям полную информацию о спектаклях и актерам, а также позволит забронировать места на выбранное мероприятие.

На основе анализа аналогов была сформирована диаграмма вариантов использования, представленная на рисунке 8.

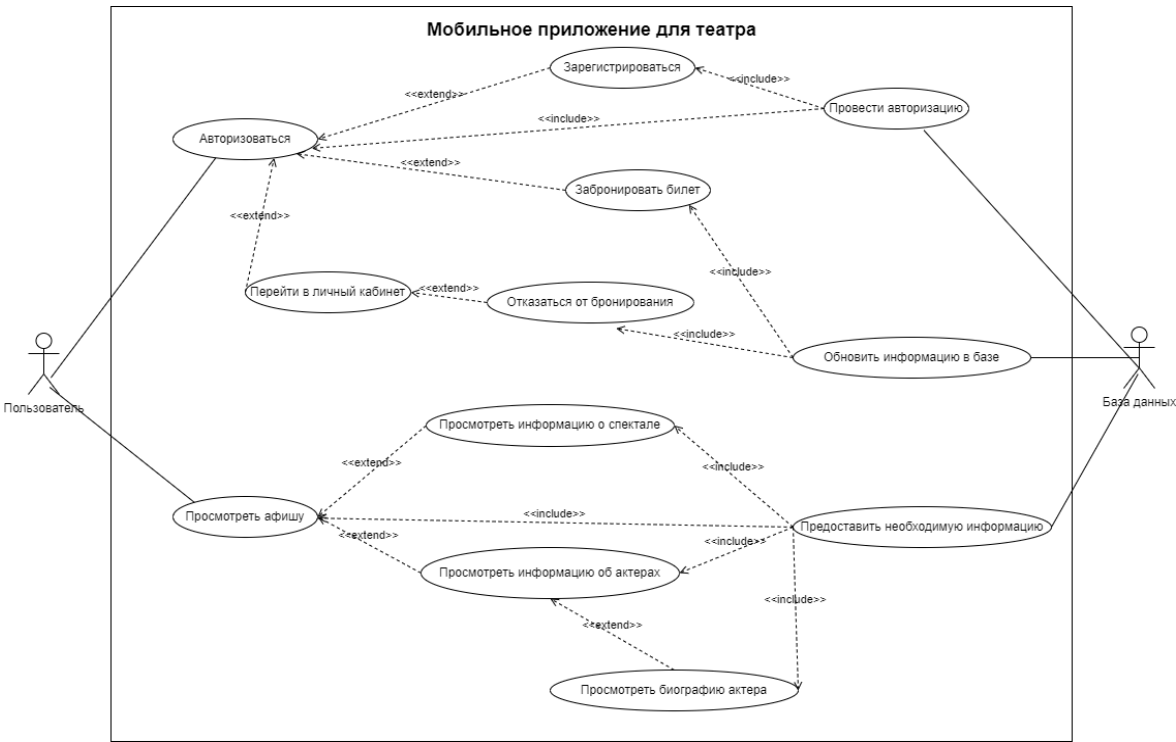


Рисунок 8 – Диаграмма вариантов использования для мобильного приложения кукольного театра

Для основных вариантов использования были составлены спецификации. На рисунках 9 – 12 представлены спецификации вариантов использования «Авторизоваться», «Просмотреть информацию о спектакле», «Просмотреть биографию выбранного актера», «Просмотреть информацию о театре».

Название: Авторизоваться
ID: 1
Краткое описание: Система предоставляет пользователю возможность войти в свою учетную запись.
Главные актеры: Пользователь, База данных.
Предусловие: База данных подключена к приложению, система загружена.
Основной поток: Система предоставляет пользователю форму для ввода данных. Пользователь вводит логин. Пользователь вводит пароль. Пользователь жмёт кнопку «Войти». Система проверяет наличие пользователя с введенными логином и паролем в базе. Система открывает пользователю доступ к основному функционалу приложения.
Постусловие: Пользователь вошел в свою учетную запись.

Рисунок 9 – Спецификация варианта использования «Авторизоваться»

Вариант использования «Авторизоваться» будет реализован добавлением в приложение страницы с авторизацией и регистрацией пользователя.

Название: Просмотреть информацию о спектакле
ID: 2
Краткое описание: Система предоставляет пользователю информацию о выбранном спектакле.
Главные актеры: Пользователь, База данных.
Предусловие: База данных подключена к приложению, система загружена.
Основной поток: Система предоставляет пользователю список спектаклей. Пользователь выбирает нужный спектакль. Система отображает краткую информацию о спектакле. Пользователь возвращается к общему списку спектаклей.
Постусловие: Пользователь просмотрел информацию о спектакле.

Рисунок 10 – Спецификация варианта использования «Просмотреть информацию о спектакле»

Для реализации варианта использования «Просмотреть информацию о спектакле» будет создана специальная страница.

Название: Просмотреть биографию выбранного актера
ID: 3
Краткое описание: Система предоставляет пользователю информацию о выбранном актере.
Главные актеры: Пользователь, База данных.
Предусловие: База данных подключена к приложению, система загружена.
Основной поток: Система предоставляет пользователю кликабельную ссылку на актеров, участвующих в спектакле. Пользователь выбирает интересующего актера. Система отображает форму просмотра информации об актере (биографии). Пользователь возвращается на страницу спектакля.
Постусловие: Пользователь просмотрел информацию об интересующем актере.

Рисунок 11 – Спецификация варианта использования «Посмотреть биографию выбранного актера»

Чтобы просмотреть биографию интересующего актера, пользователю необходимо нажать на его ФИО. После этого сформируется страница, на которой будет представлена информация.

Название: Просмотреть информацию о театре
ID: 4
Краткое описание: Система предоставляет пользователю информацию о театре.
Главные актеры: Пользователь, База данных.
Предусловие: База данных подключена к приложению, система загружена.
Основной поток: Система предоставляет пользователю пункт меню с названием «О театре». Пользователь жмет на пункт меню с названием «О театре». Система отображает форму с информацией о театре. Пользователь выходит из вкладки меню.
Постусловие: Пользователь просмотрел информацию о театре.

Рисунок 12 – Спецификация варианта использования «Посмотреть информацию о театре»

Для просмотра общей информации о театре в приложении предполагается реализация страницы «О театре», на которой будет представлена его история. Данная страница будет доступна в боковом меню как отдельная вкладка.

На рисунке 13 представлено описание варианта использования «Забронировать билет».

Название: Забронировать билет
ID: 5
Краткое описание: Система предоставляет пользователю возможность забронировать билет.
Главные актеры: Пользователь, База данных.
Предусловие: База данных подключена к приложению, система загружена.
Основной поток: Система предоставляет пользователю список спектаклей. Пользователь выбирает интересующий спектакль. Система отображает страницу спектакля с кнопкой «Забронировать билет». Пользователь жмет на кнопку «Забронировать билет». Система отображает свободные места. Пользователь выбирает место и подтверждает бронь. Система обновляет информацию в базе данных.
Постусловие: Пользователь забронировал билет, информация в базе данных обновилась.

Рисунок 13 – Спецификация варианта использования «Забронировать билет»

Для бронирования билета пользователю необходимо будет авторизоваться в системе, выбрать интересующее мероприятие, выбрать место и нажать на кнопку «Забронировать билет». Информация в базе обновится, и за пользователем закрепится место.

1.5 Анализ требований

Проанализировав полученные варианты использования и составив спецификации для основных из них, сформируем функциональные требования.

Важность выделенного набора требований объясняется несколькими причинами. Требования помогают гарантировать, что функциональность системы определяется пользователем, а не программистом. Если требования сформулированы ясно, пользователь может проанализировать и утвердить их [4].

Выделим функциональные требования к продукту.

Приложение театра должно позволять пользователю выполнять следующие функции:

- проводить авторизацию или регистрацию в системе;
- просматривать афишу спектаклей;
- проводить бронирование и отмену бронирования билетов;

- просматривать информацию о спектаклях;
- просматривать информацию об актерам в спектакле;
- просматривать биографию актеров.

Чтобы бронировать билеты должен иметься аккаунт. В противном случае доступ к этому функционалу получен не будет.

База данных должна быть разработана в среде MS SQL Server 2018. Взаимодействие с базой данных должно поддерживаться через приложение, запущенное на ПК. Приложение будет принимать запросы от пользователя и передавать их в SQL Server. ПК в данном случае будет выполнять роль сервера-посредника.

Для корректной работы приложения требуется мобильное устройство под управлением Android 8.0 и выше. Отдельных требований к мощности системы не предоставляется.

1.6 Постановка задачи

Разрабатываемое приложение должно предоставлять пользователям возможность просмотра интересующей информации о спектакле и соответствовать приведенным выше требованиям.

В таблице 4 описаны категории пользователей информационной системы.

Таблица 4 – Категория пользователей информационной системы

№	Категория пользователей	Наименование приложения	Фрагмент структуры предметной области	Информационные задачи
1	Администратор	Администрирование	Афиша; список актеров; репертуар	Пополнение базы данных спектаклям и актерами; обновление афиши
2	Пользователь	Театр	Афиша; репертуар; список актеров; информация о театре; бронирование билетов	Просмотр информации; бронирование билетов

Различные категории пользователей имеют различные права. Так, пользователю из категории «Администратор» доступны все привилегии

категории «Пользователь», а также управление данными. Категория «Пользователь», в свою очередь, не имеет доступа к изменению данных.

Для реализации разделения прав будет предусмотрен соответствующий атрибут в таблице «Пользователь».

2 Проектирование мобильного приложения театра

2.1 Архитектура приложения

Мобильное приложение театра состоит из нескольких компонентов: база данных; сервер базы данных; сервер приложения (API), построенный на архитектуре REST API; клиентская часть, содержащая реализацию интерфейса. Сервер приложения и клиентская часть должны быть разработаны с использованием шаблона проектирования MVC для обеспечения безопасности данных и простоты дальнейшей поддержки. Данный шаблон предполагает наличие трех уровней обработки информации: Model (модель, реализующая бизнес-логику), View (представление, то есть интерфейс) и Controller (посредник, отвечающий за передачу данных между моделью и интерфейсом) [5]. На рисунке 14 изображена схема взаимодействия мобильного приложения с REST API и сервером базы данных.

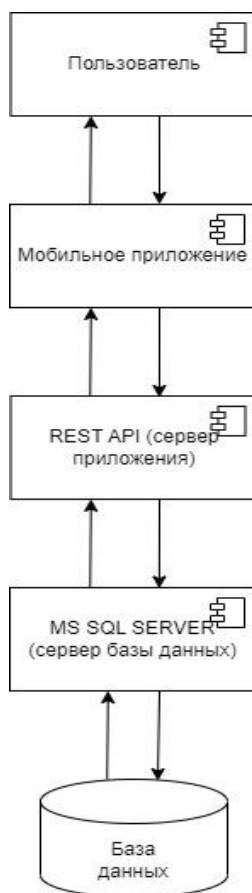


Рисунок 14 – Схема архитектуры разрабатываемого приложения

Как видно из представленного рисунка, каждый компонент обменивается данными с другим, за счет чего система реагирует на действия пользователя и отправляет необходимую информацию. Так, компонент «Мобильное приложение» обрабатывает действия компонента «Пользователь», после чего при необходимости посылает запросы к «REST API» с помощью HTTP-запросов. Сервер приложения (API) соединяется с компонентом «MS SQL SERVER» и отправляет SQL-запросы к компоненту «База данных». После обработки запроса сервер БД возвращает данные, передавая их в обратном порядке, и пользователю отображается запрашиваемая информация.

Мобильное приложение используется для отображения данных. Этот компонент преобразует действия пользователя в HTTP-запросы. Бизнес-логика содержится в сервере приложения.

2.2 Проектирование структур данных

Для хранения и организации наборов информации должна быть использована база данных. Система баз данных – это, по сути, не что иное, как компьютеризированная система хранения однотипных записей. Саму же базу данных можно рассматривать как подобие электронной картотеки, т. е. хранилище или контейнер для некоторого набора файлов данных, занесенных в компьютер [6].

2.2.1 Концептуальная модель базы данных

Концептуальная модель представляет собой описание основных сущностей (таблиц) и связей между ними независимо от ограничений, вытекающих из моделей данных, поддерживаемая той или иной СУБД [7].

Графическое отображение концептуальной модели базы данных, представляющее диаграмму вида «сущность-связь», представлено на рисунке 15.

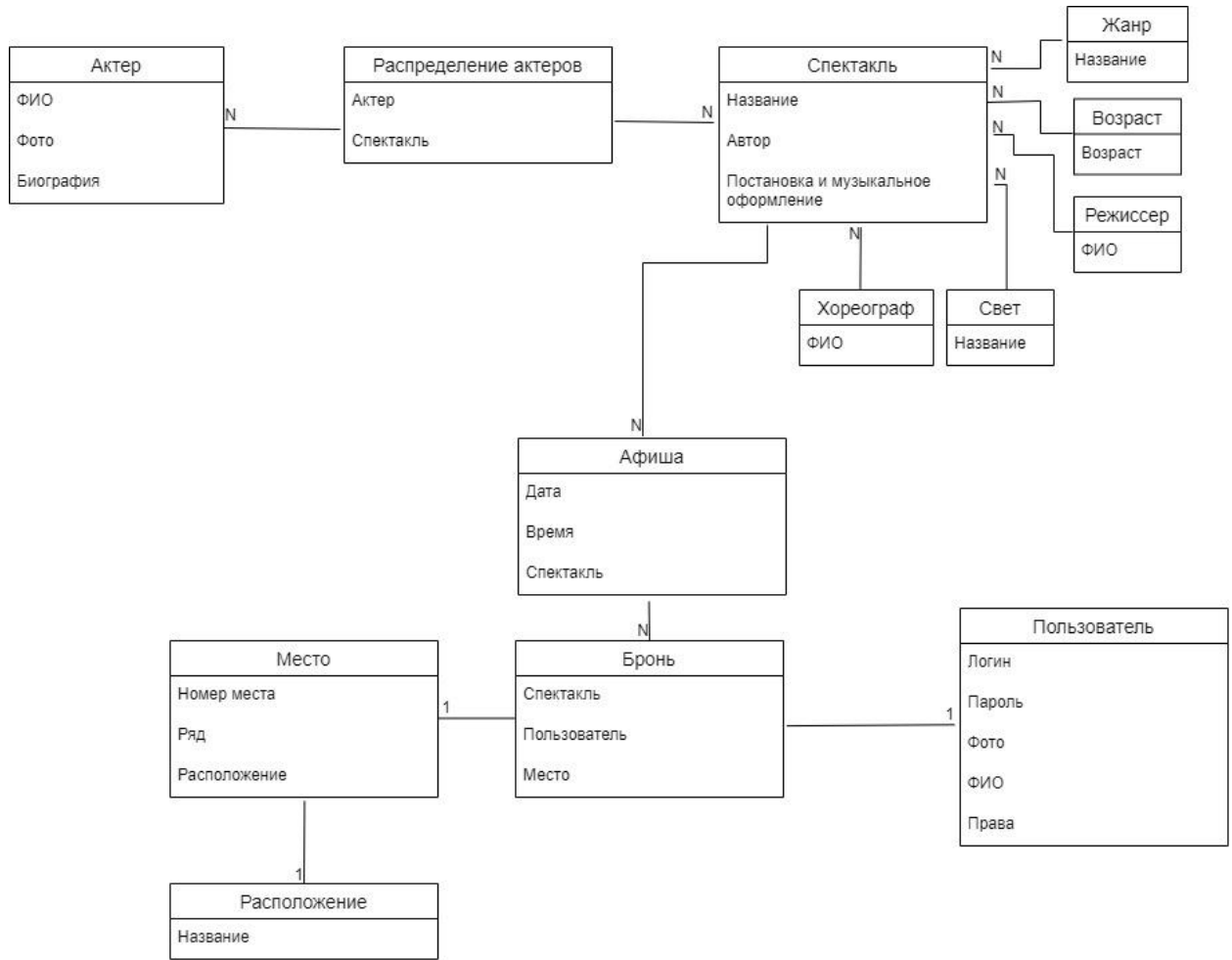


Рисунок 15 – Концептуальная модель базы данных приложения

Как видно из рисунка, база будет содержать тринадцать сущностей. В базе данных сущности представляют реальные или абстрактные объекты, которые имеют существенное значение для решаемой проблемы. Они описываются своими атрибутами и могут быть взаимосвязаны между собой через отношения [8]. Их описание представлено в таблице 5.

Таблица 5 – Описание сущностей

№	Наименование	Описание	Категория	Количество экземпляров	Режим обновления
1	Актер	Информация об актерах, принимающих участие в спектаклях	Стержневая	50	Раз в месяц или чаще
2	Спектакль	Основная информация о спектакле	Стержневая	100	Раз в месяц или реже
3	Жанр	Название жанра спектакля	Вспомогательная	5	Очень редко

Продолжение таблицы 5

№	Наименование	Описание	Категория	Количество экземпляров	Режим обновления
4	Возраст	Минимальный возраст для просмотра спектакля	Вспомогательная	5	Очень редко или никогда
5	Режиссер	ФИО режиссера-постановщика спектакля	Вспомогательная	10	Раз в месяц или реже
6	Свет	Информация о компании, отвечающей за освещение во время спектакля	Вспомогательная	10	Раз в полгода или реже
7	Хореограф	ФИО хореографа, отвечающего за постановку движений актеров	Вспомогательная	10	Раз в полгода или реже
8	Распределение актеров	Информация об участии актеров в спектаклях	Стержневая	150	Раз в месяц или чаще
9	Афиша	Информация о дате и времени проведения спектакля	Стержневая	200	Раз в день
10	Бронь	Информация о забронированных билетах на тот или иной спектакль	Стержневая	10000	Раз в час или чаще
11	Место	Информация о доступных местах в зале	Стержневая	200	Редко
12	Расположение	Информация о возможных расположениях мест	Вспомогательная	5	Очень редко или никогда
13	Пользователь	Информация о зарегистрированных в системе пользователях	Стержневая	1000	Несколько раз в час

Как видно из таблицы, база данных содержит семь стержневых таблиц и шесть вспомогательных. Вспомогательные таблицы используются для уточнения данных. Например, так как каждый спектакль имеет жанр, общее количество которых не превышает десяти, целесообразно выделить отдельную сущность с названием «Жанр».

2.2.2 Логическая модель базы данных

После выделения необходимых сущностей нужно построить логическую модель базы данных. Логическая модель отражает суть предметной области и позволяет указать первичные и внешние ключи.

Логическая модель базы данных является абстрактным представлением данных, основанным на структуре и отношениях между сущностями в предметной области, и предоставляет основу для создания физической реализации базы данных [9].

Графическое отображение логической модели базы данных в виде диаграммы вида «сущность-связь» представлено на рисунке 16.

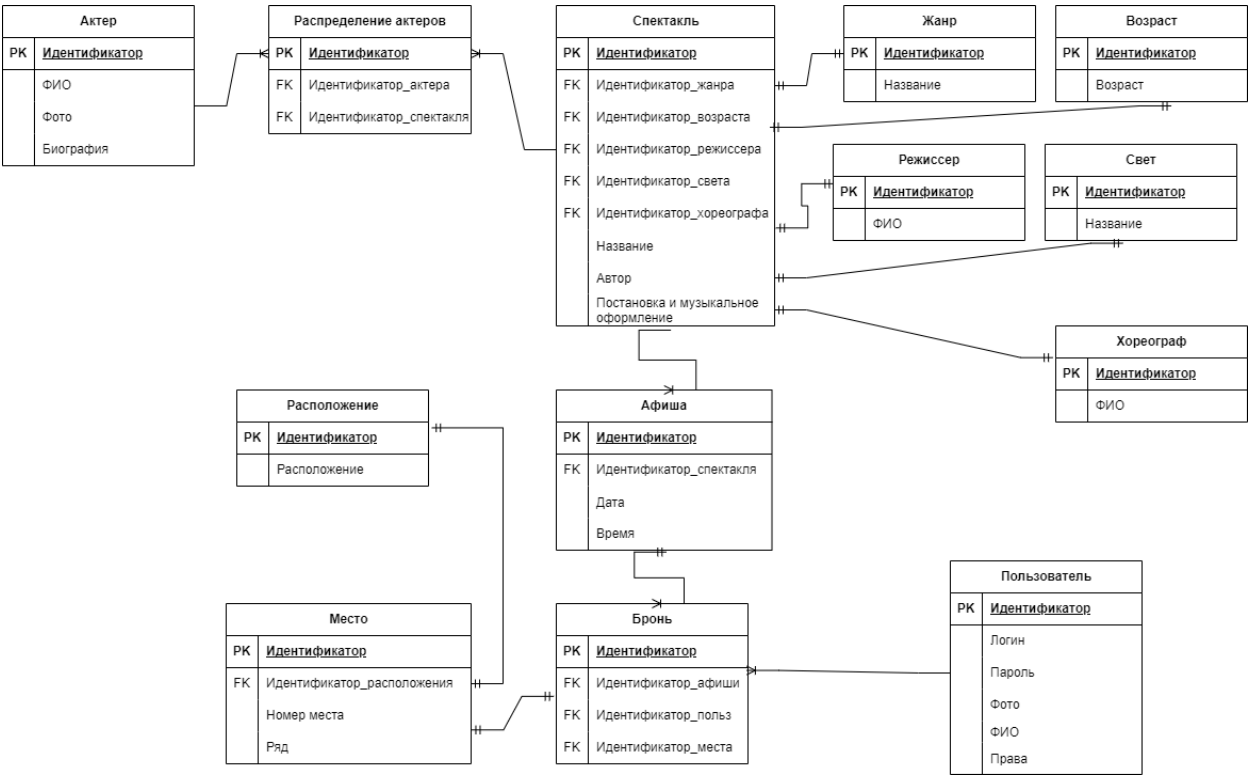


Рисунок 16 – Логическая модель базы данных

На данном этапе можно считать проектирование структур данных завершенным.

2.3 Проектирование программных средств

Для представления сущностей в ходе программирования необходимо выделить соответствующие классы. Для этого в проекте должен быть создан каталог «Сущности». Не каждая сущность нуждается в своей модели. Так, например, в программе будет представлена информация о местах, включая расположение, однако сама сущность «Расположение» рассмотрена не будет. Итоговая диаграмма классов представлена на рисунке 17.

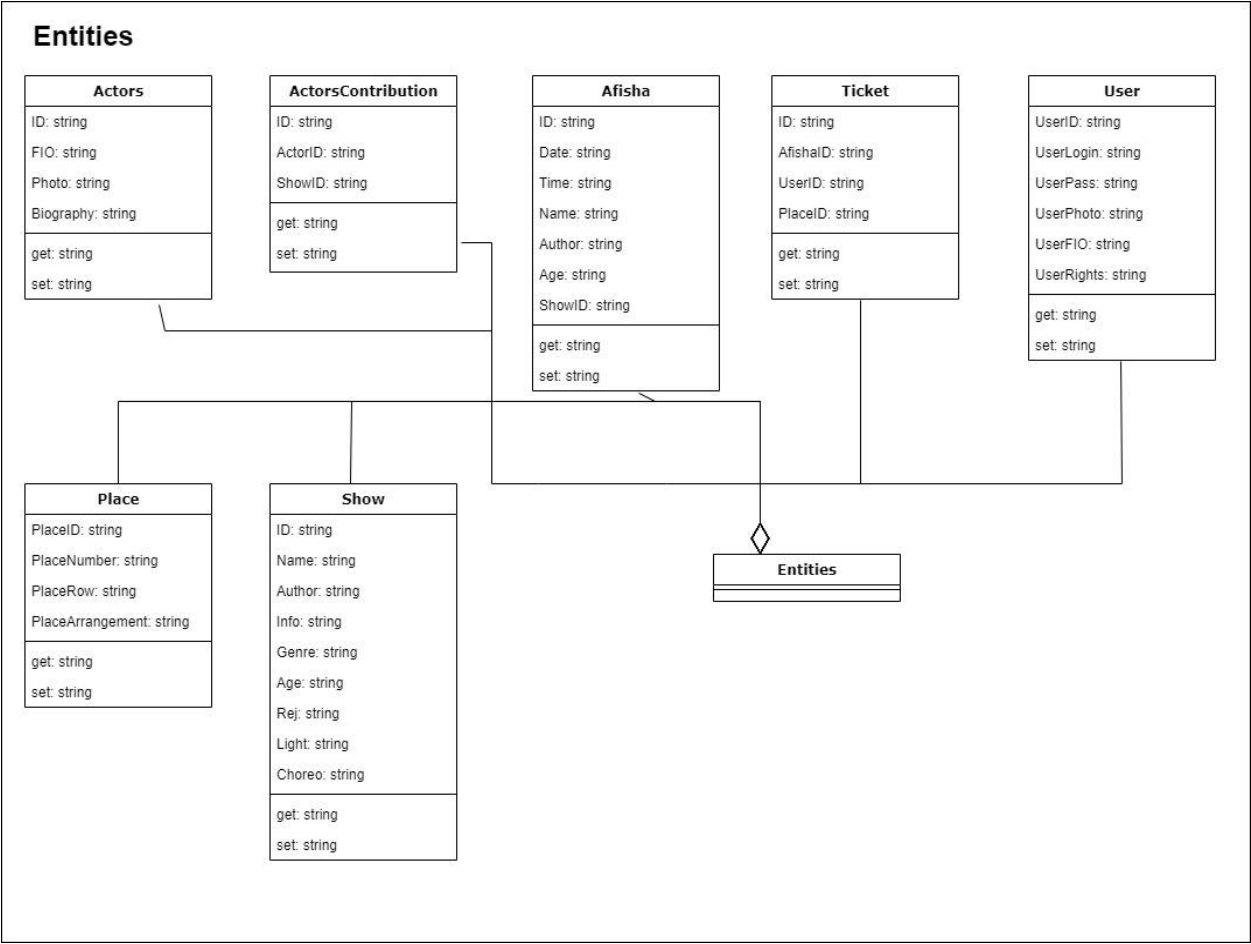


Рисунок 17 – Диаграмма классов приложения

Представленная диаграмма подходит как к архитектуре клиентского приложения, так и к архитектуре REST API-приложения, так как они получают и отправляют одни и те же данные.

Для варианта использования «Забронировать билет» была построена диаграмма деятельности. Инициатором действия является пользователь, поэтому его необходимо включить в диаграмму. Клиентское приложение обрабатывает все действия пользователя, такие как переход на страницу спектакля, переход на страницу бронирования билетов, выбор расположения, выбор места. В нем также реализована часть логики, а именно отображение информации о необходимости авторизации и сообщение о том, что выбранное место занято. Занесение информации в базу проходит в приложении-сервере. Полученный результат представлен на рисунке 18.

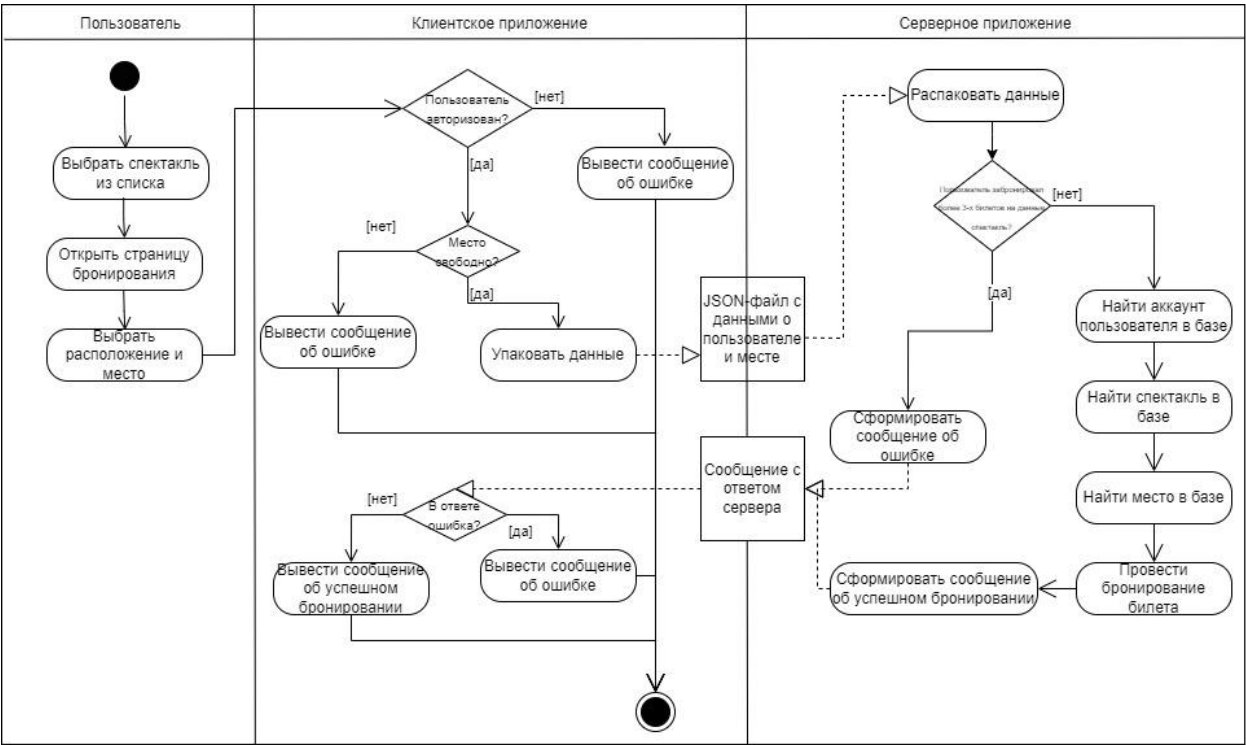


Рисунок 18 – Диаграмма деятельности мобильного приложения для театра

Диаграмма деятельности в нотации UML предоставляет возможность представить поведение системы в виде последовательности действий, позволяя легко визуализировать и анализировать процессы и взаимодействие между объектами в системе [10].

2.4 Проектирование интерфейса

Проектирование интерфейса – это задача создания не только красивого внешнего вида, но и логичной, интуитивно понятной структуры, которая позволит пользователям без труда находить нужную информацию и выполнять задачи [11].

Интерфейс приложения должен быть дружелюбным и емким. Также интерфейс должен реализовывать выделенные требования в полной мере. Должны присутствовать следующие страницы:

- страница регистрации и авторизации;
- страница личного кабинета;
- страница афиши;

- страница выбранного спектакля;
- страница с информацией о театре;
- страница бронирования.

Для правильной реализации интерфейса необходимо составить макеты. Макет страницы регистрации и авторизации представлен на рисунке 19.

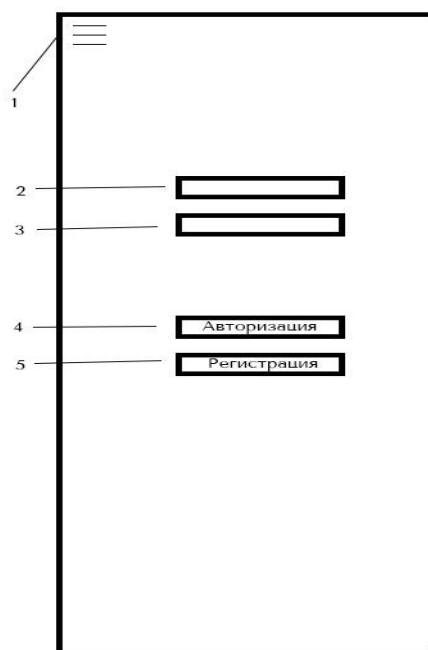


Рисунок 19 – Макет интерфейса страницы регистрации и авторизации

На рисунке выделены несколько пронумерованных элементов. Элемент 1 представляет собой иконку бокового меню, созданного для навигации между страницами. В последующих макетах данный элемент также будет присутствовать, однако по причине его рассмотрения здесь, он не будет выделен. Элемент 2 представляет собой поле для ввода логина пользователя, элемент 3 – поле для ввода пароля. Элемент 4 на рисунке представляет собой кнопку, которая используется для авторизации, а элемент 5 – кнопка для регистрации, если у пользователя нет аккаунта. В случае неверного ввода данных или попытки регистрации с занятым логином, система выдаст уведомление о неверном вводе.

На рисунке 20 представлен макет интерфейса страницы личного кабинета.

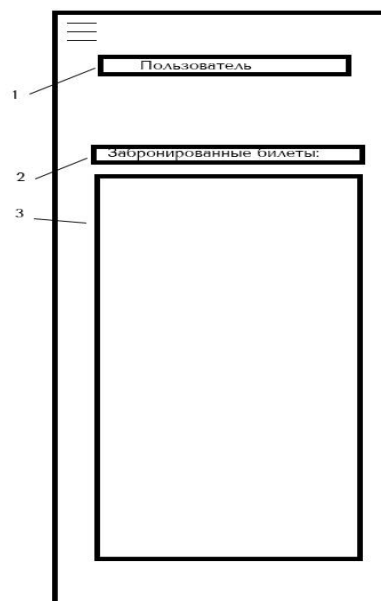


Рисунок 20 – Макет интерфейса страницы «Личный кабинет»

Элемент 1 на рисунке отражает логин пользователя. Элемент 2 является строкой, которая сообщает, что ниже представлены забронированные пользователем билеты. Элемент 3 представляет собой список, в котором содержатся забронированные данным пользователем билеты.

На рисунке 21 представлен макет интерфейса страницы афиши.

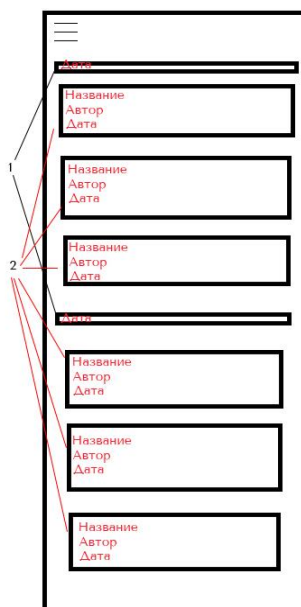


Рисунок 21 – Макет интерфейса страницы «Афиша»

Элемент 1 на рисунке представляет собой группировку по дате, то есть под ним будут выведены спектакли, проходящие в эту дату. Данный элемент

является кликабельным. При нажатии на него будут скрыты спектакли, проходящие в эту дату. Элемент 2 на рисунке – краткое отображение информации о спектакле, а именно: название, автор и дата проведения. Элемент 2 также является кликабельным, и при нажатии на него пользователю будет выведена страница спектакля.

Макет интерфейса страницы спектакля представлен на рисунке 22.

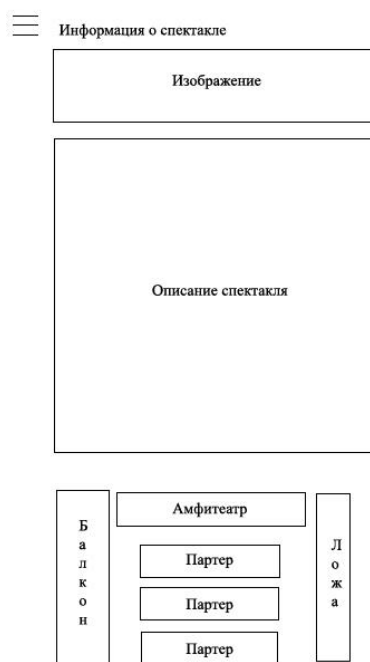


Рисунок 22 – Макет интерфейса страницы спектакля

Данная страница будет содержать название спектакля в верхней части. Первым элементом на странице является изображение, на котором отражена иллюстрация к спектаклю. На рисунке также содержится элемент «Описание спектакля», в котором представлена основная информация: автор, жанр, возраст, играющие актеры, информация о постановке. За описанием следует блок, который представляет собой расположение мест. Внутри него содержатся элементы, визуально соответствующие физическому расположению. При клике на какой-либо из них откроется страница бронирования, на которой будут отображены свободные и занятые места.

На рисунке 23 представлен макет интерфейса страницы с общей информацией о театре.

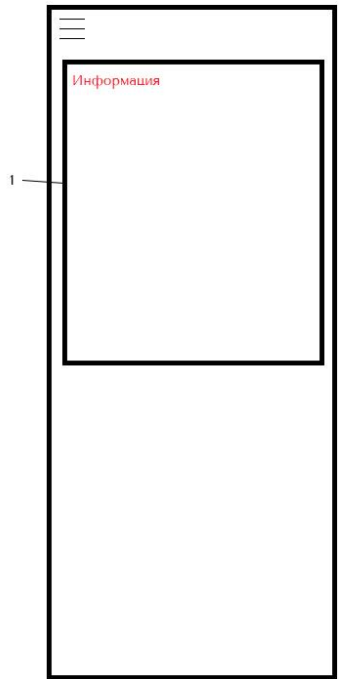


Рисунок 23 – Макет интерфейса страницы с информацией о театре

Данная страница имеет только один уникальный элемент – блок с текстом. Текст не содержится в базе данных, поэтому он будет написан напрямую в клиентском приложении.

На рисунке 24 представлен макет интерфейса страницы бронирования билетов.

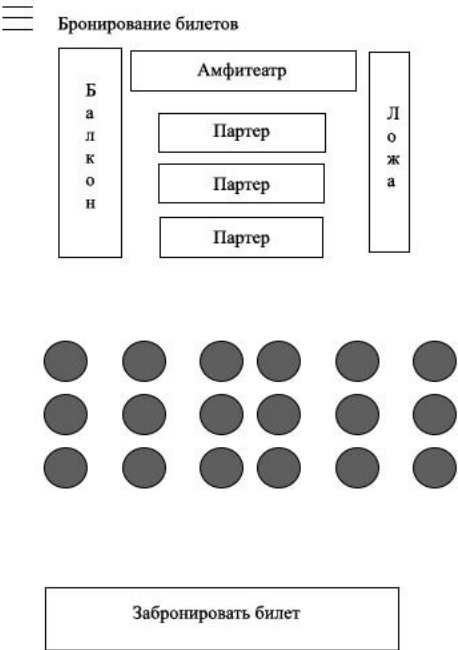


Рисунок 24 – Макет интерфейса страницы бронирования билетов

На странице бронирования билетов будет отображено выбранное расположение, а также наглядно представлены свободные и занятые места. Если место свободно, то оно будет подсвечено зеленым, если занято – красным. После выбора места будет выведена надпись, подтверждающая выбор. Для бронирования пользователю необходимо нажать на кнопку «Забронировать билет», и, в случае соблюдения всех условий (пользователь авторизован и забронировал на данный спектакль не более трех билетов, а также место не занято) выведется уведомление об успешном бронировании.

3 Реализация мобильного приложения для театра

3.1 Реализация серверной части приложения

3.1.1 Реализация базы данных

Начать реализацию базы данных следует с построения физической модели базы данных.

Физическая модель базы данных определяет, как данные будут храниться и организовываться на физическом уровне, включая аспекты, такие как файловые структуры, индексы и методы доступа к данным [12].

На основании построенной ранее логической модели были выделены характеристики атрибутов каждой сущности. Атрибуты и их описание представлены в приложении В.

Согласно составленным таблицам была построена физическая модель базы данных. Результат представлен на рисунке 25.

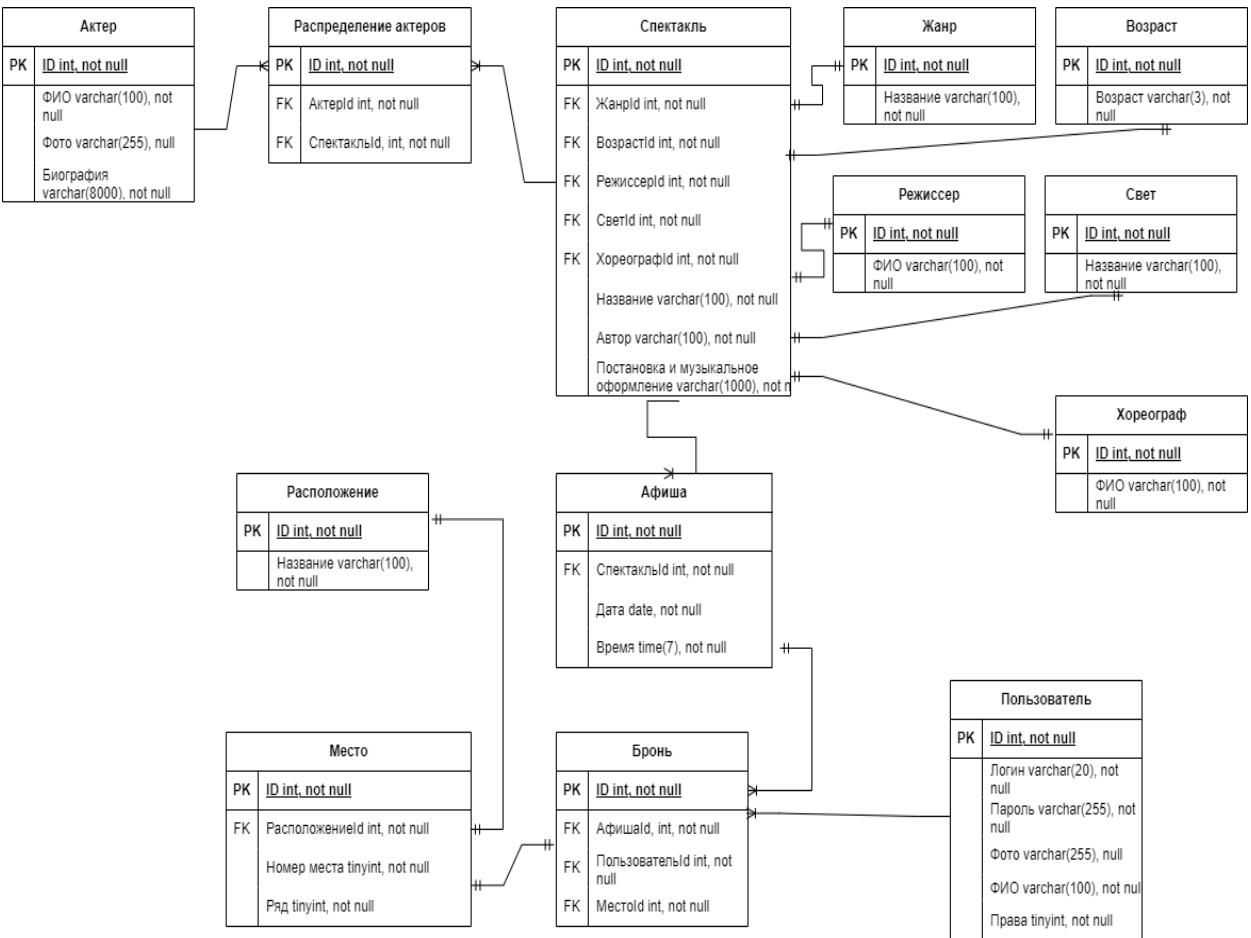


Рисунок 25 – Физическая модель базы данных приложения

Физическая модель отражает характеристики, свойственные выбранному серверу базы данных. Так, в MS SQL Server можно указывать размерность символьных атрибутов, но нельзя указать размер целочисленных.

3.1.2 Реализация серверных компонентов

Серверная часть приложения отвечает за обмен информацией между клиентом и базой данных. Она включает в себя классы, соответствующие задействуемым сущностям в базе данных.

Серверная часть реализована в среде MS Visual Studio 2022. Взаимодействие с базой данных происходит с использованием библиотеки System.Data.SqlClient.

На рисунке 26 представлена диаграмма классов приложения-сервера.

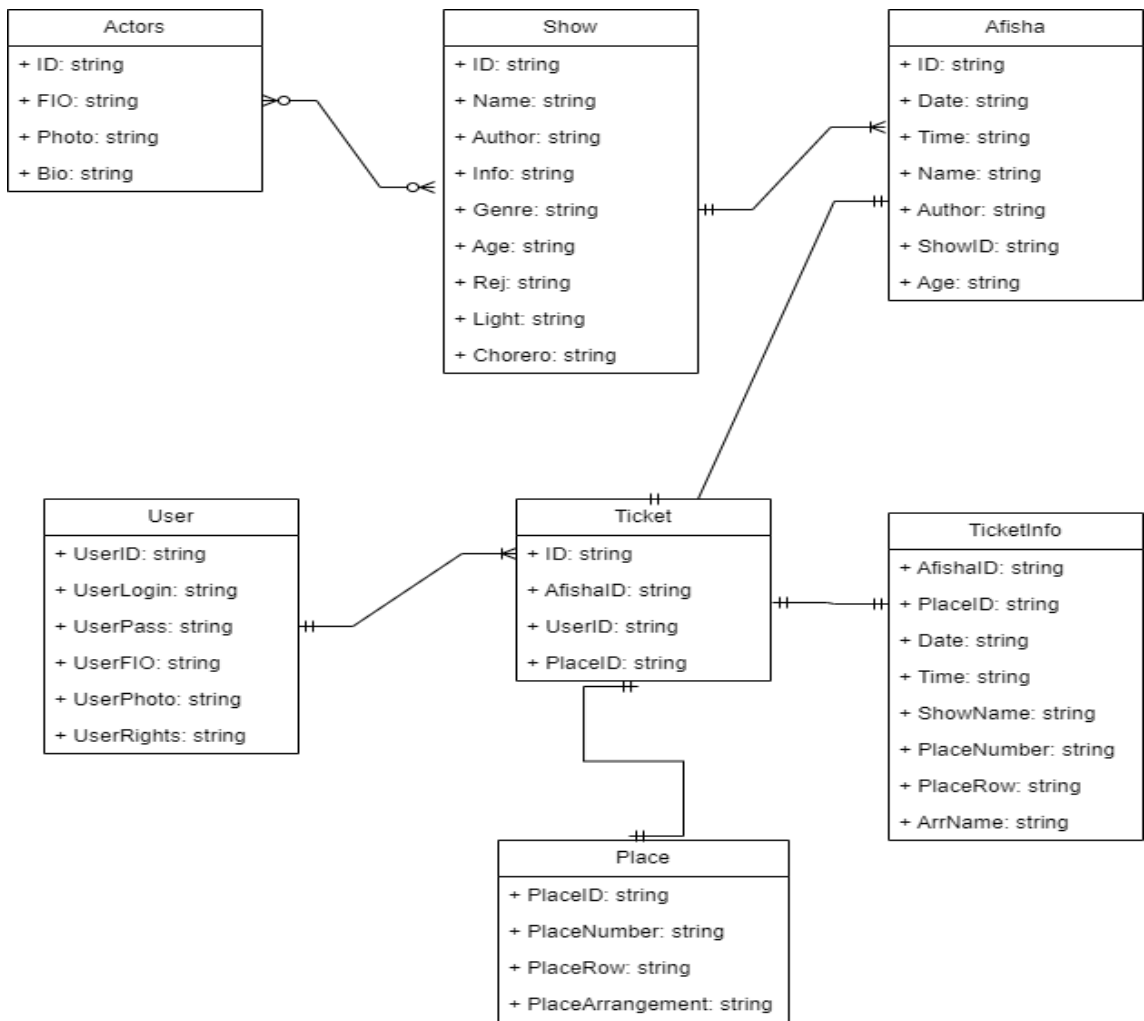


Рисунок 26 – Диаграмма классов приложения-сервера

Сервер приложения реализует архитектуру REST API с различными URL для обращения. Предоставляемые методы представлены в таблице 6.

Таблица 6 – Спецификация REST API приложения

№	Назначение сервиса	HTTP метод	URI сервиса	Класс контроллера
1	Получение афиши на ближайшие три дня	GET	/Afisha	ShowsController
2	Получение подробной информации о выбранном спектакле	GET	/Show/{ID}/	ShowsController
3	Получение списка актеров, участвующих в спектакле	GET	/Actors/{ID}	ShowsController
4	Получение списка всех мест по выбранному расположению	GET	/AllPlaces/{Arrangement}	BookingController
5	Получение информации о забронированных на спектакль билетах	POST	/BookedPlaces	BookingController
6	Получение выбранного пользователем места	POST	/Place	BookingController
7	Бронирование билета	POST	/Booking	BookingController
8	Отмена бронирования билета	POST	/RemoveBooking	BookingController
9	Регистрация пользователя в системе	POST	/Registration	AuthController
10	Авторизация пользователя в системе	POST	/Authorization	AuthController
11	Получение конкретного пользователя	POST	/GetUser	AuthController
12	Получение забронированных пользователем билетов	POST	/UserTickets	UserController

На рисунке 27 представлена диаграмма компонентов сервера мобильного приложения для театра.

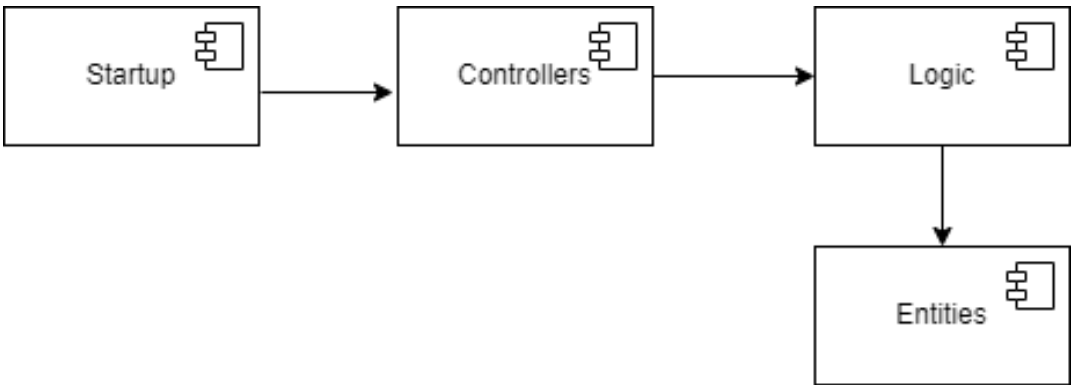


Рисунок 27 – Диаграмма компонентов сервера мобильного приложения для театра

Ниже представлено описание каждого компонента:

- компонент Startup – отвечает за логику запуска сервера в локальной сети; благодаря нему приложения-клиент может получать доступ по URL;
- компонент Controllers – промежуточный компонент, предназначенный для связи с логикой приложения; сюда включены контроллеры, перечисленные в таблице 6;
- компонент Logic – реализует логику приложения, а именно доступ к базе данных;
- компонент Entities – отвечает за представление сущностей в приложении; соответствует необходимым сущностям в базе данных.

Для корректной работы мобильного приложения в связке с сервером и базой данных, необходимо правильно развернуть все компоненты. Для этого была разработана диаграмма развертывания, представленная на рисунке 28.

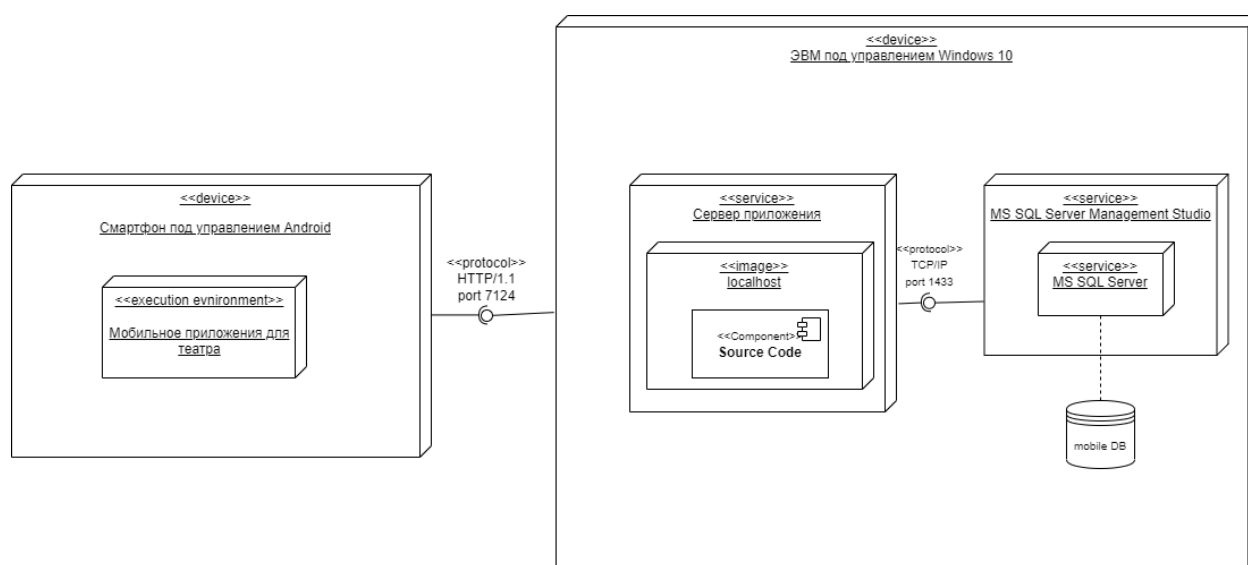


Рисунок 28 – Диаграмма развертывания приложения

Согласно диаграмме, приложение соответствует трехуровневой архитектуре.

Клиентское приложение запускается на устройстве пользователя. Для корректной работы клиент должен быть подключен к локальной сети.

Приложение-сервер запускается на персональном компьютере под управлением Windows 10 и связывается с сервером базы данных.

4 Контроль качества программного обеспечения

4.1 Выбор режима тестирования

Тестирование программного обеспечения – процесс анализа программного средства и сопутствующей документации с целью выявления дефектов и повышения качества продукта [13]

Существует множество методов тестирования, позволяющих проверить конкретные компоненты приложения. Для разработанного web-приложения было выбрано функциональное тестирование методом черного ящика, так как оно позволяет рассмотреть продукт от лица пользователя. Функциональное тестирование позволяет проверить поведение системы на соответствие ожидаемым результатам. Приложение не является крупным, поэтому данный метод вполне подходит.

Были рассмотрены следующие виды тестирования:

- функциональное тестирование;
- нефункциональное тестирование;
- интеграционное тестирование.

Функциональное тестирование – направлено на проверку того, какие функции приложения реализованы, и что они работают верным образом. Данный метод тестирования является одним из самых применяемых, так как позволяет наиболее полно проверить программу на соответствие требованиям [14].

Нефункциональное тестирование – вид тестирования, направленный на проверку нефункциональных особенностей приложения, таких как удобство использования, совместимость, производительность, безопасность и прочее [15].

Интеграционное тестирование – направлено на проверку взаимодействия между несколькими частями приложения. Этот вид тестирования позволяет проверить корректность взаимодействия частей ПО между собой [16].

На основании анализа различных методов было принято решение провести функциональное тестирование, так как оно во многом имитирует работу конечного пользователя. В ходе тестирования будет выявлено соответствие реализованных требований ожидаемым.

4.2 Тестирование серверной части мобильного приложения для театра

Для основных вариантов использования были проведены функциональные тесты. Тестирование проводилось с использованием программы Postman, позволяющей отправлять различные запросы по заданному URL-адресу. Назначение, описание, ожидаемые и полученные результаты приведены в таблице 7.

Таблица 7 – Функциональные тесты серверной части приложения

№ теста	Назначение	Описание	Входные данные	Ожидаемый результат	Актуальный результат
1	Проверка корректности выдачи информации о местах, доступных в заданном расположении	По заданному URL отправляется GET-запрос	URL: /AllPlaces/Партер	Список мест, представленный в виде json-файла	Пройден (листинг 1)
2	Проверка корректности авторизации	По заданному URL отправляется POST-запрос	URL: /Authorization Содержание json-файла: "userLogin": "admin", "userPass": "admin",	ID пользователя из базы данных	Пройден (рисунок 29)
3	Проверка регистрации с неверно указанными логином и паролем (данный пользователь уже зарегистрирован)	По заданному URL отправляется POST-запрос	URL: /Registration Содержание json-файла: "userID": null, "userLogin": "admin", "userPass": "admin", "userPhoto": null, "userFIO": null, "userRights": null	ID -1, так как такой пользователь уже существует	Пройден (рисунок 30)
4	Проверка корректности получения информации об актерах, участвующих в спектакле	По заданному URL отправляется GET-запрос	URL: /Actors/{ID} ID: 8	Информация об актерах, участвующих в данном спектакле	Пройден (листинг 2)

Продолжение таблицы 7

№ теста	Назначение	Описание	Входные данные	Ожидаемый результат	Актуальный результат
5	Проверка корректности получения подробной информации о спектакле	По заданному URL отправляется GET-запрос	URL: /Show/{ID} ID: 3	Список из пар ключ-значение, представляющий информацию о спектакле с ID 3	Пройден (листинг 3)

Листинг 1 – Часть полученного тела ответа, представляющая собой json-файл с местами с данным расположением

```
{
  "placeID": "1",
  "placeNumber": "1",
  "placeRow": "1",
  "placeArrangement": "Партер"
},
{
  "placeID": "2",
  "placeNumber": "2",
  "placeRow": "1",
  "placeArrangement": "Партер"
},
{
  "placeID": "3",
  "placeNumber": "3",
  "placeRow": "1",
  "placeArrangement": "Партер"
},
{
  "placeID": "4",
  "placeNumber": "4",
  "placeRow": "1",
  "placeArrangement": "Партер"
}
```

Согласно листингу 1, ответ соответствует ожидаемому. Это означает, что сервер корректно реализует необходимую логику.

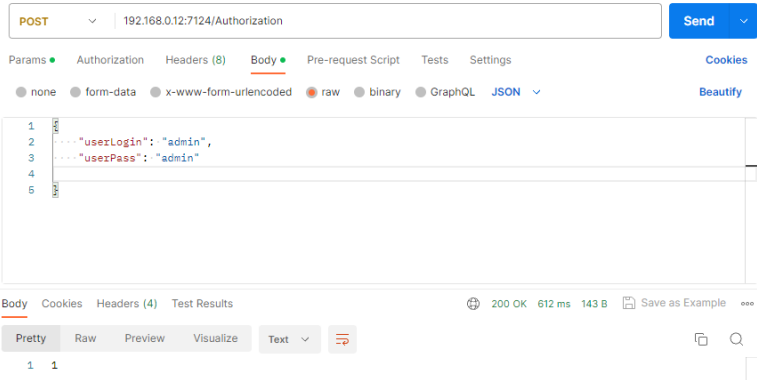


Рисунок 29 – Результат тестирования запроса на авторизацию (тест №2)

Согласно рисунку 29, в качестве ответа возвращен ID, равный «1». Данный ID соответствует пользователю с логином «admin» и паролем «admin», значит, сервер корректно реализует логику авторизации.

На рисунке 30 отображен результат теста №3.

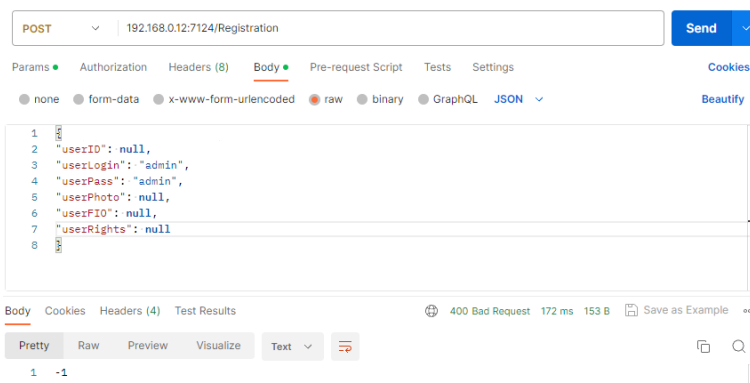


Рисунок 30 – Результат тестирования запроса на регистрацию (тест №3)

На нем видно, что в качестве ответа возвращен ID, равный -1. Такой ответ предусмотрен логикой сервера и означает, что пользователь с заданным логином уже зарегистрирован. Исходя из этого можно сделать вывод, что сервер корректно обрабатывает попытку регистрации с уже занятым логином.

В листинге 2 представлен результат теста №4.

Листинг 2 – Тело ответа, содержащее информацию об актерах

```
[
  {
    "id": "10",
    "fio": "Наталья Сизова",
    "photo": "actor.jpg",
    "bio": "Окончила Санкт-Петербургскую государственную академию
театрального искусства по специальности «артист театра кукол» в 2011 году
(Мастерская Р. Кудашова). Актриса театра кукол. Снимается в кино."
  }
]
```

Согласно листингу 2, в качестве ответа вернулась коллекция, состоящая из информации о Наталье Сизовой. Это верный ответ, так как в заданном спектакле принимает участие лишь один актер. На основании ответа можно сделать вывод, что сервер корректно реализует логику получения информации об актерах, играющих в заданном спектакле.

В листинге 3 представлен результат теста №5.

Листинг 3 – Тело ответа, содержащее информацию о спектакле

```
{
  "id": "3",
  "name": "Айболит",
  "author": "К. Чуковский",
  "info": "Художественное оформление – Евгений Петров; Музыкальное
оформление – Владимир Бычковский",
  "genre": "Сказка",
  "age": "3+",
  "rej": "Данила Александрович Чулков",
  "light": "Группа \"Сказочный свет\"",
  "choreo": "Хореов Алексей Графович"
}
```

Исходя из результата, полученного в ходе выполнения теста №5 и представленного в листинге 3, можно сделать вывод, что сервер корректно обрабатывает запросы на получение информации о спектакле.

По результатам проведения тестов можно увидеть, что сервер корректно обрабатывает все запросы, и в том числе предусматривает ответы в случае некорректного ввода.

4.3 Тестирование клиентской части мобильного приложения для театра

Для основных функций, представленных на диаграмме вариантов использования, были проведены функциональные тесты. Их перечень и описание представлены в таблице 8.

Таблица 8 – Перечень тестов клиентской части

№ теста	Назначение	Описание	Входные данные	Ожидаемый результат	Актуальный результат
1	Проверка корректности вывода афиши на предстоящие дни	Пользователь переходит на вкладку «Афиша»	—	Афиша, отсортированная по датам	Пройден (рисунок Г.1)
2	Проверка корректности вывода информации о спектакле	Пользователь выбирает спектакль из афиши	—	Информация о выбранном спектакле	Пройден (рисунок Г.2)
3	Проверка корректности вывода доступных мест в выбранном расположении	Пользователь переходит на страницу спектакля и выбирает расположение места	—	Список доступных мест	Пройден (рисунок Г.3)
4	Проверка реакции приложения на попытку бронирования билета неавторизованным пользователем	Пользователь не проходит авторизацию и пробует забронировать билет	—	Сообщение об ошибке, бронирования не происходит	Пройден (рисунок Г.4)

Продолжение таблицы 8

№ теста	Назначение	Описание	Входные данные	Ожидаемый результат	Актуальный результат
5	Проверка реакции системы на бронирование билета авторизованным пользователем	Пользователь проходит авторизацию и пробует забронировать билет	Логин: admin Пароль: admin	Сообщение об успешном бронировании, бронирование произошло	Пройден (рисунок Г.5)
6	Проверка реакции системы на попытку бронирования билета на занятое место	Пользователь проходит авторизацию и пробует забронировать билет, помеченный красным	Логин: admin Пароль: admin	Сообщение об ошибке, бронирования не происходит	Пройден (рисунок Г.6)
7	Проверка реакции системы на скрытие афиши в определенный день	Пользователь жмет на дату в списке афиши, после чего события, проходящие в этот день, должны быть скрыты	Дата: 05.06.2023	События, проходящие в данный день, будут скрыты	Пройден (рисунок Г.7)

Исходя из результатов, полученных в ходе тестирования, можно сделать вывод, что функциональные требования реализованы в полной мере, в том числе и обработка исключительных ситуаций.

4.4 Анализ качества кода

4.4.1 Измерение удобства поддержки встроенными средствами MS Visual Studio

Microsoft Visual Studio содержит набор инструментов, позволяющих проанализировать качество кода.

Перед проведением анализа следует выполнить рефакторинг, а именно убрать лишние директивы using и комментарии, так как они входят в общее число строк. Список подключенных директив в основном классе приложения до рефакторинга представлен на рисунке 31.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Collections.ObjectModel;
using System.Text.Json;
using System.Text.Json.Serialization;
using System.Net.Http.Json;
...
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
using mobile.Services;
using mobile.Entities;
using mobile.Helpers;
```

Рисунок 31 – Директивы using до рефакторинга

На рисунке 32 представлен измененный список подключаемых библиотек.

```
using System;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
using mobile.Services;
using mobile.Entities;
using mobile.Helpers;
```

Рисунок 32 – Директивы using после рефакторинга.

Подобная работа была проведена в каждом классе, поэтому итоговый список исходных строк кода будет оптимальным.

На рисунке 33 представлен результат вычисления метрик кода встроенными средствами MS Visual Studio.

Иерархия ▲	Индекс удобства поддержки	Сложность организации ци...	Глубина наследования	Взаимозависимость классов	Строки исходного кода	Строки исполняемого кода
TheatreWebAPI (Debug)	80	159	2	65	759	338
TheatreWebAPI	90	6	1	20	49	9
TheatreWebAPI.Controllers	64	32	2	36	241	105
TheatreWebAPI.Models	80	121	1	16	469	224

Рисунок 33 – Результат вычисления метрик кода приложения-сервера

Исходя из полученных значений, можно сделать вывод, что код является оптимальным.

4.4.2 Измерение устойчивости и абстрактности кода

Метрика устойчивости для компоненты отражает количественную меру работы, которую необходимо произвести при изменении других зависимых компонент.

Для вычисления позиционной устойчивости кода используется следующая формула:

$$I = \frac{Ce}{Ca + Ce},$$

где I – мера неустойчивости, изменяется в диапазоне от [0; 1];

Ce – количество классов вне данного компонента, зависящих от классов внутри компонента;

Ca – количество классов внутри данного компонента, зависящих от классов вне компонента.

В созданном приложении можно выделить две основные компоненты:

- компонента, реализующая логику обработки данных;
- компонента для обработки запросов клиентского приложения.

Для каждой компоненты была просчитана величина I – метрика неустойчивости:

$$I_1 = \frac{0}{0+0} = 0,$$

$$I_2 = \frac{0}{9+0} = 0.$$

Исходя из расчетов, оба компонента имеют максимальную меру устойчивости.

Для вычисления меры абстрактности используется следующая формула:

$$A = \frac{Na}{Nc},$$

где A – мера абстрактности, изменяется в диапазоне от 0 до 1, где 0 означает полное отсутствие абстрактных классов в компоненте, а 1 означает, что компонент не содержит ничего, кроме абстрактных классов;

Nc – количество классов в компоненте;

Na – количество абстрактных классов в том же компоненте.

Для всех компонентов приложения была вычислена мера абстрактности:

$$A_1 = \frac{2}{4} = 0.5,$$

$$A_2 = \frac{0}{4} = 0.$$

Согласно расчетам, компонента, реализующая логику обработки данных, наполовину состоит из абстрактных классов, а компонент для обработки запросов вовсе не содержит их. Это означает, что абстрактность компонентов приложения находится на высоком уровне, что облегчает дальнейшую поддержку приложения.

4.4.3 Главная последовательность

После вычисления меры устойчивости и абстрактности компоненты можно построить график, на вертикальной оси которого будет отложена мера абстрактности (A), а как горизонтальной оси – мера неустойчивости (I).

Компоненты, которые располагаются в верхнем левом углу или в нижнем правом являются максимально качественным. На практике часто случается, так что полученная точка лежит на диагонали между точкой $(1,0)$ и точкой $(0,1)$. Такая диагональ называется главная последовательность (рисунок 34).

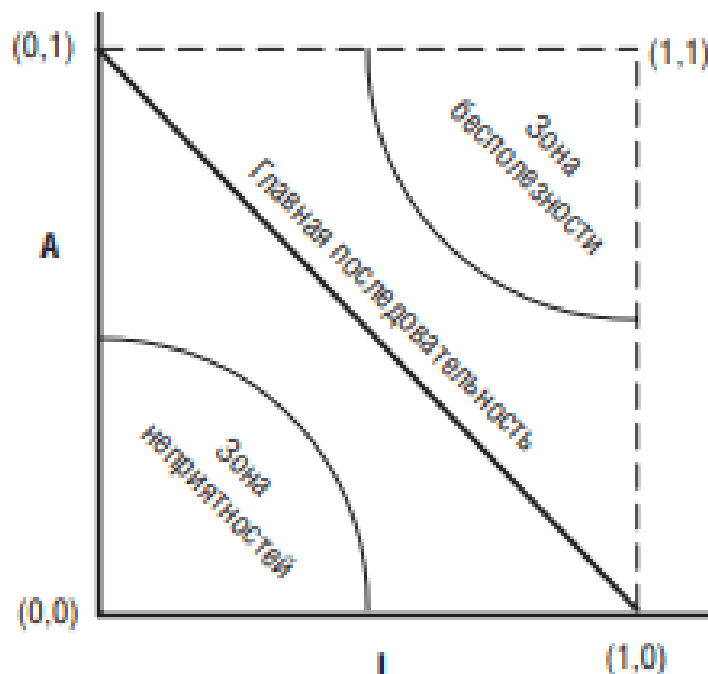


Рисунок 34 – Зоны исключения

Для вычисления количественной оценки используется формула вычисления расстояние до главной последовательности:

$$D = \left| \frac{(A + I - 1)}{\sqrt{2}} \right|,$$

где A – мера абстрактности;

I – мера устойчивости.

Было вычислено расстояние до главной последовательности:

$$D_1 = \left| \frac{0.5 + 0 - 1}{\sqrt{2}} \right| = 0.354,$$

$$D_2 = \left| \frac{0 + 0 - 1}{\sqrt{2}} \right| = 0.707.$$

Для логических компоненты был построен график (рисунок 35).

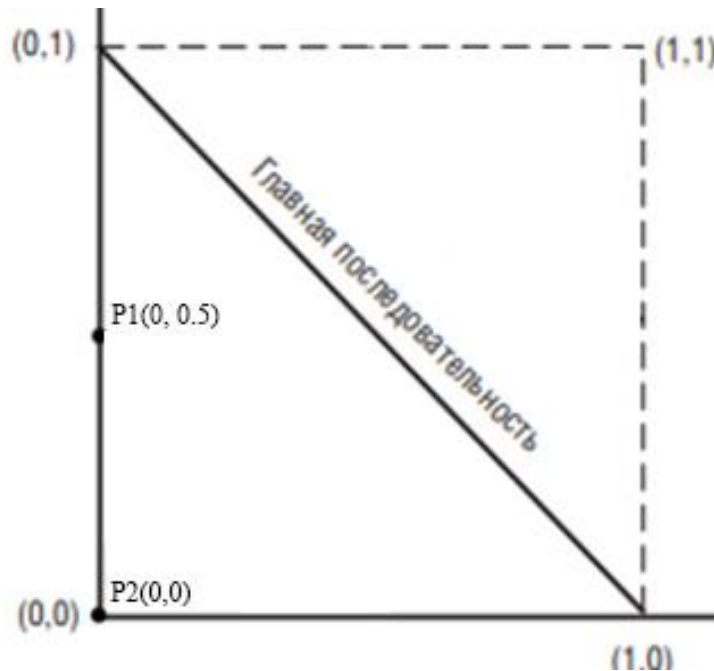


Рисунок 35 – Графическое отображение метрик на графике

Расстояние отношений метрик первой компоненты до главной последовательности достаточно близко к нулю, а значит ее отношение абстрактности к устойчивости является правильным.

Вторая компонента достаточно удалена от главной последовательности (точка P2). Такое расположение компоненты во многом связано с небольшим количеством классов, которые в ней находятся.

4.5 Планирование разработки и оценка бюджета

Планирование разработки и оценка бюджета была выполнена при помощи программы для управления проектами Microsoft Project [17]. Проект был начат 23 марта 2023 года.

При планировании разработки было выделено 36 задач. Список задач с общей информацией (длительность, дата начала, дата окончания и др.) представлены на рисунке 36.

	i	Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
1		Начало проекта	0 дней	Чт 23.03.23	Чт 23.03.23		
2		☐ Анализ требований	8 дней	Чт 23.03.23	Пн 03.04.23		
3		Изучение предметной	2 дней	Чт 23.03.23	Пт 24.03.23	1	Аналитик;Компьютер[
4		Анализ приложений ан	2 дней	Пн 27.03.23	Вт 28.03.23	3	Аналитик;Компьютер[
5		Исследование вариан	2 дней	Ср 29.03.23	Чт 30.03.23	4	Аналитик;Компьютер[
6		Анализ требований	1 день	Пт 31.03.23	Пт 31.03.23	5	Аналитик;Компьютер[
7		Поостановка задачи	1 день	Пн 03.04.23	Пн 03.04.23	6	Аналитик;Компьютер[
8		Анализ требований за	0 дней	Пн 03.04.23	Пн 03.04.23	7	
9		☐ Выбор средств	3 дней	Вт 04.04.23	Чт 06.04.23	8	
10		Выбор аппаратных сре	1 день	Вт 04.04.23	Вт 04.04.23	8	Программист;Компьют
11		Выбор программных с	2 дней	Ср 05.04.23	Чт 06.04.23	10	Программист;Компьют
12		Выбор средств завери	0 дней	Чт 06.04.23	Чт 06.04.23	11	
13		☐ Проектирование прил	11 дней	Пт 07.04.23	Пт 21.04.23	12	
14		Проектирование архит	2 дней	Пт 07.04.23	Пн 10.04.23	12	Программист;Компьют
15		☐ Проектирование стр	3 дней	Вт 11.04.23	Чт 13.04.23	14	
16		Составление концеп	1 день	Вт 11.04.23	Вт 11.04.23	14	Программист;Компьют
17		Составление логиче	1 день	Ср 12.04.23	Ср 12.04.23	16	Программист;Компьют
18		Составление физиче	1 день	Чт 13.04.23	Чт 13.04.23	17	Программист;Компьют
19		Проектирование прогр	3 дней	Пт 14.04.23	Вт 18.04.23	18	Программист;Компьют
20		Проектирование польз	3 дней	Ср 19.04.23	Пт 21.04.23	19	Дизайнер;Компьютер[
21		Проектирование завер	0 дней	Пт 21.04.23	Пт 21.04.23	20	
22		☐ Реализация приложен	18 дней	Пн 24.04.23	Пт 19.05.23	21	
23		Разработка базы данн	3 дней	Пн 24.04.23	Ср 26.04.23	21	Программист;Компьют
24		Наполнение базы данн	3 дней	Чт 27.04.23	Пн 01.05.23	23	Программист[50%];Ком
25		Разработка клиентской	5 дней	Вт 02.05.23	Ср 10.05.23	24	Программист;Компьют
26		Разработка серверной	2 дней	Чт 11.05.23	Пт 12.05.23	25	Программист;Компьют
27		Разработка пользоват	3 дней	Пн 15.05.23	Ср 17.05.23	26	Программист[50%];Диз
28		Отладка приложения	2 дней	Чт 18.05.23	Пт 19.05.23	27	Программист[50%];Тес
29		Реализация приложени	0 дней	Пт 19.05.23	Пт 19.05.23	28	
30		☐ Тестирование прилож	4 дней	Пн 22.05.23	Чт 25.05.23	29	
31		Выбор методов тести	1 день	Пн 22.05.23	Пн 22.05.23	29	Тестировщик;Интерне
32		Функциональное тести	1 день	Вт 23.05.23	Вт 23.05.23	31	Тестировщик;Мобильн
33		Модульное тестирова	1 день	Ср 24.05.23	Ср 24.05.23	32	Тестировщик;Мобильн
34		UI-тестирование	1 день	Чт 25.05.23	Чт 25.05.23	33	Тестировщик;Мобильн
35		Тестирование завери	0 дней	Чт 25.05.23	Чт 25.05.23	34	
36		Конец проекта	0 дней	Чт 25.05.23	Чт 25.05.23	35	

Рисунок 36– Список задач проектирования

Для того чтобы успешно выполнить проект, необходимо использовать определенные ресурсы. Список ресурсов, их тип, краткое название, максимальное количество единиц, стандартная ставка, сверхурочная ставка, назначение и некоторые другие параметры, не представляющие особой важности, представлены на рисунке 37.

	1	Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочных	Затраты на исполн.	Начисление	Базовый календарь
1		Программист	Трудовой		П		100%	50 000,00 Р/мес	500,00 Р/ч	0,00 Р	Пропорциональное	Стандартный
2		Аналитик	Трудовой		А		100%	30 000,00 Р/мес	0,00 Р/ч	0,00 Р	Пропорциональное	Стандартный
3		Тестировщик	Трудовой		Т		100%	30 000,00 Р/мес	0,00 Р/ч	0,00 Р	Пропорциональное	Стандартный
4		Дизайнер	Трудовой		Д		100%	45 000,00 Р/мес	0,00 Р/ч	0,00 Р	Пропорциональное	Стандартный
5		Компьютер	Материальный		К			0,00 Р		0,00 Р	Пропорциональное	
6		Мобильное устройство	Материальный		М			0,00 Р		0,00 Р	Пропорциональное	
7		Интернет	Затраты		И						Пропорциональное	

Рисунок 37– Список ресурсов проекта

После назначения задач была получена полноценная диаграмма Ганта (рисунок 38). На ней наглядно видно связи между задачами, используемые ресурсы и даты начала и окончания.

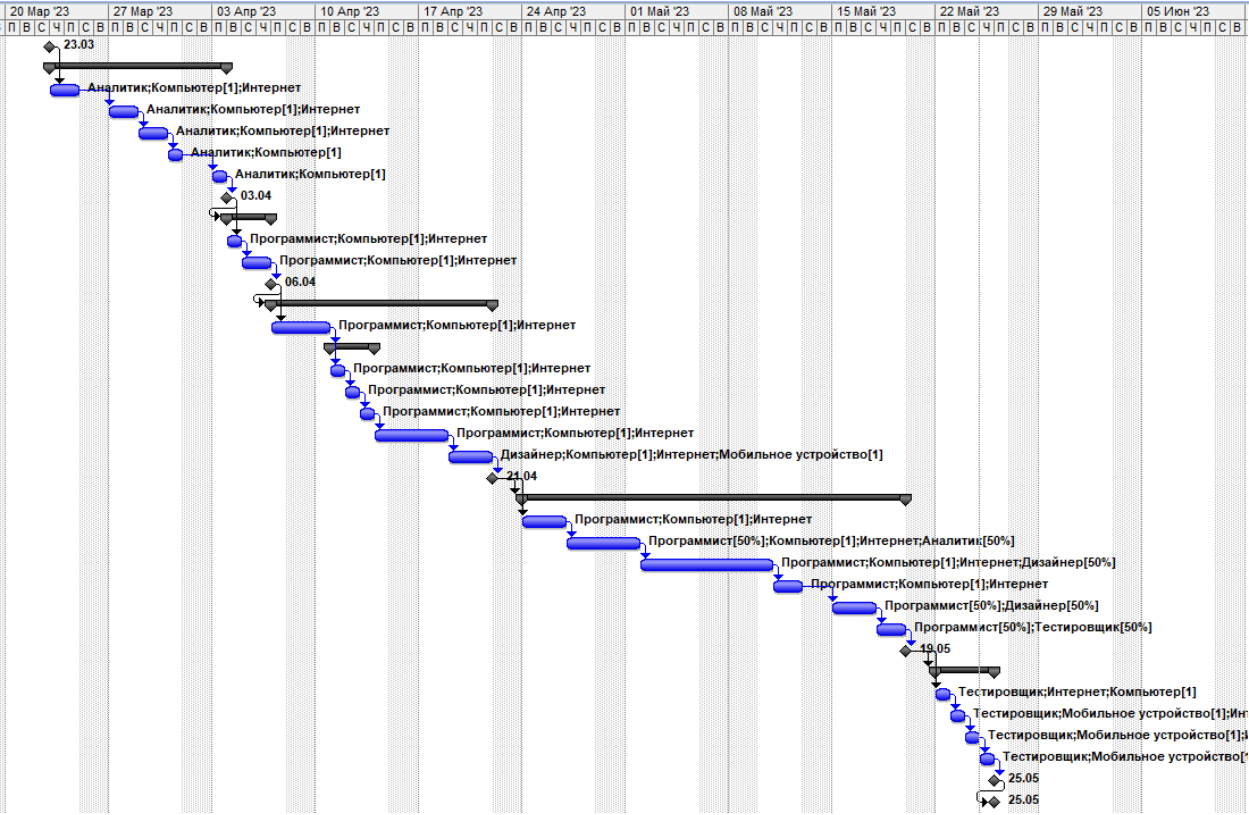


Рисунок 38 – Диаграмма Ганта проекта

После составления проекта, определения дат и используемых ресурсов, был построен отчет, представленный на рисунке 39. Отчет отражает общую информацию о проекте, такую как дату начала и дату окончания, трудозатраты на разработку по часам, материальные затраты на ресурсы и общее количество ресурсов.

<u>Даты</u>			
Начало:	Чт 23.03.23	Окончание:	Чт 25.05.23
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала:	0 дней	Отклон. окончания:	0 дней
<u>Длительность</u>			
Текущий план:	44 дней	Оставш ася:	44 дней
Базовый план:	0 дней?	Фактическая:	0 дней
Отклонение:	44 дней	Процент завершения:	0%
<u>Трудозатраты</u>			
Текущий план:	372 ч	Оставш иеся:	372 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	372 ч	Процент завершения:	0%
<u>Затраты</u>			
Текущий план:	100 000,00 Р	Оставш иеся:	100 000,00 Р
Базовый план:	0,00 Р	Фактические:	0,00 Р
Отклонение:	100 000,00 Р		
<u>Состояние задачи</u>		<u>Состояние ресурса</u>	
Еще не начатые задачи	36	Трудовые ресурсы:	5
Выполняющиеся задачи	0	Трудовые ресурсы с прев. дост	0
Завершенные задачи:	0	Материальные ресурсы:	2
Итого задач:	36	Итого ресурсов:	7

Рисунок 39 – Отчет по проекту

На основании этого отчета и времени выполнения проекта можно сделать вывод о целесообразности разработки.

Заключение

Результатом выполнения выпускной квалификационной работы является разработанное и отлаженное мобильное приложение для театра.

В ходе выполнения этапа анализа требований была обоснована актуальность разрабатываемого приложения. Рассмотрены приложения-аналоги и осуществлен их обзор, выделены преимущества и недостатки. На основании этих критериев проведена оценка. Результатом стало получение интегральных оценок каждого аналога. Также выявлены функциональных требования и проведен их анализ.

В ходе выполнения этапа проектирования приложения определена архитектура приложения. Принято решение использовать клиент-серверное взаимодействие между мобильным приложением и базой данных. Для обмена данными между ними спроектировано дополнительное приложение, использующее шаблон REST API. Для его реализации, а также реализации приложения-клиента, выбран шаблон MVC. Также спроектированы структуры данных и созданы макеты интерфейса. Проектирование структур данных включило в себя построение концептуальной, логической и физической моделей данных, нацеленных на приложение MS SQL Server 2018.

В ходе выполнения этапа реализации была разработана база данных и сервер приложения, соответствующий архитектуре REST API. Реализация как приложения-сервера, так и мобильного приложения проходила в среде разработки MS Visual Studio 2022 на языке C# с использованием различных компонентов, в том числе фреймворка Xamarin Forms. Взаимодействие с базой данных осуществлено с использованием библиотеки SqlClient.

Приложение было протестировано двумя видами тестирования. Серверная часть проверена с помощью модульного тестирования, а клиентская – с помощью функционального тестирования. В результате все тесты были пройдены.

Был составлен план и проведена оценка бюджета. В ходе выполнения данного этапа выявлены временные и финансовые затраты на разработку и необходимые ресурсы. Результатом стала диаграмма Ганта и отчет.

Таким образом, поставленная по реализации мобильного приложения для театра задача выполнена в полной мере.

Список использованных источников

1. Опрос: доля россиян, посещающих театры, составила 48% в 2022 году. // ВЦИОМ [Электронный ресурс] URL: https://culture.gov.ru/press/news/vtsiom_chislo_poseshchayushchikh_teatry_rossiyan_vyroslo_pochti_vdvoe_za_poslednie_10 лет/ (дата обращения: 22.05.2023).
2. Исследование: почти 70% россиян используют смартфон для выхода в интернет // ТАСС [Электронный ресурс]. – 2022. – 21 сентября. – URL: <https://tass.ru/obschestvo/9508331> (дата обращения: 22.05.2023).
3. Ричи, Г. UML. Руководство по моделированию, 3-е издание / Г. Ричи; пер. с англ.: А. Слинкин. – М.: Издательский дом "Питер", 2010. – 412 с.
4. Макконнелл, С. Совершенный код, 2-е издание / С. Макконнелл; пер. с англ.: А. Слинкин. – М.: Издательский дом "Русская редакция", 2018. – 887 с.
5. Гамма, Э. Приемы объектно-ориентированного программирования / Э. Гамма; пер. с англ.: А. Слинкин. – М.: Издательский дом "Питер", 2001. – 343 с.
6. Дейт, К. Дж. Введение в системы баз данных, 8-е издание / К. Дж. Дейт; пер. с англ. – М.: Издательский дом "Вильямс", 2005. – 1328 с.
7. Костюкова, А. П. Базы данных: общий курс: учебное пособие / А. П. Костюкова. – Москва: Акад. естествознания, 2018. – Текст: непосредственный. Ч. 1. – 81 с.
8. Зильбершнац, А. Database System Concepts, 6-е издание / А. Зильбершнац, Г. Корт, С. Сударшан. – Издательство "Connect Learn Succeed", 2010. – 1349 с.
9. Эрнандес, М. Database design for Mere Mortals: A Hands-On guide to Relational database design / М. Эрнандес. – М.: Издательский дом "Addison-Wesley", 2013. – 672 с.
10. Буч, Г. Язык UML. Руководство пользователя, 2-е издание / Г. Буч, Дж. Рамбо, И. Якобсон; пер. с англ.: Н. Мухин. – М.: Издательский дом "ДМК Пресс", 2006. – 496 с.

11. Тидвелл, Дж. Дизайн интерфейса: Лучшие практики для взаимодействия с пользователем / Дж. Тидвелл; пер. с англ.: О. Березкина. – М.: Издательский дом "Питер", 2012. – 352 с.
12. Конноли, Т. Введение в системы базы данных, 8-е издание / Т. Конноли. – М.: Издательский дом "Pearson Education Limited", 2008. – 1272 с.
13. Куликов, С. Тестирование программного обеспечения. Базовый курс, 3-е издание / С. Куликов. – М.: Издательство МГТУ им. Н. Э. Баумана, 2020. – 301 с.
14. Функциональное тестирование программного обеспечения // Лаборатория программирования [Электронный ресурс]. URL: <https://daglab.ru/funkcionalnoe-testirovanie-programmnogo-obespechenija/> (дата обращения 25.05.2023).
15. Нефункциональное тестирование // ИксБи Софтваре [Электронный ресурс]. URL: <https://xbsoftware.ru/testirovanie-po/polnijtsykl/nefunctional-test/> (дата обращения 25.05.2023).
16. Для чего нужно интеграционное тестирование? // Сообщество IT-специалистов [Электронный ресурс]. URL: <https://habr.com/ru/post/556002/> (дата обращения 25.05.2023).
17. Управление проектами в Microsoft Project: учеб.-метод. пособие для студентов по направлению специальности «Информационные системы и технологии (издательско-полиграфический комплекс)» / Т. П. Брусенцова, В. В. Смелов. – Минск: БГТУ, 2011. – 160 с.

ПРИЛОЖЕНИЕ А

ГЛОССАРИЙ

(справочное)

API (Application Programming Interface) – набор готовых классов, функций, процедур, констант и структур, предоставляемых приложением или операционной системой для использования во внешних программных продуктах.

REST API – архитектурный стиль взаимодействия компонентов распределённого приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой гипермедиа-системы.

SQL (Structured Query Language) – декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

HTTP (HyperText Transfer Protocol) – протокол прикладного уровня передачи данных предназначенный для передачи гипертекстовых документов.

UML (Unified Modeling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

СУБД – система управления базами данных.

MS SQL Server – свободно распространяемый сервер баз данных, разработанный компанией Microsoft.

MS SQL Server Management Studio – свободно распространяемая СУБД, разработанная компанией Microsoft, предназначенная для взаимодействия с MS SQL Server.

PK – Primary Key (первичный ключ), в реляционной модели данных один из потенциальных ключей отношения, выбранный в качестве основного ключа (или ключа по умолчанию).

FK – Foreign Key (внешний ключ), понятие теории реляционных баз данных, относящееся к ограничениям целостности базы данных.

JSON – JavaScript Object Notation, текстовый формат, полностью независимый от языка реализации.

ПРИЛОЖЕНИЕ Б
ЛИСТИНГ ПРОГРАММЫ
(рекомендуемое)

Серверная часть.

Листинг Б.1 – Файл Actors.cs

```
namespace TheatreWebAPI.Models
{
    public class Actors
    {
        public string ID { get; set; }
        public string FIO { get; set; }
        public string Photo { get; set; }
        public string Bio { get; set; }
    }
}
```

Листинг Б.2 – Файл Afisha.cs

```
namespace TheatreWebAPI.Models
{
    public class Afisha
    {
        public string ID { get; set; }
        public string Date { get; set; }
        public string Time { get; set; }
        public string Name { get; set; }
        public string Author { get; set; }
        public string ShowID { get; set; }
        public string Age { get; set; }
    }
}
```

Листинг Б.3 – Файл Place.cs

```
namespace TheatreWebAPI.Models
{
    public class Place
    {
        public string PlaceID { get; set; }
        public string PlaceNumber { get; set; }
        public string PlaceRow { get; set; }
        public string PlaceArrangement { get; set; }
    }
}
```

Листинг Б.4 – Файл Show.cs

```
namespace TheatreWebAPI.Models
{
    public class Show
    {
        public string ID { get; set; }
        public string Name { get; set; }
        public string Author { get; set; }
        public string Info { get; set; }
        public string Genre { get; set; }
        public string Age { get; set; }
        public string Rej { get; set; }
        public string Light { get; set; }
        public string Choreo { get; set; }
    }
}
```

Листинг Б.5 – Файл Ticket.cs

```
namespace TheatreWebAPI.Models
{
    public class Ticket
```

```

    {
        public string ID { get; set; }
        public string AfishaID { get; set; }
        public string UserID { get; set; }
        public string PlaceID { get; set; }
    }
}

```

Листинг Б.6 – Файл TicketInfo.cs

```

namespace TheatreWebAPI.Models
{
    public class TicketInfo
    {
        public string AfishaID { get; set; }
        public string PlaceID { get; set; }
        public string Date { get; set; }
        public string Time { get; set; }
        public string ShowName { get; set; }
        public string PlaceNumber { get; set; }
        public string PlaceRow { get; set; }
        public string ArrName { get; set; }
    }
}

```

Листинг Б.7 – Файл User.cs

```

namespace TheatreWebAPI.Models
{
    public class User
    {
        public string UserID { get; set; }
        public string UserLogin { get; set; }
        public string UserPass { get; set; }
        public string UserPhoto { get; set; }
        public string UserFIO { get; set; }
        public string UserRights { get; set; }
    }
}

```

Листинг Б.8 – Файл TheatreDatabase.cs

```

using System.Data.SqlClient;

namespace TheatreWebAPI.Models
{
    public class TheatreDatabase
    {
        SqlConnection sqlConnection = new SqlConnection(@"Data
Source=DESKTOP-004NMG0\SQLEXPRESS;Initial Catalog=mobile;Integrated
Security=True");

        public void openConnection()
        {
            if (sqlConnection.State == System.Data.ConnectionState.Closed)
            {
                sqlConnection.Open();
            }
        }
    }
}

```

```

    }

    public void closeConnection()
    {
        if (sqlConnection.State == System.Data.ConnectionState.Open)
        {
            sqlConnection.Close();
        }
    }

    public SqlConnection GetConnection()
    {
        return sqlConnection;
    }
}
}

```

Листинг Б.9 – Файл SearchForShow.cs

```

using System.Data.SqlClient;

namespace TheatreWebAPI.Models
{
    public class SearchForShow
    {
        TheatreDatabase database = new TheatreDatabase();
        public Show Searching(string ID)
        {
            int id;
            Int32.TryParse(ID, out id);
            database.openConnection();
            Show show = new Show();

            string sqlQuery = $"select * from СпектакльView where ID='{id}'";
            SqlCommand sqlCommand = new SqlCommand(sqlQuery,
database.GetConnection());
            SqlDataReader dr = sqlCommand.ExecuteReader();
            while (dr.Read())
            {
                Object[] values = new Object[dr.FieldCount];
                for (int i = 0; i < dr.FieldCount; i++)
                {
                    values[i] = dr.GetValue(i);
                }
                show.ID = values[0].ToString();
                show.Name = values[1].ToString();
                show.Author = values[2].ToString();
                show.Info = values[3].ToString();
                show.Genre = values[4].ToString();
                show.Age = values[5].ToString();
                show.Rej = values[6].ToString();
                show.Light = values[7].ToString();
                show.Choreo = values[8].ToString();
            }
            dr.Close();
            database.closeConnection();
            return show;
        }
    }
}

```

Листинг Б.10 – Файл ModelUser.cs

```

using System.Data;
using System.Data.SqlClient;

namespace TheatreWebAPI.Models
{
    public class ModelUser
    {
        TheatreDatabase database = new TheatreDatabase();

        public User userInfo(string login, string password)
        {
            database.openConnection();
            string sqlQuery = $"select * from Пользователь where
Логин='{login}' and ХэшПароль='{password}'";
            SqlCommand sqlCommand = new SqlCommand(sqlQuery,
database.GetConnection());
            SqlDataReader dr = sqlCommand.ExecuteReader();
            User user = new User();
            user.UserID = "-1";
            while (dr.Read())
            {
                Object[] values = new Object[dr.FieldCount];
                for (int i = 0; i < dr.FieldCount; i++)
                {
                    values[i] = dr.GetValue(i);
                }
                user.UserID = values[0].ToString();
                user.UserLogin = values[1].ToString();
                user.UserPass = values[2].ToString();
                user.UserPhoto = values[3].ToString();
                user.UserFIO = values[4].ToString();
                user.UserRights = values[5].ToString();
            }
            dr.Close();
            database.closeConnection();
            return user;
        }

        public User userRegistration(string login, string password)
        {
            User user = new User();
            database.openConnection();
            string sqlQuery = $"insert into Пользователь (Логин, ХэшПароль,
Фото, ФИО, Права) values ('{login}','{password}','', '',1)";
            SqlDataAdapter adapter = new SqlDataAdapter();
            DataTable table = new DataTable();
            SqlCommand sqlCommand = new SqlCommand(sqlQuery,
database.GetConnection());
            adapter.SelectCommand = sqlCommand;
            adapter.Fill(table);
            user = userInfo(login, password);
            database.closeConnection();
            return user;
        }

        public User GetUserByID(string ID)
        {
            User user = new User();
            database.openConnection();
            string sqlQuery = $"select * from Пользователь where Id =
'{ID}'";
            SqlCommand sqlCommand = new SqlCommand(sqlQuery,
database.GetConnection());

```

```

SqlDataReader dr = sqlCommand.ExecuteReader();
while (dr.Read())
{
    Object[] values = new Object[dr.FieldCount];
    for (int i = 0; i < dr.FieldCount; i++)
    {
        values[i] = dr.GetValue(i);
    }
    user.UserID = values[0].ToString();
    user.UserLogin = values[1].ToString();
    user.UserPass = values[2].ToString();
    user.UserPhoto = values[3].ToString();
    user.UserFIO = values[4].ToString();
    user.UserRights = values[5].ToString();
}
dr.Close();
database.closeConnection();
return user;
}
}
}

```

Листинг Б.11 – Файл ModelTicketInfo.cs

```

using System.Collections.ObjectModel;
using System.Data.SqlClient;

namespace TheatreWebAPI.Models
{
    public class ModelTicketInfo
    {
        TheatreDatabase database = new TheatreDatabase();
        public ObservableCollection<TicketInfo> GetTickets(string userID)
        {
            database.openConnection();
            string sqlQuery = $"select Афиша.Id as 'АфишаId', Место.Id as 'МестоId', Афиша.Дата, Афиша.Время, Спектакль.Название, Место.[Номер места], Место.Ряд, Расположение.Название from Бронь join Афиша on Бронь.АфишаId = Афиша.Id join Спектакль on Афиша.СпектакльId = Спектакль.Id join Место on Бронь.МестоId = Место.Id join Расположение on Место.РасположениеId = Расположение.Id where Бронь.ПользовательId = '{userID}'";
            SqlCommand sqlCommand = new SqlCommand(sqlQuery, database.GetConnection());
            SqlDataReader dr = sqlCommand.ExecuteReader();
            ObservableCollection<TicketInfo> tickets = new ObservableCollection<TicketInfo>();
            while (dr.Read())
            {
                TicketInfo ticket = new TicketInfo();
                Object[] values = new Object[dr.FieldCount];
                for (int i = 0; i < dr.FieldCount; i++)
                {
                    values[i] = dr.GetValue(i);
                }
                ticket.AfishaID = values[0].ToString();
                ticket.PlaceID = values[1].ToString();
                ticket.Date = values[2].ToString();
                ticket.Time = values[3].ToString();
                ticket.ShowName = values[4].ToString();
                ticket.PlaceNumber = values[5].ToString();
                ticket.PlaceRow = values[6].ToString();
                ticket.ArrName = values[7].ToString();
                tickets.Add(ticket);
            }
        }
    }
}

```

```

    }
    dr.Close();
    database.closeConnection();
    return tickets;
}
}
}

```

Листинг Б.12 – Файл ModelPlace.cs

```

using System.Collections.ObjectModel;
using System.Data.SqlClient;

namespace TheatreWebAPI.Models
{
    public class ModelPlace
    {
        TheatreDatabase database = new TheatreDatabase();
        public ObservableCollection<Place> GetAllPlaces(string Arrangement)
        {
            database.openConnection();
            string sqlQuery =
                $"select Место.Id, [Номер места], Ряд, Расположение.Название
from Место join Расположение on Место.РасположениеId = Расположение.Id where
Расположение.Название = '{Arrangement}'";
            SqlCommand sqlCommand = new SqlCommand(sqlQuery,
            database.GetConnection());
            SqlDataReader dr = sqlCommand.ExecuteReader();

            ObservableCollection<Place> places = new
ObservableCollection<Place>();
            while (dr.Read())
            {
                Place place = new Place();
                Object[] values = new Object[dr.FieldCount];
                for (int i = 0; i < dr.FieldCount; i++)
                {
                    values[i] = dr.GetValue(i);
                }
                place.PlaceID = values[0].ToString();
                place.PlaceNumber = values[1].ToString();
                place.PlaceRow = values[2].ToString();
                place.PlaceArrangement = values[3].ToString();
                places.Add(place);
            }
            dr.Close();
            database.closeConnection();
            return places;
        }

        public ObservableCollection<Place> GetBookedPlaces(string afishaid,
string Arrangement)
        {
            database.openConnection();
            string sqlQuery =
                $"select Место.Id, [Номер места], Ряд, Расположение.Название
from Место join Расположение on Место.РасположениеId = Расположение.Id where
Место.Id in (select МестоId from Бронь where АфишаId = '{afishaid}' and
Расположение.Название = '{Arrangement}')";
            SqlCommand sqlCommand = new SqlCommand(sqlQuery,
            database.GetConnection());
            SqlDataReader dr = sqlCommand.ExecuteReader();

```

```

        ObservableCollection<Place> places = new
ObservableCollection<Place>();
        while (dr.Read())
        {
            Place place = new Place();
            Object[] values = new Object[dr.FieldCount];
            for (int i = 0; i < dr.FieldCount; i++)
            {
                values[i] = dr.GetValue(i);
            }
            place.PlaceID = values[0].ToString();
            place.PlaceNumber = values[1].ToString();
            place.PlaceRow = values[2].ToString();
            place.PlaceArrangement = values[3].ToString();
            places.Add(place);
        }
        dr.Close();
        database.closeConnection();
        return places;
    }

    public Place place(string row, string placenumb, string arrage)
    {
        database.openConnection();
        string sqlQuery = $"select Место.Id, [Номер места], Ряд,
Расположение.Название from Место join Расположение on Место.РасположениеId =
Расположение.Id where Ряд = '{row}' and [Номер места] = '{placenumb}' and
Расположение.Название = '{arrage}'";
        SqlCommand sqlCommand = new SqlCommand(sqlQuery,
database.GetConnection());
        SqlDataReader dr = sqlCommand.ExecuteReader();
        Place place = new Place();
        while (dr.Read())
        {
            Object[] values = new Object[dr.FieldCount];
            for (int i = 0; i < dr.FieldCount; i++)
            {
                values[i] = dr.GetValue(i);
            }
            place.PlaceID = values[0].ToString();
            place.PlaceNumber = values[1].ToString();
            place.PlaceRow = values[2].ToString();
            place.PlaceArrangement = values[3].ToString();
        }
        dr.Close();
        database.closeConnection();
        return place;
    }
}

```

Листинг Б.13 – Файл ModelBooking.cs

```

using System.Data;
using System.Data.SqlClient;

namespace TheatreWebAPI.Models
{
    public class ModelBooking
    {
        TheatreDatabase database = new TheatreDatabase();
        public string bookingTicket(string afishaid, string userid, string
placeid)

```

```

    {
        string errorMessage = "";
        database.openConnection();
        SqlInfoMessageEventHandler handler = (sender, args) =>
        {
            if (args.Message != null)
            {
                errorMessage = args.Message;
            }
        };
        database.GetConnection().InfoMessage += handler;
        string sqlQuery = $"exec Tickets '{afishaid}', '{userid}',
'{placeid}'";
        SqlDataAdapter adapter = new SqlDataAdapter();
        DataTable table = new DataTable();
        SqlCommand sqlCommand = new SqlCommand(sqlQuery,
database.GetConnection());
        adapter.SelectCommand = sqlCommand;
        adapter.Fill(table);
        database.closeConnection();
        return errorMessage;
    }

    public string removeBooking(string afishaid, string userid, string
placeid)
    {
        string errorMessage = "";
        database.openConnection();
        SqlInfoMessageEventHandler handler = (sender, args) =>
        {
            if (args.Message != null)
            {
                errorMessage = args.Message;
            }
        };
        database.GetConnection().InfoMessage += handler;
        string sqlQuery = $"delete from Бронь where ПользовательId =
'{userid}' and АфишаId = '{afishaid}' and МестоId = '{placeid}'";
        SqlDataAdapter adapter = new SqlDataAdapter();
        DataTable table = new DataTable();
        SqlCommand sqlCommand = new SqlCommand(sqlQuery,
database.GetConnection());
        adapter.SelectCommand = sqlCommand;
        adapter.Fill(table);
        database.closeConnection();
        return errorMessage;
    }
}

```

Листинг Б.14 – Файл ModelAfisha.cs

```

using System.Collections.ObjectModel;
using System.Data.SqlClient;

namespace TheatreWebAPI.Models
{
    public class ModelAfisha
    {
        TheatreDatabase database = new TheatreDatabase();
        public ObservableCollection<Afisha> afishaInfo()
        {
            database.openConnection();

```

```

        ObservableCollection<Afisha> Afishas = new
ObservableCollection<Afisha>();
        string sqlQuery = $"DECLARE @MinDate DATE = GETDATE(), @MaxDate
DATE = GETDATE() + 3; SELECT * FROM АфишаView WHERE Дата >= @MinDate
AND Дата < @MaxDate;";
        SqlCommand sqlCommand = new SqlCommand(sqlQuery,
database.GetConnection());
        SqlDataReader dr = sqlCommand.ExecuteReader();
        while (dr.Read())
        {
            Afisha afisha = new Afisha();
            Object[] values = new Object[dr.FieldCount];
            for (int i = 0; i < dr.FieldCount; i++)
            {
                values[i] = dr.GetValue(i);
            }
            afisha.ID = values[0].ToString();
            afisha.Date = values[1].ToString();
            afisha.Time = values[2].ToString();
            afisha.Name = values[3].ToString();
            afisha.Author = values[4].ToString();
            afisha.Age = values[5].ToString();
            afisha.ShowID = values[6].ToString();
            Afishas.Add(afisha);
        }
        dr.Close();
        database.closeConnection();
        return Afishas;
    }
}

```

Листинг Б.15 – Файл ModelActors.cs

```

using System.Collections.ObjectModel;
using System.Data.SqlClient;

namespace TheatreWebAPI.Models
{
    public class ModelActors
    {
        TheatreDatabase database = new TheatreDatabase();
        public ObservableCollection<Actors> getActors(string ID)
        {
            database.openConnection();
            string sqlQuery = $"select Актер.Id, Актер.ФИО, Актер.Фото,
Актер.Биография from Актер join [Распределение актеров] on [Распределение
актеров].АктерId2 = Актер.Id join Спектакль on [Распределение
актеров].СпектакльId = Спектакль.Id where Спектакль.Id = '{ID}'";
            SqlCommand sqlCommand = new SqlCommand(sqlQuery,
database.GetConnection());
            SqlDataReader dr = sqlCommand.ExecuteReader();
            ObservableCollection<Actors> actors = new
ObservableCollection<Actors>();
            while (dr.Read())
            {
                Actors actor = new Actors();
                Object[] values = new Object[dr.FieldCount];
                for (int i = 0; i < dr.FieldCount; i++)
                {
                    values[i] = dr.GetValue(i);
                }
                actor.ID = values[0].ToString();
            }
        }
    }
}

```

```

        actor.FIO = values[1].ToString();
        actor.Photo = values[2].ToString();
        actor.Bio = values[3].ToString();
        actors.Add(actor);
    }
    dr.Close();
    database.closeConnection();
    return actors;
}
}
}

```

Листинг Б.16 – Файл ShowsController.cs

```

using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using System.Collections.ObjectModel;
using TheatreWebAPI.Models;

namespace TheatreWebAPI.Controllers
{
    [ApiController]
    public class ShowsController : ControllerBase
    {
        // ModelShows db = new ModelShows();
        ModelAfisha dbAfisha = new ModelAfisha();
        SearchForShow ss = new SearchForShow();
        ModelUser mu = new ModelUser();
        ModelPlace mp = new ModelPlace();
        ModelBooking mb = new ModelBooking();
        ModelActors ma = new ModelActors();
        ModelTicketInfo mti = new ModelTicketInfo();

        [HttpGet]
        [Route("Options")]
        public async Task<ActionResult<HttpMethod>> GetOptions()
        {
            HttpRequestOptions requestOptions = new HttpRequestOptions();
            requestOptions.ToString();
            Response.Headers.Add("Allow", "GET, OPTIONS, POST");
            HttpMethod httpMethod = HttpMethod.Options;

            return httpMethod;
        }

        [HttpGet]
        [Route("Afisha")]
        public async Task<ActionResult<IEnumerable<Afisha>>> Get() //
получить все спектакли из афиши
        {
            return dbAfisha.afishaInfo();
        }

        [HttpGet]
        [Route("Show/{ID}")]
        public async Task<ActionResult<Show>> Get(string ID) // получить
подробную инф о конкретном спектакле
        {
            Show Show = ss.Searching(ID);
            return Show;
        }

        [HttpGet]

```

```

        [Route("Actors/{ID}")] // получить актеров участвующих в ID спектакле
        public async Task<ActionResult<IEnumerable<Actors>>> GetActors(string
ID) // получить подробную инф о конкретном спектакле
        {
            return ma.getActors(ID);
        }

        [HttpGet]
        [Route("AllPlaces/{Arrangement}")]
        public async Task<ActionResult<IEnumerable<Place>>>
GetAllPlaces(string Arrangement) // получить все свободные места
        {
            return mp.GetAllPlaces(Arrangement);
        }

        [HttpGet]
        [Route("Users/{ID}")]
        public async Task<ActionResult<User>> GetUserByID(string ID) //
получить объект Пользователь по ид
        {
            return mu.GetUserByID(ID);
        }

        [HttpPost]
        [Route("BookedPlaces")]
        public async Task<ActionResult<IEnumerable<Place>>> GetBookedPlaces()
// получить все свободные места
        {
            try
            {
                var Reader = new StreamReader(Request.Body);
                var content = await Reader.ReadToEndAsync();
                var jsonDeserialize =
JsonConvert.DeserializeObject<Dictionary<string, string>>(content);
                if (jsonDeserialize == null || jsonDeserialize["placeID"] ==
null || jsonDeserialize["placeArrangement"] == null)
                {
                    return BadRequest("Не все нужные данные были введены");
                }

                string afishaid = jsonDeserialize["placeID"];
                string arrang = jsonDeserialize["placeArrangement"];

                ObservableCollection<Place> booked =
mp.GetBookedPlaces(afishaid, arrang);

                return Ok(booked);
            }
            catch
            {
                return BadRequest();
            }
        }

        [HttpPost]
        [Route("Place")]
        public async Task<ActionResult<Place>> GetPlace() // получить ID
конкретного места
        {
            try
            {
                var Reader = new StreamReader(Request.Body);
                var content = await Reader.ReadToEndAsync();

```

```

        var jsonDeserialize =
JsonConvert.DeserializeObject<Dictionary<string, string>>(content);
        string placenumb = jsonDeserialize["placeNumber"];
        string placerow = jsonDeserialize["placeRow"];
        string placearr = jsonDeserialize["placeArrangement"];

        Place place = mp.place(placerow, placenumb, placearr);
        return Ok(place.PlaceID);
    }
    catch
    {
        return BadRequest();
    }
}

[HttpPost]
[Route("Authorization")]
public async Task<ActionResult<User>> PostAuthorization() //
авторизация в системе
{
    try
    {
        var Reader = new StreamReader(Request.Body);
        var content = await Reader.ReadToEndAsync();
        var jsonDeserialize =
JsonConvert.DeserializeObject<Dictionary<string, string>>(content);
        if (jsonDeserialize == null || jsonDeserialize["userLogin"]
== null || jsonDeserialize["userPass"] == null)
        {
            return BadRequest("Не все нужные данные были введены");
        }
        string login = jsonDeserialize["userLogin"];
        string password = jsonDeserialize["userPass"];
        User user = mu.userInfo(login, password);
        if (user.UserID == "-1")
        {
            return BadRequest("Пользователь не найден");
        }
        return Ok(user.UserID.ToString());
    }
    catch
    {
        return BadRequest();
    }
}

[HttpPost]
[Route("Registration")]
public async Task<ActionResult<User>> PostRegistration() //
регистрация в приложении
{
    try
    {
        var Reader = new StreamReader(Request.Body);
        var content = await Reader.ReadToEndAsync();
        var jsonDeserialize =
JsonConvert.DeserializeObject<Dictionary<string, string>>(content);
        if (jsonDeserialize == null || jsonDeserialize["userLogin"]
== null || jsonDeserialize["userPass"] == null)
        {
            return BadRequest("Не все нужные данные были введены");
        }
        string login = jsonDeserialize["userLogin"];
        string password = jsonDeserialize["userPass"];
    }
}

```

```

        User user = mu.userRegistration(login, password);
        return Ok(user.UserID);
    }
    catch
    {
        return BadRequest("-1");
    }
}

[HttpPost]
[Route("Booking")]
public async Task<ActionResult<User>> PostBooking() // регистрация в
приложении
{
    try
    {
        var Reader = new StreamReader(Request.Body);
        var content = await Reader.ReadToEndAsync();
        var jsonDeserialize =
JsonConvert.DeserializeObject<Dictionary<string, string>>(content);
        if (jsonDeserialize == null || jsonDeserialize["userID"] ==
null || jsonDeserialize["placeID"] == null)
        {
            return BadRequest("Не все нужные данные были введены");
        }
        string afishaID = jsonDeserialize["afishaID"];
        string userID = jsonDeserialize["userID"];
        string placeID = jsonDeserialize["placeID"];
        string error = mb.bookingTicket(afishaID, userID, placeID);
        return Ok(error);
    }
    catch
    {
        return BadRequest();
    }
}

[HttpPost]
[Route("UserTickets")]
public async Task<ActionResult<IEnumerable<TicketInfo>>>
GetUserTickets() // регистрация в приложении
{
    try
    {
        var Reader = new StreamReader(Request.Body);
        var content = await Reader.ReadToEndAsync();
        var jsonDeserialize =
JsonConvert.DeserializeObject<Dictionary<string, string>>(content);
        if (jsonDeserialize == null || jsonDeserialize["userID"] ==
null)
        {
            return BadRequest("Ошибка");
        }
        string userID = jsonDeserialize["userID"];
        ObservableCollection<TicketInfo> tickets =
mti.GetTickets(userID);
        return Ok(tickets);
    }
    catch
    {
        return BadRequest();
    }
}

```

```

[HttpPost]
[Route("RemoveBooking")]
public async Task<ActionResult<IEnumerable<TicketInfo>>>
RemoveBooking() // регистрация в приложении
{
    try
    {
        var Reader = new StreamReader(Request.Body);
        var content = await Reader.ReadToEndAsync();
        var jsonDeserialize =
JsonConvert.DeserializeObject<Dictionary<string, string>>(content);
        if (jsonDeserialize == null || jsonDeserialize["userID"] ==
null || jsonDeserialize["afishaID"] == null || jsonDeserialize["placeID"] ==
null)
        {
            return BadRequest("Ошибка");
        }
        string userid = jsonDeserialize["userID"];
        string afishaId = jsonDeserialize["afishaID"];
        string placeid = jsonDeserialize["placeID"];
        string msg = mb.removeBooking(afishaId, userid, placeid);
        return Ok(msg);
    }
    catch
    {
        return BadRequest();
    }
}
}
}

```

Листинг Б.17 – Файл launchSettings.json

```

{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://192.168.0.12:27711",
      "sslPort": 44340
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "MyServer": {
      "commandName": "Project",
      "launchBrowser": false,
      "applicationUrl": "http://192.168.0.12:7124",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "TheatreWebAPI": {
      "commandName": "Project",

```

```

        "launchBrowser": true,
        "applicationUrl": "http://192.168.0.12:7124;http://192.168.0.12t:5124",
        "environmentVariables": {
            "ASPNETCORE_ENVIRONMENT": "Development"
        }
    }
}
}

```

Клиентская часть.

Листинг Б.18 – Файл RegistrationChecker.cs

```

using System.Text.RegularExpressions;

namespace mobile.Helpers
{
    public static class RegistrationChecker
    {
        public static bool ValidateLogin(string input) // проверка строки на
        правильность ввода логина
        {
            if (input != null)
            {
                string pattern = @"^[^/;:*&!?,\[\] ()^%$#@'\\"{4,16}$"; //
                шаблон для проверки
                Regex regex = new Regex(pattern);
                return regex.IsMatch(input);
            }
            else
            {
                return false;
            }
        }

        public static bool ValidatePassword(string input) // проверка строки
        на правильность ввода пароля
        {
            if (input != null)
            {
                string pattern = @"^(?=.*\d){8,32}$"; // шаблон для проверки
                Regex regex = new Regex(pattern);
                return regex.IsMatch(input);
            }
            else
            {
                return false;
            }
        }
    }
}

```

Листинг Б.19 – Файл HttpHelper.cs

```

using System.Net;
using System.Net.Http;
using System.Threading.Tasks;
using System.Net.Http.Json;
using System;

namespace mobile.Services

```

```

{
    public class HttpHelper
    {
        public static async Task<HttpResponseBody> POST(string URI,
        JsonConvert json)
        {
            var client = new HttpClient();
            HttpResponseMessage responseBody = null;
            try
            {
                responseBody = await client.PostAsync(URI, json);
            }
            catch (Exception ex)
            {
                if (responseBody == null)
                    responseBody = new HttpResponseMessage();
                responseBody.StatusCode =
                HttpStatusCode.InternalServerError;
                responseBody.Content = new StringContent("Подключение
                отсутствует");
            }
            return responseBody;
        }
    }
}

```

Листинг Б.20 – Файл ShowService.cs

```

using System.Collections.Generic;
using System.Net.Http;
using System.Text.Json;
using System.Threading.Tasks;
using mobile.Entities;
using System.Net.Http.Json;

namespace mobile.Services
{
    public class ShowService
    {
        string Url; // обращайтесь внимание на конечный слеш //
        // настройки для десериализации для нечувствительности к регистру символов
        JsonSerializerOptions options = new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true,
        };

        // настройка клиента
        private HttpClient GetClient()
        {
            HttpClient client = new HttpClient();
            client.DefaultRequestHeaders.Add("Accept", "application/json");
            return client;
        }

        // получаем все спектакли
        public async Task<IEnumerable<Afisha>> Get()
        {
            Url = "http://192.168.0.12:7124/Afisha/";
            HttpClient client = GetClient();
            string result = await client.GetStringAsync(Url);
        }
    }
}

```

```

        return JsonSerializer.Deserialize<IEnumerable<Afisha>>(result,
options);
    }

    //получаем 1 спектакль
    public async Task<Show> GetShow(string ID)
    {
        Url = "http://192.168.0.12:7124/Show/" + ID;
        HttpClient client = GetClient();
        string result = await client.GetStringAsync(Url);
        return JsonSerializer.Deserialize<Show>(result, options);
    }

    public async Task<IEnumerable<Place>> GetAllPlaces(string
Arrangement)
    {
        Url = "http://192.168.0.12:7124/AllPlaces/" + Arrangement;
        HttpClient client = GetClient();
        string result = await client.GetStringAsync(Url);
        return JsonSerializer.Deserialize<IEnumerable<Place>>(result,
options);
    }

    public async Task<HttpResponseMessage> GetBookedPlaces(Place place)
    {
        JsonContent json = JsonContent.Create(place);
        var result = await
HttpHelper.POST("http://192.168.0.12:7124/BookedPlaces/", json);
        return result;
    }

    public async Task<HttpResponseMessage> GetPlace(string row, string
numb, string arr)
    {
        Place place = new Place();
        place.placeRow = row;
        place.placeNumber = numb;
        place.placeArrangement = arr;
        JsonContent json = JsonContent.Create(place);
        var result = await
HttpHelper.POST("http://192.168.0.12:7124/Place/", json);
        return result;
    }

    // авторизация пользователя
    public async Task<HttpResponseMessage> Authorization(User user)
    {
        JsonContent json = JsonContent.Create(user);
        var result = await
HttpHelper.POST("http://192.168.0.12:7124/Authorization/", json);
        return result;
    }

    // регистрация пользователя
    public async Task<HttpResponseMessage> Registration(User user)
    {
        JsonContent json = JsonContent.Create(user);
        var result = await
HttpHelper.POST("http://192.168.0.12:7124/Registration/", json);
        return result;
    }

    // забронировать билет
    public async Task<HttpResponseMessage> Booking(Ticket ticket)
    {

```

```

        JsonConvert json = JsonConvert.Create(ticket);
        var result = await
HttpHelper.POST("http://192.168.0.12:7124/Booking/", json);
        return result;
    }

    // убрать бронь билета
    public async Task<HttpResponseMessage> RemoveBooking(Ticket ticket)
    {
        JsonConvert json = JsonConvert.Create(ticket);
        var result = await
HttpHelper.POST("http://192.168.0.12:7124/RemoveBooking/", json);
        return result;
    }

    // получить актеров которые участвуют в спектакле
    public async Task<IEnumerable<Actors>> GetActors(string ID)
    {
        Url = "http://192.168.0.12:7124/Actors/" + ID;
        HttpClient client = GetClient();
        string result = await client.GetStringAsync(Url);
        return JsonSerializer.Deserialize<IEnumerable<Actors>>(result,
options);
    }

    public async Task<HttpResponseMessage> GetUserTickets(User user)
    {
        JsonConvert json = JsonConvert.Create(user);
        var result = await
HttpHelper.POST("http://192.168.0.12:7124/UserTickets/", json);
        return result;
    }

    public async Task<User> GetUserByID(string ID)
    {
        Url = "http://192.168.0.12:7124/Users/" + ID;
        HttpClient client = GetClient();
        string result = await client.GetStringAsync(Url);
        return JsonSerializer.Deserialize<User>(result, options);
    }
}
}

```

Листинг Б.21 – Файл ActorInfoPage.xaml

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:mobile"
        x:DataType="local:ActorInfoPage"
        x:Class="mobile.ActorInfoPage"
        Title="Биография">
    <ContentPage.Content>
        <ScrollView>
            <StackLayout x:Name="actorStack">

                </StackLayout>
        </ScrollView>
    </ContentPage.Content>
</ContentPage>

```

Листинг Б.22 – Файл ActorInfoPage.xaml.cs

```

using Xamarin.Forms;
using Xamarin.Forms.Xaml;
using mobile.Entities;

namespace mobile
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class ActorInfoPage : ContentPage
    {
        Actors actor1;
        public ActorInfoPage(Actors actor)
        {
            actor1 = actor;
            actorStack = new StackLayout();
            InitializeComponent();
            GetActorInfo();
        }

        public void GetActorInfo()
        {
            Image actorImage = new Image
            {
                Source = actor1.Photo,
                Margin = new Thickness(1,1,1,0),
            };
            Frame imageFrame = new Frame
            {
                Margin = new Thickness(5),
                BorderColor = Color.Black,
                Content = actorImage,
            };
            Label nameLabel = new Label
            {
                FontSize = 24,
                FontAttributes = FontAttributes.Bold,
                Margin = new Thickness(15,0,0,0),
                TextColor = Color.Black,
                Text = actor1.FIO,
            };
            actorStack.Children.Add(imageFrame);
            actorStack.Children.Add(nameLabel);

            Label lbl = new Label
            {
                Margin = new Thickness(15,0,0,0),
                TextColor= Color.Black,
                FontAttributes= FontAttributes.Bold,
                FontSize = 18,
                Text = "Биография",
            };
            actorStack.Children.Add(lbl);

            Label bioLabel = new Label
            {
                FontSize = 14,
                FontAttributes = FontAttributes.Italic,
                TextColor = Color.Black,
                Margin = new Thickness (30,0,30,20),
                Text = actor1.Bio,
                HorizontalTextAlignment = TextAlignment.Start,
            };
            actorStack.Children.Add(bioLabel);
        }
    }
}

```

```

    }
}

```

Листинг Б.23 – Файл Afisha.xaml

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              xmlns:local="clr-namespace:mobile"
                    x:DataType="local:Page1"
                    x:Class="mobile.Page1"
                    Title="Афиша"
                    >
    <ContentPage.Content>
        <ScrollView VerticalOptions="FillAndExpand"
                    HorizontalOptions="FillAndExpand" VerticalScrollBarVisibility="Always">
            <StackLayout x:Name="page1Stack" VerticalOptions="FillAndExpand"
                    HorizontalOptions="FillAndExpand">
                </StackLayout>
            </ScrollView>
        </ContentPage.Content>
    </ContentPage>

```

Листинг Б.24 – Файл Afisha.xaml.cs

```

using System;
using System.Linq;
using System.Collections.Generic;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
using mobile.Services;
using mobile.Entities;

namespace mobile
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Page1 : ContentPage
    {
        public Page1()
        {
            InitializeComponent();
            GetShows();
        }

        public async void GetShows()
        {
            ShowService ss = new ShowService();
            IEnumerable<Afisha> afishas = await ss.Get(); // получаем все
спектакли
            var dates = from af in afishas group af by af.Date;
            foreach(var date in dates)
            {
                StackLayout stackLayout = new StackLayout();
                Label label = new Label
                {
                    Text = date.Key.Remove(11),
                    Padding = new Thickness(15,5,5,5),
                    FontSize = 20,
                    TextColor = Color.Black,
                    BackgroundColor = Color.DarkGray,
                };
                stackLayout.Children.Add(label);
                StackLayout childStack = new StackLayout();
                foreach (var af in date)

```

```

{
    StackLayout sl = new StackLayout
    {
        VerticalOptions = LayoutOptions.FillAndExpand,
        HeightRequest = MinimumHeightRequest,
        Spacing = 1,
    };
    Label lb = new Label
    {
        FontSize = 20,
        TextColor = Color.Black,
        FontAttributes = FontAttributes.Bold,
        Text = af.Name,
    };
    sl.Children.Add(lb);
    Label lbl = new Label
    {
        FontSize = 14,
        TextColor = Color.Black,
        Text = "Автор: " + af.Author + "\nВозраст: " + af.Age
+ "\n\nДата и время: " + af.Date.Remove(11) + af.Time,
    };
    sl.Children.Add(lbl);
    var tap1 = new TapGestureRecognizer();
    tap1.Tapped += (object sender, EventArgs e) =>
    {
        Navigation.PushAsync(new Page2(af.ShowID, af.ID));
    };
    sl.GestureRecognizers.Add(tap1);
    Frame fr = new Frame
    { Content = sl, BackgroundColor = Color.LightCyan };
    fr.BorderColor = Color.Black;
    fr.GestureRecognizers.Add(tap1);
    childStack.Children.Add(fr);
}
var tap = new TapGestureRecognizer();
tap.Tapped += (object sender, EventArgs e) =>
{
    StackLayout stack = (sender as Label).Parent as
StackLayout; // page2Stack
    foreach (var st in stack.Children)
    {
        if (st.Equals(childStack) && st.IsVisible)
        {
            st.IsVisible = false;
            label.BackgroundColor = Color.Gray;
        }
        else if (st.Equals(childStack) && !st.IsVisible)
        {
            st.IsVisible = true;
            label.BackgroundColor = Color.DarkGray;
        }
    }
};
label.GestureRecognizers.Add(tap);
stackLayout.Children.Add(childStack);
page1Stack.Children.Add(stackLayout);
}
Label label1 = new Label();
label1.Text = "";
page1Stack.Children.Add(label1);
}
}

```

Листинг Б.25 – Файл BookingPlacesPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>

    <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="mobile.BookingPlacesPage"
        Title="Бронирование билетов">

        <ContentPage.Content>
            <StackLayout x:Name="bookingPlacesStack">
            </StackLayout>
        </ContentPage.Content>

    </ContentPage>
```

Листинг Б.26 – Файл BookingPlacesPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text.Json;
using mobile.Entities;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
using mobile.Services;

namespace mobile
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class BookingPlacesPage : ContentPage
    {
        Frame fr;
        string afisha;
        ShowService ss;
        public BookingPlacesPage(Frame frame, string afishaid)
        {
            fr = frame;
            afisha = afishaid;
            ss = new ShowService();
            InitializeComponent();
            MakePage(fr);
        }

        public async void MakePage(Frame fr)
        {
            string selectedPlace = "";
            string Arrangement = "";
            var cont = fr.Content as Label;
            string name = cont.Text;
            Arrangement = name;
            ScrollView plScrollView = new ScrollView { Orientation =
            ScrollOrientation.Horizontal };
            Image image = new Image
            {
                Margin = new Thickness(10),

            };
            Place pll = new Place
            {
```

```

        placeArrangement = Arrangement,
        placeID = afisha,
    };

    double size = 20;
    double fontsize = 12;

    if (name == "Партер" || name == "Амфитеатр" || name ==
"Бельэтаж")
    {
        if (name == "Амфитеатр")
        {
            image.Source = "amf.jpg";
            size = 26;
            fontsize = 10;
        }
        if (name == "Партер")
            image.Source = "parter.jpg";
        if (name == "Бельэтаж")
            image.Source = "bel.jpg";
    }
    else
    {
        if (name == "Балкон")
            image.Source = "balcon.jpg";
        if (name == "Ложа")
            image.Source = "loja.jpg";
    }

    IEnumerable<Place> allPlaces = await
ss.GetAllPlaces(Arrangement);
    var bookedPlacesResponse = await ss.GetBookedPlaces(pl1);
    string response = await
bookedPlacesResponse.Content.ReadAsStringAsync();
    ObservableCollection<Place> bookedPlacesCollection =
JsonSerializer.Deserialize<ObservableCollection<Place>>(response);
    Label labelSelected = new Label { TextColor = Color.Black,
FontSize = 20, Margin = new Thickness(10,0,0,0), FontAttributes =
FontAttributes.Bold };
    StackLayout plStack = new StackLayout { Margin = new
Thickness(10, 0, 0, 0) };
    plStack.HorizontalOptions = LayoutOptions.Center;
    var rows = from row in allPlaces group row by row.placeRow;
    bookingPlacesStack.Children.Add(image);
    bookingPlacesStack.Children.Add(new Label { Text = "Выберите
место", Margin = new Thickness(10, 0, 0, -10), FontSize = 20, FontAttributes
= FontAttributes.Bold, TextColor = Color.Black });

    foreach (var r in rows)
    {

        Thickness thick = new Thickness(5);
        StackLayout childStack = new StackLayout { Margin = new
Thickness(10) };

        if (name == "Партер" || name == "Амфитеатр" || name ==
"Бельэтаж")
        {
            if (name == "Амфитеатр")
            {
                thick = new Thickness(0, 0, 0, 0);
                plStack.HorizontalOptions = LayoutOptions.Start;
                plStack.Margin = new Thickness(0, 10, 0, 0);
                childStack.Margin = new Thickness(0, 10, 0, 0);
            }
        }
    }
}

```

```

        childStack.HorizontalOptions = LayoutOptions.Start;
        childStack.Spacing = 3;
    }
    plStack.Orientation = StackOrientation.Vertical;
    childStack.Orientation = StackOrientation.Horizontal;
}
else
{
    plStack.Orientation = StackOrientation.Horizontal;
    childStack.Orientation = StackOrientation.Vertical;
}

foreach (var ap in r)
{
    string row = "";
    string numb = "";
    Frame frame = new Frame();

    frame.HeightRequest = size;
    frame.WidthRequest = size;
    frame.CornerRadius = (float)size;
    frame.Padding = thick;
    frame.HorizontalOptions = LayoutOptions.Start;
    frame.VerticalOptions = LayoutOptions.Start;
    frame.BackgroundColor = Color.Green;

    Label rowLab = new Label();
    rowLab.Text = ap.placeNumber;
    var tapplace = new TapGestureRecognizer();

    tapplace.Tapped += async (object sender1, EventArgs e1)
=>
    {
        row = ap.placeRow; numb = ap.placeNumber;
        var placeGot = await ss.GetPlace(row, numb,
Arrangement);
        selectedPlace = await
placeGot.Content.ReadAsStringAsync();
        labelSelected.Text = "Выбранное место: " +
selectedPlace;
    };

    rowLab.HorizontalOptions = LayoutOptions.Center;
    rowLab.VerticalOptions = LayoutOptions.Center;
    rowLab.FontSize = fontsize;

    frame.Content = rowLab;
    rowLab.GestureRecognizers.Add(tapplace);
    frame.GestureRecognizers.Add(tapplace);
    foreach (var bp in bookedPlacesCollection)
    {
        if (bp.placeID == ap.placeID)
        {
            frame.BackgroundColor = Color.Red;
            var tapbooked = new TapGestureRecognizer();
            tapbooked.Tapped += async (object sender2,
EventArgs e2) =>
            {
                await DisplayAlert("Уведомление", "Данное
место уже занято!", "OK");
            };
            frame.GestureRecognizers.Remove(tapplace);
            rowLab.GestureRecognizers.Remove(tapplace);

```

```

        frame.GestureRecognizers.Add(tapbooked);
        rowLab.GestureRecognizers.Add(tapbooked);
    }
    }
    childStack.Children.Add(frame);
}
plScrollView.Content = plStack;
plStack.Children.Add(childStack);
}

bookingPlacesStack.Children.Add(plStack);
bookingPlacesStack.Children.Add(labelSelected);

Button myButton = new Button {
    Margin = new Thickness(40, 20, 40, 10),
    Text = "Забронировать билет",
    BackgroundColor = Color.White,
    BorderColor = Color.Black,
    BorderWidth = 1,
};
myButton.Clicked += async (sender, e) =>
{
    if (LoginVariable.UserID != "-1")
    {
        Ticket ticket = new Ticket();
        ticket.AfishaID = afisha;
        ticket.UserID = LoginVariable.UserID;
        ticket.PlaceID = selectedPlace;
        var bookingmessage = await ss.Booking(ticket);
        string errormessage = await
bookingmessage.Content.ReadAsStringAsync();
        if (errormessage == "")
        {
            await DisplayAlert("Уведомление", "Билет успешно
забронирован!", "OK");
        }
        else
        {
            await DisplayAlert("Ошибка", errormessage, "OK");
        }
    }
    else
    {
        await DisplayAlert("Ошибка", "Вы не вошли в аккаунт!",
"OK");
    }
};

bookingPlacesStack.Children.Add(myButton);
}
}
}

```

Листинг Б.27 – Файл LoginPage.xaml

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:mobile"
        x:DataType="local:LoginPage"
        x:Class="mobile.LoginPage"
        Title="Авторизация">
    <ContentPage.Content>

```

```

        <StackLayout x:Name="LoginStack" HorizontalOptions="Center"
Margin="0,20,0,0">
            <Entry x:Name="loginBox" Margin="10,10,10,10"
Placeholder="Логин"/>
            <Entry IsPassword="True" Margin="10,10,10,10"
x:Name="passwordBox" Placeholder="Пароль"/>
            <Button x:Name="loginButton" Margin="10,10,10,10"
Text="Авторизоваться" Clicked="loginButton_Clicked" BorderColor="Black"
BorderWidth="1" BackgroundColor="White"/>
            <Button x:Name="registrationButton" Margin="10,10,10,10"
Text="Зарегистрироваться" Clicked="registrationButton_Clicked"
BorderColor="Black" BorderWidth="1" BackgroundColor="White"/>
            <Button x:Name="personalAccountButton" Margin="10,10,10,10"
Text="Перейти в личный кабинет" Clicked="personalAccountButton_Clicked"
BorderColor="Black" BorderWidth="1" BackgroundColor="White"
VerticalOptions="End"/>
            <Button x:Name="logoutButton" Margin="10,10,10,10" Text="Выход"
Clicked="logoutButton_Clicked" BorderColor="Black" BorderWidth="1"
BackgroundColor="White" VerticalOptions="End"/>
        </StackLayout>

    </ContentPage.Content>
</ContentPage>

```

Листинг Б.28 – Файл LoginPage.xaml.cs

```

using System;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
using mobile.Services;
using mobile.Entities;
using mobile.Helpers;

namespace mobile
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class LoginPage : ContentPage
    {
        ShowService ss = new ShowService();
        public LoginPage()
        {
            InitializeComponent();
        }

        private void loginButton_Clicked(object sender, EventArgs e)
        {
            loginMethod(loginBox.Text, passwordBox.Text);
        }

        private async void loginMethod(string login, string pass)
        {
            if (LoginVariable.UserID == "-1")
            {
                User user = new User();
                user.UserLogin = login;
                user.UserPass = pass;
                ShowService ss = new ShowService();
                var userId = await ss.Authorization(user);
                string useridthmp = await userId.Content.ReadAsStringAsync();
                int c;
                Int32.TryParse(useridthmp, out c);
                if (c == 0)

```

```

        await DisplayAlert("Ошибка", "Логин или пароль указаны
неверно!", "OK");
    else
    {
        LoginVariable.UserID = userdtmp;
        await DisplayAlert("Уведомление", "Вы успешно
авторизованы!", "OK");
        loginBox.Text = "";
        passwordBox.Text = "";
        await Navigation.PushAsync(new Page1());
    }
}
else
{
    await DisplayAlert("Ошибка", "Вы уже авторизованы!", "OK");
}
}

private async void registrationMethod(string login, string password)
{
    if (LoginVariable.UserID != "-1")
        await DisplayAlert("Ошибка", "Вы уже авторизованы!", "OK");
    else
    {
        User user = new User();
        user.UserLogin = login;
        user.UserPass = password;
        ShowService ss = new ShowService();
        string userId = "-1";
        var tmp = await ss.Authorization(user);
        userId = await tmp.Content.ReadAsStringAsync();
        int c;
        Int32.TryParse(userId, out c);
        if (c == 0)
        {
            user = new User();
            user.UserLogin = login;
            user.UserPass = password;
            ss = new ShowService();
            var newUserId = await ss.Registration(user);
            string temp = await
newUserId.Content.ReadAsStringAsync();
            if (temp == "-1")
                await DisplayAlert("Ошибка", "Пользователь с таким
логинем уже существует!", "OK");
            else
            {
                await DisplayAlert("Уведомление", "Вы
зарегистрированы!", "OK");
                LoginVariable.UserID = await
newUserId.Content.ReadAsStringAsync();
            }
        }
        else
        {
            await DisplayAlert("Ошибка", "Пользователь с данным
логинем уже существует!", "OK");
        }
    }
}

private async void registrationButton_Clicked(object sender,
EventArgs e)
{

```

```

        if (loginBox.Text == "")
            await DisplayAlert("Ошибка", "Логин не может быть пустым!",
"OK");
        else
        {
            if (RegistrationChecker.ValidateLogin(loginBox.Text))
            {
                if
(RegistrationChecker.ValidatePassword(passwordBox.Text))
                    registrationMethod(loginBox.Text, passwordBox.Text);
                else
                {
                    await DisplayAlert("Ошибка", "Пароль должен иметь как
минимум 1 цифру и содержать не менее 8 символов", "OK");
                }
            }
            else
            {
                await DisplayAlert("Ошибка", "Логин должен содержать не
менее 4 и не более 16 символов!", "OK");
            }
        }
    }

    private void logoutButton_Clicked(object sender, EventArgs e)
    {
        if (LoginVariable.UserID == "-1")
        {
            DisplayAlert("Ошибка", "Вы еще не вошли в аккаунт!", "OK");
        }
        else
        {
            LoginVariable.UserID = "-1";
            DisplayAlert("Уведомление", "Вы вышли из аккаунта", "OK");
        }
    }

    private async void personalAccountButton_Clicked(object sender,
EventArgs e)
    {
        if (LoginVariable.UserID == "-1")
        {
            await DisplayAlert("Ошибка", "Вы не вошли в аккаунт!", "OK");
        }
        else
        {
            await Navigation.PushAsync(new PersonalAccount());
        }
    }
}

public static class LoginVariable
{
    public static string UserID = "-1";
}
}

```

Листинг Б.29 – Файл MainPage.xaml

```

<?xml version="1.0" encoding="utf-8" ?>
<Shell
    xmlns="http://xamarin.com/schemas/2014/forms"

```

```

        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:mobile" x:DataType="local:MainPage"
        x:Class="mobile.MainPage"
        BackgroundColor="LightCoral"
        Title="О нас"
    >

    <Shell.FlyoutFooter>
        <StackLayout>
            <Label Text="ИТВ, 2023" FontSize="30" FontAttributes="Bold"
HorizontalOptions="Center" VerticalOptions="Center" />
        </StackLayout>
    </Shell.FlyoutFooter>

    <Shell.FlyoutHeader>
        <StackLayout>
            <Image Source="th.png" HeightRequest="100" WidthRequest="100"
HorizontalOptions="Center" VerticalOptions="Center" Margin="0,20,0,20"/>
        </StackLayout>

    </Shell.FlyoutHeader>
    <FlyoutItem Title="Авторизация" Icon="loginicon.png">
        <ShellContent>
            <local:LoginPage/>
        </ShellContent>
    </FlyoutItem>

    <FlyoutItem Title="Афиша" Icon="afishaicon.png">
        <ShellContent>
            <local:Page1/>
        </ShellContent>
    </FlyoutItem>

    <FlyoutItem Title="О театре" Icon="info.png">
        <ContentPage Title="О нас">
            <ScrollView>
                <StackLayout BackgroundColor="White">

                    <Frame Margin="30,25,30,25" BorderColor="Black"
BackgroundColor="LightCyan">
                        <Label Margin="5,0,5,0" TextColor="Black"
FontAttributes="Italic" FontSize="16" Text=
                            "Я -- Пензенский областной театр кукол, родился, как вы
поняли, в Пензе в декабре 1942 года, когда актерами-энтузиастами М. Калакиным
и Я. Кусакиным была организована при оборонном театре миниатюр кукольная
бригада. По счастью, в мае 1943 г. рождение мое было узаконено официальным
постановлением о создании в Пензе театра кукол. Именно эта дата стоит в моем
паспорте, а поскольку с таким серьезным документом не спорят, то она и
считается официальной датой рождения. С 1946 по 1991 г. я неоднократно менял
место жительства. Первое мое пристанище было по адресу ул. Московская, 7. В
1961 г. состоялся переезд на ул. Московская, 1. Меня вообще в этой жизни
помотало, побросало, побило. В 1977 г. я переехал на Московскую, 17, но в
августе 1991 г. здание признали аварийным и уходящая в отставку партия отдала
мне одно из своих помещений по ул. Чкалова 35, где я и сейчас живу под новым
именем 'Кукольный Дом' и стараюсь быть Домом для всех, кто ко мне приходит.
Стараюсь быть добрым, умным, нежным. Вот и все обо мне. Остальное надо
видеть. Так что - милости просим!"/>
                    </Frame>
                </StackLayout>
            </ScrollView>
        </ContentPage>
    </FlyoutItem>

```

```

        </FlyoutItem>

        <!-- -->

    </Shell>

```

Листинг Б.30 – Файл MainPage.xaml.cs

```

using Xamarin.Forms;

namespace mobile
{
    public partial class MainPage : Shell
    {
        public MainPage()
        {
            InitializeComponent();
            ContentPage Page11 = new ContentPage();
        }
    }
}

```

Листинг Б.31 – Файл PersonalAccount.xaml

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              xmlns:local="clr-namespace:mobile"
              x:DataType="local:PersonalAccount"
              x:Class="mobile.PersonalAccount"
              Title="Личный кабинет">
    <ContentPage.Content>
        <ScrollView>
            <StackLayout x:Name="AccountStack">
            </StackLayout>
        </ScrollView>
    </ContentPage.Content>
</ContentPage>

```

Листинг Б.32 – Файл PersonalAccount.xaml.cs

```

using System;
using System.Collections.ObjectModel;
using System.Text.Json;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
using mobile.Services;
using mobile.Entities;
using System.Globalization;

namespace mobile
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class PersonalAccount : ContentPage
    {
        private ShowService ss = new ShowService();
        public PersonalAccount()
        {
            InitializeComponent();
            MakePage();
        }
    }
}

```

```

    }

    public async void MakePage()
    {
        User userInfo = await ss.GetUserByID(LoginVariable.UserID);
        Image userImage = new Image
        {
            Source = "actor.jpg",
            HeightRequest = 200,
            Margin = new Thickness(5),
        };
        Frame photoFrame = new Frame
        {
            Content = userImage,
            BorderColor = Color.Black,
            Margin = new Thickness(5),
        };
        Label userLabel = new Label
        {
            Text = userInfo.UserLogin + " (Вы)",
            Margin = new Thickness(15, 15, 0, 10),
            FontSize = 24,
            TextColor = Color.Black,
            FontAttributes = FontAttributes.Bold,
        };
        AccountStack.Children.Add(photoFrame);
        AccountStack.Children.Add(userLabel);

        Label brtext = new Label
        {
            Text = "Забронированные билеты:",
            Margin = new Thickness(30, 0, 0, 0),
            FontSize = 20,
            TextColor = Color.Black,
        };
        AccountStack.Children.Add(brtext);

        StackLayout ticketsstack = new StackLayout
        {
            Margin = new Thickness(0, 0, 0, 15),
        };
        User user = new User();
        user.UserID = LoginVariable.UserID;
        var tickets = await ss.GetUserTickets(user);
        string ticket = await tickets.Content.ReadAsStringAsync();
        ObservableCollection<TicketInfo> ticks;
        ticks =
        JsonSerializer.Deserialize<ObservableCollection<TicketInfo>>(ticket);

        int c = 1;
        foreach (var t in ticks)
        {
            Label ticketsLabel = new Label();
            ticketsLabel.FontSize = 14;
            ticketsLabel.TextColor = Color.Black;
            ticketsLabel.Text = "Дата и время: " + t.date.Remove(10) + ", " +
            t.time.Remove(5) + " \nНазвание спектакля: " + t.showName +
            "\nРасположение: " + t.arrName + " \nРяд, место: " + t.placeRow + ", " +
            t.placeNumber;

            Frame ticketFrame = new Frame
            {
                Margin = new Thickness(30, 0, 30, 0),
                BorderColor = Color.Black,
                BackgroundColor = Color.LightCyan,
            }

```

```

        Content = ticketsLabel,
    };
    var tap1 = new TapGestureRecognizer();
    tap1.Tapped += async (object sender, EventArgs e) =>
    {
        string selectedOption = await
DisplayActionSheet("Выберите опцию", "Отмена", null, "Отменить бронь");
        if (selectedOption != null && selectedOption != "Отмена")
        {
            Ticket ticket1 = new Ticket();
            ticket1.PlaceID = t.placeID;
            ticket1.AfishaID = t.afishaID;
            ticket1.UserID = LoginVariable.UserID;
            var tmp = await ss.RemoveBooking(ticket1);
            string msg = await tmp.Content.ReadAsStringAsync();
            if (msg == "")
                msg = "Бронь отменена!";
            await DisplayAlert("Уведомление", msg, "OK");
        }
    };
    ticketsLabel.GestureRecognizers.Add(tap1);
    ticketFrame.GestureRecognizers.Add(tap1);
    string dateString = t.date;
    string format = "dd.MM.yyyy H:mm:ss";
    DateTime result;
    DateTime.TryParseExact(dateString, format,
CultureInfo.InvariantCulture, DateTimeStyles.None, out result);
    string todayDate = DateTime.Today.ToString(); //"5/21/2023
12:00:00 AM"

    string bookingDate = "";
    if (result == DateTime.Today || result > DateTime.Today)
        ticketsstack.Children.Add(ticketFrame);
    c++;
}
AccountStack.Children.Add(ticketsstack);
}
}
}

```

Листинг Б.33 – Файл ShowInfo.xaml

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="mobile.Page2"
    Title="Информация о спектакле">
    <ContentPage.Content>
        <ScrollView>
            <StackLayout x:Name="page2Stack">

                </StackLayout>
            </ScrollView>
        </ContentPage.Content>
    </ContentPage>

```

Листинг Б.34 – Файл ShowInfo.xaml.cs

```

using System;
using System.Collections.ObjectModel;
using mobile.Entities;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

```

```

using mobile.Services;

namespace mobile
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class Page2 : ContentPage
    {
        private string show_id;
        private string afisha_id;
        public Page2()
        {
            InitializeComponent();
        }
        public Page2(string showID, string AfishaID)
        {
            show_id = showID;
            afisha_id = AfishaID;
            InitializeComponent();
            MakePage();
        }

        public async void MakePage()
        {
            page2Stack.Spacing = 2;
            string[] sources = { "photo1.jpg", "photo2.jpg", "photo3.jpg",
"photo4.jpg", "photo5.jpg", "photo6.jpg", "photo7.jpg", "photo8.jpg",
"photo9.jpg", "photo10.jpg" };
            Random rand = new Random();
            int randomImageInex = rand.Next(0, 10);
            Image showImage = new Image
            {
                Margin = new Thickness(0,0,0,0),
                Source = sources[randomImageInex],
                HorizontalOptions = LayoutOptions.Center,
            };

            ShowService ss = new ShowService();
            Show show = await ss.GetShow(show_id);
            Frame frameImage = new Frame
            {
                Margin = new Thickness(2, 3, 2, 5),
                BorderColor = Color.Black,
                Padding = new Thickness(2),
            };
            frameImage.Content = showImage;
            page2Stack.Children.Add(frameImage);

            Frame frame1 = new Frame()
            {
                Content = new Label()
                {
                    Text = "Балкон",
                    TextColor = Color.Black,
                    WidthRequest = 14,
                    Rotation = 0,
                    FontSize = 14,
                    FontAttributes = FontAttributes.Bold,
                    HorizontalOptions = LayoutOptions.Center,
                },
                BorderColor = Color.Black,
                HeightRequest = Content.Height + 20,
                WidthRequest = Content.Width + 15,
                Margin = new Thickness(50, 0, 25, 0),
            }
        }
    }
}

```

```

        Padding = new Thickness(5, 5, 5, 5)
    };

    Frame frame2 = new Frame
    {
        Content = new Label()
        {
            Text = "Папреп",
            TextColor = Color.Black,
            HeightRequest = 12,
            FontSize = 16,
            FontAttributes = FontAttributes.Bold,
            HorizontalOptions = LayoutOptions.Center,

        },
        BorderColor = Color.Black,
        HeightRequest = Content.Height + 30,
        WidthRequest = Content.Width + 100,
        Margin = new Thickness(0, 0, 0, 0),
        Padding = new Thickness(5, 5, 5, 5)
    };

    Frame frame3 = new Frame
    {
        Content = new Label
        {
            Text = "Амфитеатр",
            TextColor = Color.Black,
            FontSize = 16,
            FontAttributes = FontAttributes.Bold,
            HorizontalOptions = LayoutOptions.Center,
            VerticalOptions = LayoutOptions.Center,
        },
        BorderColor = Color.Black,
        HeightRequest = Content.Height + 30,
        WidthRequest = Content.Width + 150,
        Margin = new Thickness(0, 0, 0, 0),
        Padding = new Thickness(5, 5, 5, 5)
    };

    Frame frame4 = new Frame
    {
        Content = new Label
        {
            Text = "Бельэтаж",
            TextColor = Color.Black,
            FontSize = 16,
            FontAttributes = FontAttributes.Bold,
            HorizontalOptions = LayoutOptions.Center,
            VerticalOptions = LayoutOptions.Center,
        },
        BorderColor = Color.Black,
        HeightRequest = Content.Height + 30,
        WidthRequest = Content.Width + 150,
        Margin = new Thickness(0, 0, 0, 0),
        Padding = new Thickness(5, 5, 5, 5)
    };

    Frame frame5 = new Frame
    {
        Content = new Label
        {
            Text = "Ложа",
            Rotation = 0,

```

```

        FontSize = 14,
        TextColor = Color.Black,
        WidthRequest = 14,
        FontAttributes = FontAttributes.Bold,
        HorizontalOptions = LayoutOptions.Center,
    },
    BorderColor = Color.Black,
    HeightRequest = Content.Height + 20,
    WidthRequest = Content.Width + 15,
    Margin = new Thickness(25, 0, 50, 0),
    Padding = new Thickness(5, 5, 5, 5)
};

ObservableCollection<Frame> frames = new
ObservableCollection<Frame>();
frames.Add(frame1);
frames.Add(frame2);
frames.Add(frame3);
frames.Add(frame4);
frames.Add(frame5);
string selectedPlace = "";

foreach (var fr in frames)
{
    var taparrangement = new TapGestureRecognizer(); // клик по
рамке с расположением
=> taparrangement.Tapped += async (object sender, EventArgs e)
    {
        await Navigation.PushAsync(new BookingPlacesPage(fr,
afisha_id));
    };
    fr.GestureRecognizers.Add(taparrangement);
}

StackLayout verticalStack = new StackLayout();
verticalStack.Children.Add(frame3);
verticalStack.Children.Add(frame2);
verticalStack.Children.Add(frame4);

StackLayout horizontalStack = new StackLayout();
horizontalStack.Orientation = StackOrientation.Horizontal;
horizontalStack.Children.Add(frame1);
horizontalStack.Children.Add(verticalStack);
horizontalStack.Children.Add(frame5);
horizontalStack.Margin = new Thickness(0, 0, 0, 30);

Label lb = new Label
{
    Margin = new Thickness(15, 0, 0, 0),
    FontSize = 22,
    FontAttributes = FontAttributes.Bold,
    TextColor = Color.Black,
    Text = show.Name
}; // название
page2Stack.Children.Add(lb);
string[] parts = show.Info.Split(';');
Label lbl = new Label
{
    Text = "Автор: " + show.Author + "\nЖанр: " + show.Genre +
"\nВозраст: " + show.Age + "\nРежиссер: " + show.Rej + "\nCвет: " +
show.Light + "\nХореограф: " + show.Choreo + "\n" + parts[0] + "\n" +
parts[1],

```

```

        FontSize = 14,
        TextColor = Color.Black,
        Margin = new Thickness(30,0,0,0),
        FontAttributes = FontAttributes.Italic,
    };
    page2Stack.Children.Add (lb1);

    Label lb9 = new Label
    {
        Text="Актеры",
        FontSize = 18,
        FontAttributes= FontAttributes.Bold,
        Margin = new Thickness(15,0,0,0),
        TextColor = Color.Black,
    };
    page2Stack.Children.Add(lb9);

    ScrollView actorssv = new ScrollView();
    StackLayout actorssl = new StackLayout();
    var show_actors = await ss.GetActors(show_id);
    actorssl.Spacing = 0.5;
    foreach (var act in show_actors)
    {
        Label label = new Label
        {
            Text = act.FIO,
            Margin = new Thickness(30,0,0,0),
            TextColor = Color.SlateBlue,
            FontSize = 14,
            FontAttributes = FontAttributes.Italic,
        };
        actorssl.Children.Add(label);
        var tap1 = new TapGestureRecognizer();
        tap1.Tapped += (object sender, EventArgs e) =>
        {
            Navigation.PushAsync(new ActorInfoPage(act));
        };
        label.GestureRecognizers.Add(tap1);
    }
    actorssv.Content = actorssl;
    page2Stack.Children.Add(actorssv);

    Label lb8 = new Label
    {
        Text = "Выберите зону для бронирования билета",
        FontAttributes = FontAttributes.Bold,
        TextColor = Color.Black,
        Margin = new Thickness(15,0,0,10),
        FontSize = 18
    };

    page2Stack.Children.Add(lb8);
    page2Stack.Children.Add(horizontalStack);
    page2Stack.Children.Add(new Label());
}
}
}

```

ПРИЛОЖЕНИЕ В
ОПИСАНИЕ СУЩНОСТЕЙ
(рекомендуемое)

В таблице В.1 представлено описание атрибутов сущности «Актер».

Таблица В.1 – Описание атрибутов сущности «Актер»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор	Уникальный номер актера в таблице	Целый	4	1 – 9999	Нет	РК	1553	–
ФИО	Фамилия, имя и отчество актера	Символьный	100	Любые символы	Нет	–	Иван Солнцев	Формат ФИО предполагает наличие отчества, однако во многих случаях оно опущено
Фото	Путь до фотографии актера на сервере	Символьный	255	Любые символы	Да	–	C:/Photos/photo1.jpg	–
Биография	Краткая биография актера	Символьный	8000	Любые символы	Нет	–	Окончил ПГУ по специальности «Актерское мастерство»	–

В таблице В.2 представлено описание атрибутов сущности «Спектакль».

Таблица В.2 – Описание атрибутов сущности «Спектакль»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор	Уникальный номер спектакля в таблице	Целый	4	1 – 9999	Нет	РК	1553	–
Название	Название спектакля	Символьный	100	Любые символы	Нет	–	Айболит	–
Автор	ФИО автора спектакля	Символьный	100	Любые символы	Нет	–	К. Чуковский	–
Постановка и музыкальное оформление	Дополнительная информация о постановке спектакля	Символьный	1000	Любые символы	Нет	–	Художественное оформление – Евгений Петров;	–

Продолжение таблицы В.2

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
ЖанрId	Идентификатор жанра спектакля	Целый	4	1–9999	Нет	FK	3	–
ВозрастId	Идентификатор допустимого для просмотра спектакля возраста	Целый	4	1–9999	Нет	FK	1	–

В таблице В.3 представлено описание атрибутов сущности «Жанр».

Таблица В.3 Описание атрибутов сущности «Жанр»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор	Уникальный номер жанра в таблице	Целый	4	1 – 9999	Нет	PK	1553	–
Название	Словесное представление жанра спектакля	Символьный	100	Любые символы	Нет	–	Драма Комедия	–

В таблице В.4 представлено описание атрибутов сущности «Возраст».

Таблица В.4 Описание атрибутов сущности «Возраст»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор	Уникальный номер возрастного ограничения в таблице	Целый	4	1 – 9999	Нет	PK	1553	–
Возраст	Символьное представление допустимого возраста	Символьный	3	09+	Нет	–	3+ 16+	–

В таблице В.5 представлено описание атрибутов сущности «Режиссер».

Таблица В.5 Описание атрибутов сущности «Режиссер»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор	Уникальный номер режиссера в таблице	Целый	4	1 – 9999	Нет	РК	1553	–
ФИО	ФИО режиссера спектакля	Символьный	100	Любые символы	Нет	–	Алексей Иванович Носов	–

В таблице В.6 представлено описание атрибутов сущности «Свет».

Таблица В.6 Описание атрибутов сущности «Свет»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор	Уникальный номер группы света в таблице	Целый	4	1 – 9999	Нет	РК	1553	–
Название	Название команды, отвечающей за свет в спектакле	Символьный	100	Любые символы	Нет	–	Группа "Светофор"	–

В таблице В.7 представлено описание сущности «Хореограф».

Таблица В.7 Описание атрибутов сущности «Хореограф»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор	Уникальный номер хореографа в таблице	Целый	4	1 – 9999	Нет	РК	1553	–
ФИО	ФИО хореографа для спектакля	Символьный	100	Любые символы	Нет	–	Захарова Елена Петровна	–

В таблице В.8 представлено описание атрибутов сущности «Распределение актеров».

Таблица В.8 Описание атрибутов сущности «Хореограф»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор	Уникальный номер записи о распределении в таблице	Целый	4	1 – 9999	Нет	PK	1553	–
АктерId	Идентификатор актера в распределении	Целый	4	19999	Нет	FK	112	–
СпектакльId	Идентификатор спектакля в распределении	Целый	4	19999	Нет	FK	564	–

В таблице В.9 представлено описание атрибутов сущности «Афиша».

Таблица В.9 Описание атрибутов сущности «Афиша»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор	Уникальный номер афиши в таблице	Целый	4	1 – 9999	Нет	PK	1553	–
Дата	Дата проведения спектакля	Дата	8	01.01.2023 31.12.2050	Нет	–	2023 – 05 – 20	Дата указывается в формате год / месяц / день
Время	Время проведения спектакля	Время	6	00:00:00 23:59:59	Нет	–	11:00:00 23:05:00	Время указывается в формате час : минуты: секунды
СпектакльId	Идентификатор сущности «Спектакль»	Целый	4	1 9999	Нет	–	510	–

В таблице В.10 представлено описание атрибутов сущности «Место».

Таблица В.10 – Описание атрибутов сущности «Место»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор	Уникальный номер места в таблице	Целый	4	1 – 9999	Нет	PK	1553	–
Номер места	Номер места в указанном ряду с указанным расположением	Целый уменьшенный	3	1 255	Нет	–	510	–
Ряд	Номер ряда в указанном расположении	Целый уменьшенный	6	1 255	Нет	–	110	–
РасположениеId	Идентификатор сущности «Расположение»	Целый	4	1 9999	Нет	FK	14	–

В таблице В.11 представлено описание атрибутов сущности «Расположение».

Таблица В.11 Описание атрибутов сущности «Расположение»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор	Уникальный номер расположения в таблице	Целый	4	1 – 9999	Нет	PK	1553	–
Название	Словесное название расположения	Символьный	100	Все слова, состоящие из букв русского алфавита	Нет	–	Ложа Партер	–

В таблице В.12 представлено описание атрибутов сущности «Пользователь».

Таблица В.12 Описание атрибутов сущности «Пользователь»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор	Уникальный номер пользователя в таблице	Целый	4	1 – 9999	Нет	PK	1553	–
Логин	Логин пользователя	Символьный	20	Любые символы	Нет	–	Login123	Используется для входа в систему
Пароль	Пароль пользователя	Символьный	255	Любые символы	Нет	–	Password123	Используется для входа в систему
Фото	Путь до фотографии профиля пользователя	Символьный	255	Любые символы	Да	–	C:/Users/Photos/Photo1.jpg	–
ФИО	ФИО пользователя	Символьный	100	Любые символы	Да	–	Иванов Иван Иванович	–
Права		Целый уменьшенный	3	1 3	Нет	–	1	Используется для разделения прав между пользователями

В таблице В.13 представлено описание атрибутов сущности «Бронь».

Таблица В.13 Описание атрибутов сущности «Бронь»

Наименование	Описание	Тип данных	Размерность	Область допустимых значений	Возможность значения NULL	Роль	Пример	Примечание
Идентификатор	Уникальный номер записи о бронировании в таблице	Целый	4	1 – 9999	Нет	PK	1553	–
АфишаId	Идентификатор сущности «Афиша»	Целый	4	1 – 9999	Нет	FK	1553	–
ПользовательId	Идентификатор сущности «Пользователь»	Целый	4	1 – 9999	Нет	FK	1553	–
МестоId	Идентификатор сущности «Место»	Целый	4	1 – 9999	Нет	FK	1553	–

ПРИЛОЖЕНИЕ Г
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ
(рекомендуемое)

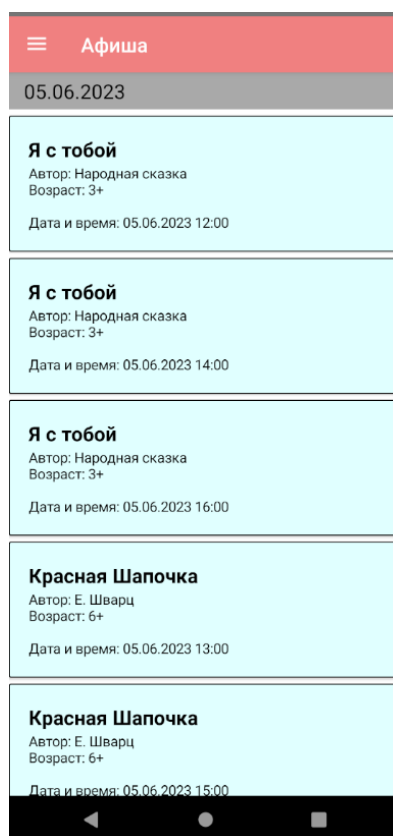


Рисунок Г.1 – Результат теста №1

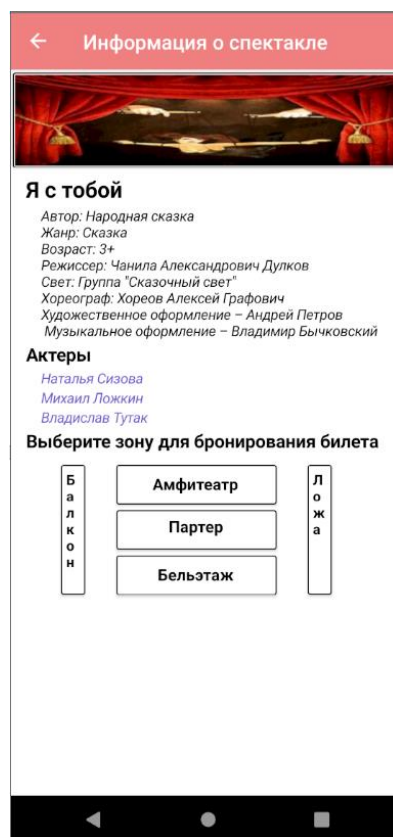


Рисунок Г.2 – Результат теста №2



Рисунок Г.3 – Результат теста №3

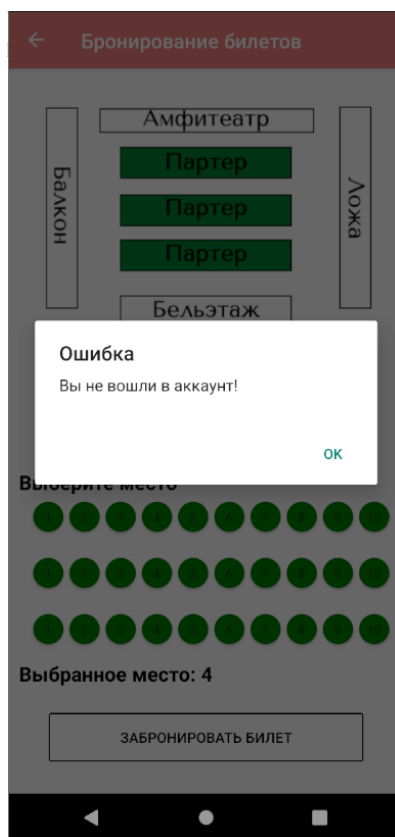


Рисунок Г.4 – Результат теста №4

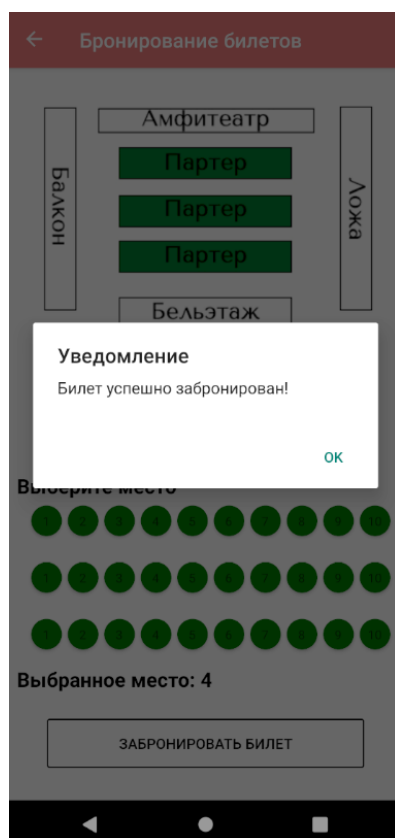


Рисунок Г.5 – Результат теста №5

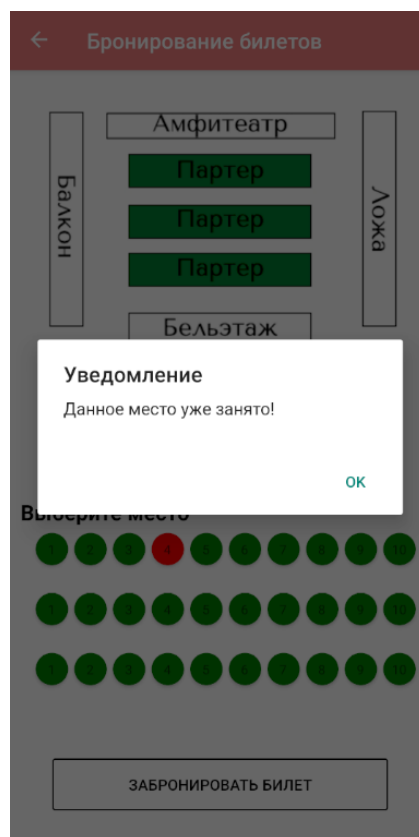


Рисунок Г.6 – Результат теста №6

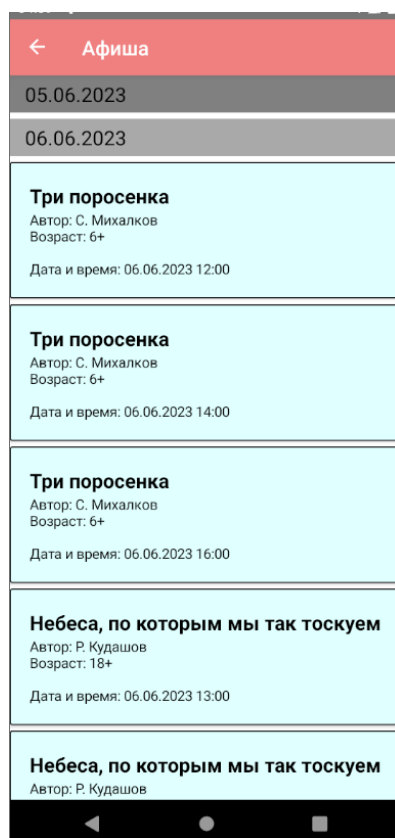


Рисунок Г.7 – Результат теста №7



Рисунок Г.8 – Результат теста №8

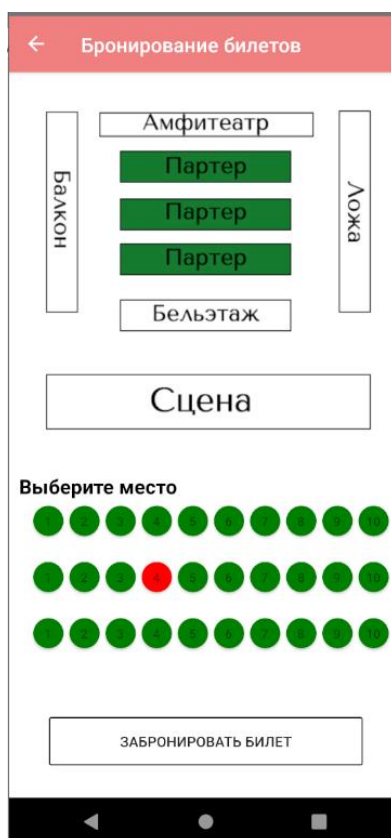


Рисунок Г.9 – Места в спектакле до отмены бронирования

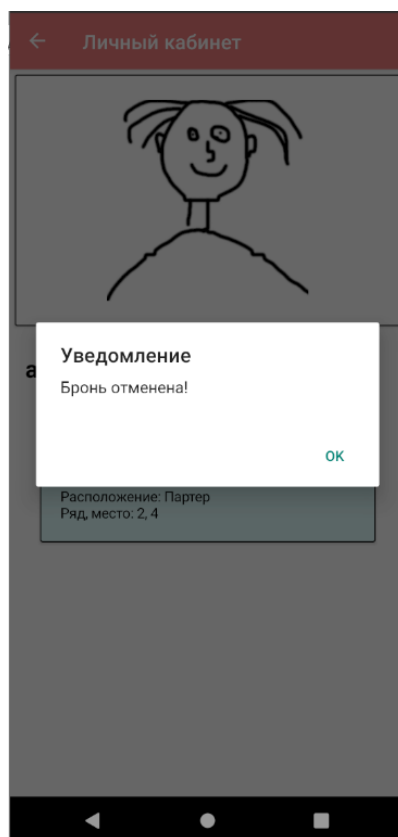


Рисунок Г.10 – Уведомление об отмене бронирования



Рисунок Г.11 – Места в спектакле после отмены бронирования