

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

Кафедра Автоматизированных Систем Управления

К защите

Руководитель работы:

дата, подпись

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

по дисциплине

«Технологии программирования»

Тема:

«Разработка информационно-поисковой системы «Организация концертов»

Выполнил студент группы 135

Шандала С.Д.

дата сдачи на проверку, подпись

Руководитель работы

Преподаватель каф. АСУ

Аникеев Д.В.

оценка

дата защиты, подпись

Рязань 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
Рязанский государственный радиотехнический университет
имени В. Ф. Уткина
Кафедра Автоматизированных Систем Управления

З А Д А Н И Е

на курсовую работу по дисциплине
"Технологии программирования"

Студент — Шандала Сергей Дмитриевич группа — 135

1. Тема: Разработка информационно-поисковой системы «Организация концертов»

2. Срок представления работы к защите - «7» июня 2023г.

3. Исходные данные для проектирования:

3.1. Язык программирования – С#.

3.2. СУБД — MS SQL Compact Edition, MS SQL Express Edition

3.3. Технология объектного связывания данных – Entity Framework.

3.4. Общее количество таблиц - не менее 5.

3.5. Обязательные операции с данными — просмотр записей таблиц в виде списка, сортировка, поиск, добавление, изменение, удаление записей.

3.6. Все страницы сайт должны быть наполнены осмысленной информацией.

4. Содержание курсовой работы

4.1. Задание.

4.2. Пояснительная записка, оформленная по ГОСТ 19.404-79, содержащая:

4.2.1. Описание предметной области.

4.2.2. Структурную схему модели данных.

4.2.3. Структурную схему программы.

Руководитель работы _____ / Аникеев Д.В.

(подпись)

Задание принял к исполнению _____ « 14 » февраля 2023г.

(подпись)

Содержание

Введение.....	4
1 Описание предметной области	5
1.1 Семантическое описание предметной области	5
1.2 Use Case модель.....	7
1.3 BPMN модель	9
2 Сравнительный анализ существующих решений.....	12
3 Разработка системы	17
3.1 Архитектура информационной системы	17
3.2 Диаграмма сущностей.....	19
3.3 Создание базы данных	23
3.4 Data Access Layer	26
3.5 Слой бизнес логики – Domain.....	31
3.6 Пользовательский интерфейс.....	35
4 Тестирование системы	44
Заключение	55
Список используемых источников.....	56

Введение

Организация концертов является важным событием в культурной жизни общества [1]. Сегодня на рынке массовых мероприятий существует множество организаций, занимающихся проведением концертов, фестивалей и других мероприятий. Однако, для успешной реализации проекта, требуется управлять не только культурной частью, но и делами, связанными с организацией мероприятия[2]. В этом контексте создание информационной системы помогает оптимизировать организационные процессы[3].

В данной курсовой работе разработана информационная система для организации концертов и рассмотрены основные этапы разработки, такие как проектирование структуры базы данных, создание пользовательского интерфейса, реализация функционала, а также тестирование разработанной системы.

Необходимо разработать Use Case и BPMN модели, а так же диаграмму сущностей. Выполнить сравнительный анализ в поисках подходящих альтернатив. Определиться с архитектурой информационной системы, создать базу данных, написать программы для предоставления пользователю возможности работать с ней и провести тестирование системы.

1 Описание предметной области

1.1 Семантическое описание предметной области

В жизни любого человека важное место занимает отдых. Отдых бывает разный, одним из его видов служат концерты тех исполнителей, которые ему нравятся. Концерт — это публичное исполнение музыкальных произведений, мероприятие цель которого при нести радость и душевное удовлетворение пришедшим на него людям. Концерты могут иметь самую разную аудиторию, и сейчас разнообразие жанров и исполнителей настолько велико, что для любой аудитории найдется артист, отвечающий ее требованиям[4].

Но чтобы концерт состоялся его нужно организовать. Обычно у самих артистов нет времени самим организовывать свои выступления, ведь это довольно трудоемкая задача, в которой нужно учитывать много мелких деталей[5]. Организация концертов – это сложный процесс, требующий вовлечения профессиональных специалистов с опытом работы. Как правило, подготовкой подобных мероприятий занимается специальный человек – концертный менеджер[6]. Концертные менеджеры бывают разные. Есть те, кто работают только с одним артистом и организуют все его выступления во всех городах куда он едет. А есть те, кто организует много концертов разных артистов в одном городе. Именно эти менеджеры далее и будут рассматриваться. Обычно эти менеджеры не работают напрямую с артистом, а взаимодействуют с рассмотренными выше менеджерами, прикрепленными к конкретному артисту. Далее менеджер, организующий концерты в конкретном городе, будет именоваться - концертный менеджер, а менеджер, связанный с одним артистом - менеджер артиста. Вариантов событий может быть два[7]:

- 1) концертный менеджер хочет организовать концерт и пригласить артиста, тогда он связывается с менеджером артиста, они обсуждают условия и потом, если приходят к согласию, концертный менеджер организует концерт артиста в своем городе;

2) инициатива организовать концерт исходит от артиста, менеджер артиста связывается с концертным менеджером, а дальше все идет по той же схеме, что и в первом случае. Отличается лишь инициатор действия.

Задач и требований к организации концерта много, и все они подразумевают скрупулёзную проработку, учет всех нюансов. Концертному менеджеру нужно найти подходящую сцену, нужное оборудование, необходимых работников, организовать рекламную кампанию, позаботиться о безопасности мероприятия[8]. Необходимо учитывать, что проведение концертов начинает планироваться задолго до предполагаемой даты его проведения, чтобы все можно было грамотно организовать. За это время менеджер должен в соответствии с техническим райдером, выдвинутым артистом найти соответствующие его требованиям концертную площадку и оборудование. Договориться с командой технических специалистов, если они необходимы. Организовать рекламную кампанию мероприятия. Посчитать предполагаемые расходы и прибыль и исходя из этого подбирать все необходимое[9].

Размер затрат на организуемый концерт зависят от популярности артиста[10]. Также необходимо четко понимать, что это за артист, для какой аудитории он выступает, что он исполняет, где и при каких условиях его выступление принесет наибольшую прибыль, и стоит ли вообще работать с этим артистом. Кроме подробного коммерческого плана также понадобится расписанный сценарий планируемого концерта. Концертный менеджер должен контролировать весь процесс подготовки к концерту и проведение самого концерта. После его завершения необходимо рассчитаться со всеми финансовыми обязательствами.

1.2 Use Case модель

Назначение Use Case диаграмм[11] состоит в следующем: проектируемая информационная система представляется в форме так называемых вариантов использования, с которыми взаимодействуют внешние сущности или акторы. При этом актором или действующим лицом называется любой объект, субъект или система, взаимодействующая с моделируемой бизнес-системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая служит источником воздействия на моделируемую систему так, как определит разработчик. Вариант использования служит для описания сервисов, которые система предоставляет актору. Другими словами, каждый вариант использования определяет набор действий, совершаемый системой при диалоге с актором. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие акторов с системой и собственно выполнение вариантов использования.

Описание объектов и их взаимодействия в USE CASE диаграмме:

- «зритель» посещает концерт для получения удовольствия. Чтобы посетить концерт он должен купить билет, а так же на самом концерте он может приобрести фирменные товары артиста;
- «промоутер» занимается рекламой концерта и разработкой маркетинговой стратегии для этого. Он может организовать какое-то взаимодействие артиста со зрителями вне мероприятия в рекламных целях если считает, что это принесет пользу;
- «артист» выступает на концерте, для правильной организации которого он составляет и отправляет организатору технический райдер. Вне концерта для создания дополнительного ажиотажа он может участвовать в рекламе и различных взаимодействиях с потенциальными зрителями;
- «концертный работник» может заниматься самыми различными задачами. Он может обслуживать сцену, настраивать оборудование и следить

за ним, а так же продавать билеты, следить за порядком и обслуживать зрителей;

— «владелец сцены» предоставляет сцену, работников и оборудование для концерта;

— «концертный менеджер» организует концерт, ищет подходящих работников и промоутеров, выбирает сцену, всячески работает с артистом и всевозможными партнерами, управляет бюджетом и контролирует процесс проведения мероприятия.

Диаграмма представлена на рисунке 1.

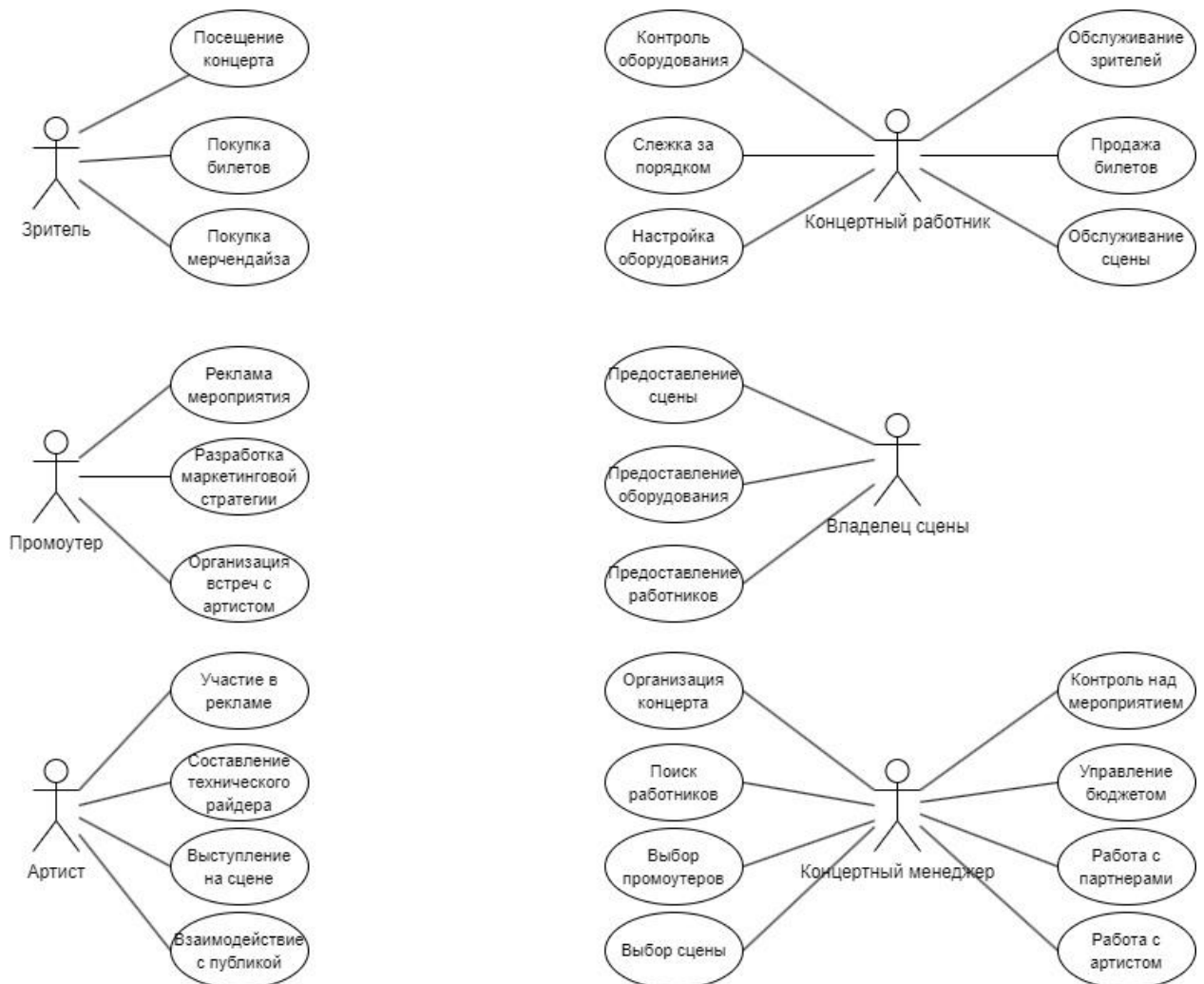


Рисунок 1 – Use Case диаграмма

1.3 BPMN модель

Данная диаграмма отображает бизнес-процессы с помощью блок-схем. Она показывает в какой последовательности совершаются рабочие действия и перемещаются потоки информации[12].

Описание процесса организации концерта (рисунок 2):

- 1) получение предложения от артиста (чаще от его менеджеров, но на схеме артист и все его представители обобщаются);
- 2) с помощью информационной системы, в которую записаны артисты проверить благонадежность того, который хочет организовать концерт;
 - 2.1) артист неблагонадежен – отказать в организации;
 - 3) если артист благонадежен узнать о требованиях артиста;
 - 3.1) если не сошлись в требованиях – отказать в организации;
 - 4) с помощью информационной системы, содержащей в себе данные о ценах, площадках и т.д. подобрать подходящие условия;
 - 5) узнать устраивают ли артиста условия;
 - 6) если артиста условия не устраивают ли узнать о перспективах дальнейшего сотрудничества;
 - 6.1) перспектив нет – прервать процесс организации;
 - 6.2) перспективы есть – возврат к пункту 5;
 - 7) если артиста все устраивает провести рекламную кампанию и организовать подготовку к концерту;
 - 8) возможное неожиданное происшествие, которое помешало проведению концерта;
 - 9) действовать в зависимости от обстоятельств (ввиду многообразия возможных причин срыва концерта не имеет смысла подробно останавливаться на какой-либо из них);
 - 10) если ничего плохого не случилось – проконтролировать проведение концерта;
 - 11) выполнить все финансовые договоренности и получить прибыль или же понести убытки по ряду различных причин.

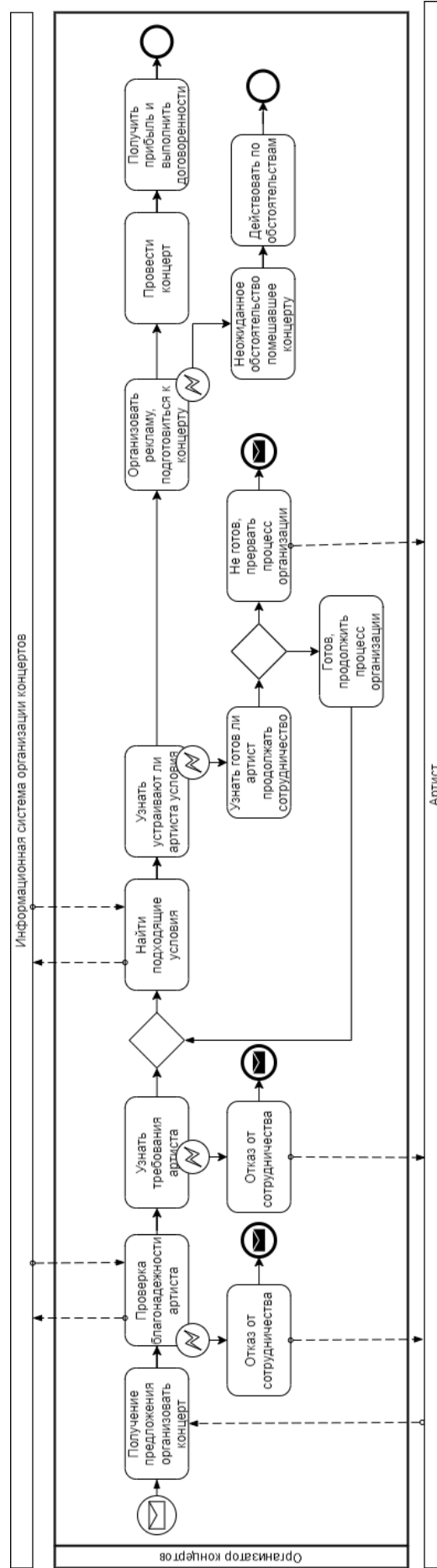


Рисунок 2 – BPMN диаграмма организации концерта

Описание процесса подготовки к концерту (рисунок 3):

- 1) получить от артиста (или его представителей) технический райдер;
- 2) найти работников сцены;
- 2.1) получить предложение организовать концерт (для работников сцены);
- 3) совместно спланировать мероприятие;
- 4) подготовить рекламу (для организатора);
- 5) подготовить техническую часть (для работников);
- 6) провести концерт (для работников);
- 6.1) проконтролировать ход концерта (для организатора);
- 7) рассчитаться с работниками (для организатора);
- 7.1) получить плату (для работников).

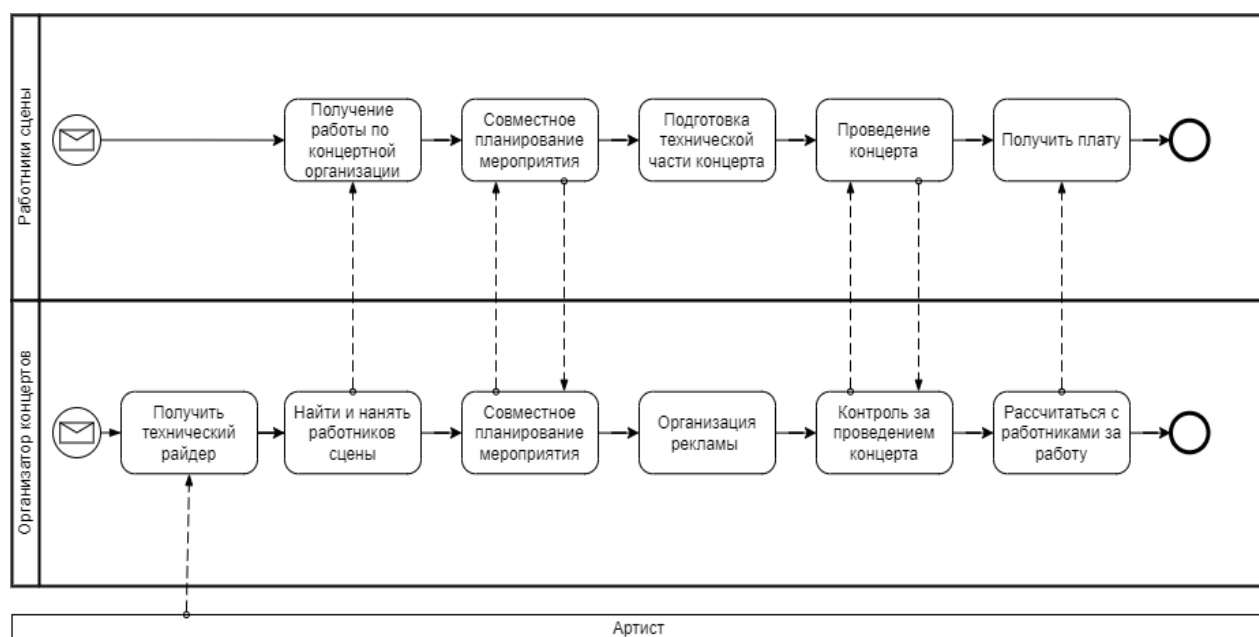


Рисунок 3 – BPMN диаграмма подготовки к концерту

2 Сравнительный анализ существующих решений

Существует множество решений для организации концертов[13], и каждое из них имеет свои преимущества и недостатки. Некоторые из них:

1. Специализированные агентства[14] по организации концертов. Эти агентства специализируются на организации концертов и обладают большим опытом и знаниями в этой области. Они могут предложить полный спектр услуг, включая поиск места проведения концерта, аренду оборудования, продвижение и рекламу, продажу билетов и т.д. Однако, такие услуги могут быть довольно дорогими, и они могут не подходить для всех. В таких агентствах работают менеджеры[15] по организации концертов. Это люди, которые занимаются поиском исполнителей, выбором места проведения концерта, продажей билетов и другими организационными вопросами. Они предлагают все услуги, связанные с организацией концертов, и могут взять на себя все аспекты организации. Однако, их услуги могут оказаться дорогими для клиента, и они могут быть не настолько эффективными, как клиенты ожидали.

2. SaaS-платформы[16] для организации концертов, такие как Ticketmaster, StubHub и Eventbrite. Они позволяют организовывать концерты и продавать билеты онлайн и предоставляют полный набор инструментов для организации концерта, от поиска исполнителей до продажи билетов и управления маркетингом. Эти платформы имеют широкую аудиторию и могут быть полезными для продвижения концертов также они могут быть гораздо более доступными и удобными, чем услуги агентств. Они также помогают клиентам управлять финансами, персоналом и графиком мероприятий[17]. Однако они могут брать высокие комиссии за продажу билетов, что может увеличить стоимость концерта для клиентов, а так же самостоятельная организация концерта может быть трудной и требовать много времени и усилий.

Далее описаны три SaaS платформы, разобраны их преимущества и недостатки.

Платформа Ticketmaster[18] основана на множестве технологий и систем, которые позволяют ей обрабатывать миллионы билетов, обработать миллионы транзакций по продаже билетов и общаться с миллионами клиентов ежедневно. Основным компонентом Ticketmaster является веб-сайт, доступный всем пользователям через браузер. Веб-сайт позволяет пользователям искать мероприятия, выбирать места и покупать билеты. Кроме этого, Ticketmaster имеет мобильное приложение, которое позволяет пользователям искать мероприятия, покупать билеты и получать уведомления о предстоящих мероприятиях, а так же более 6 тысяч билетных касс по всему миру, где пользователи могут покупать билеты в реальном времени.

Ticketmaster имеет собственную систему обработки платежей, которая позволяет пользователям покупать билеты с помощью кредитных карт, PayPal и других способов оплаты. Для проверки подлинности билетов Ticketmaster использует технологию SafeTix[19], которая позволяет проверять подлинность билетов через мобильное приложение или веб-сайт. Система защиты и безопасности Ticketmaster использует различные меры безопасности, включая SSL-шифрование[20], генерацию уникальных кодов и многоуровневые системы проверки подлинности.

Также Ticketmaster предоставляет API[21] для разработчиков, которые могут использовать его для интеграции с другими приложениями или сайтами. API позволяет получать информацию о мероприятиях, продавать билеты и управлять заказами.

В целом, Ticketmaster представляет собой комплексную техническую систему, которая объединяет множество различных компонентов для обеспечения эффективной продажи билетов и управления клиентскими данными. Особенность Ticketmaster — это широкие возможности для персонализации, в том числе автоматические подборки событий на основе предпочтений клиента, уведомления о будущих мероприятиях и онлайн-поддержка по всем вопросам, связанным с билетами и заказами. Кроме того, Ticketmaster имеет широкое партнерство с другими организациями по всему

миру, что позволяет клиентам купить билеты на множество мероприятий в одном месте.

StubHub[22] — это глобальная платформа, предназначенная для покупки и продажи билетов на различные культурные и спортивные мероприятия. StubHub предоставляет клиентам широкий выбор билетов на различные мероприятия, такие как концерты, спортивные события, музыкальные фестивали, театральные представления и другие. StubHub является онлайн-рынком, соединяющим покупателей и продавцов билетов. Клиенты на StubHub могут покупать билеты напрямую у продавцов, а продавцы имеют возможность продать свои билеты имея доступ к огромному рынку потенциальных покупателей. StubHub также предлагает широкие возможности для перепродажи билетов, которые могут быть проданы в одном месте на любую доступную цену.

StubHub разработан на основе языка программирования Java[23] и использованием системы управления базами данных Oracle[24]. Система управления билетами разработана на основе технологий Spring MVC[25] и Apache Struts[26] для обеспечения более высокой производительности и эффективности.

Основным компонентом StubHub, доступным всем пользователям через браузер, как и в случае с Ticketmaster является веб-сайт, который позволяет пользователям искать мероприятия, выбирать места и покупать билеты. Так же имеется мобильное приложение, которое позволяет пользователям искать мероприятия, покупать билеты и получать уведомления о предстоящих мероприятиях.

StubHub имеет собственную систему обработки платежей, которая позволяет пользователям покупать билеты с помощью кредитных карт, PayPal и других способов оплаты. Для проверки подлинности билетов StubHub имеет систему, которая позволяет подтвердить адрес электронной почты клиента для обеспечения безопасной покупки билетов.

Кроме этого, StubHub использует аналитические инструменты, чтобы собирать данные о пользовательском поведении и оптимизировать работу платформы. Так же платформа интегрирована с социальными сетями, что позволяет пользователям легко делиться информацией о мероприятиях и билетах с друзьями.

Подобно платформе Ticketmaster платформа StubHub предоставляет API для разработчиков, которые могут использовать его для интеграции с другими приложениями или сайтами. API позволяет получать информацию о мероприятиях, продавать билеты и управлять заказами.

В целом, технологические компоненты StubHub направлены на обеспечение максимально простого и эффективного процесса покупки билетов и управления заказами для клиентов.

Eventbrite[27] — это платформа, предназначенная для онлайн-организации, продажи билетов и управления мероприятиями. Она предоставляет широкие возможности для создания, продвижения и продажи билетов на большинство видов мероприятий, таких как концерты, фестивали, семинары, вебинары, ярмарки и другие.

Eventbrite позволяет организаторам создавать и настраивать страницы мероприятий, добавлять описание, даты, время и цены билетов, предоставляет инструменты для настройки и управления билетами, в том числе для создания комбинированных билетов, настройки купонов и скидок, а также уведомлений о продажах и доступности билетов. Клиенты могут купить билеты на мероприятие через онлайн-платформу Eventbrite, используя кредитные карты, PayPal или другие методы оплаты.

Eventbrite имеет веб-сайт позволяет пользователям искать мероприятия, выбирать места и покупать билеты и мобильное приложение, которое позволяет пользователям искать мероприятия, покупать билеты и получать уведомления о предстоящих мероприятиях. Для проверки подлинности билетов Eventbrite использует технологию Barcode Scanner[28], которая

позволяет проверять подлинность билетов через мобильное приложение или веб-сайт.

Eventbrite отслеживает продажи и метрики, такие как количество проданных билетов, посетителей страницы мероприятия и другие, что позволяет организаторам мероприятия оценить эффективность своих продаж и рассчитать бюджет на будущие мероприятия. К тому же платформа интегрирована с социальными сетями.

Подобно предыдущим двум платформам Eventbrite предоставляет API для разработчиков.

Eventbrite — это развитая технологическая сервисная платформа, которая позволяет организаторам мероприятий, покупателям билетов и другим участникам событий находить и подключаться к нужным мероприятиям в целом мире.

Проанализировав все три платформы можно сделать вывод что все они представляют масштабные и сложные информационные системы, обладающие широким спектром возможностей и предоставляющие множество полезных и удобных инструментов. Наличие таких платформ, а так же их популярность позволяет утверждать, что в контексте предметной области разработка информационной системы является выгодным решением, удовлетворяющим потребности, как зрителей, так и артистов с менеджерами, поскольку она поможет автоматизировать множество процессов, связанных с организацией концертов и других массовых мероприятий.

3 Разработка системы

3.1 Архитектура информационной системы

В соответствии с ранее упомянутыми схемами и диаграммами можно начать разработку системы. В данной работе система будет реализована с помощью трехзвенной архитектуры[29] приложения. Трехзвенная архитектура состоит из четырех слоев.

Первым слоем является слой представления, расположенный на уровне клиента. На этом слое реализуется пользовательский интерфейс. Он отвечает за взаимодействие с пользователем и отображение данных в удобном для восприятия формате. Основная задача слоя представления – это преобразование и передача информации между пользователем и другими слоями приложения. Основным принцип слоя представления – это отделение пользовательского интерфейса от бизнес-логики приложения. Это позволяет повысить гибкость и масштабируемость приложения, а также облегчает его поддержку и сопровождение в долгосрочной перспективе.

Вторым слоем является слой бизнес логики или же domain, расположенный на уровне сервера приложений. Этот слой отвечает за выполнение бизнес-логики приложения. Он представляет собой набор классов и методов, которые выполняют основные операции над данными и обеспечивают правильность и целостность данных. Domain может включать в себя различные модули и компоненты, которые обеспечивают работу приложения в соответствии с бизнес-требованиями. Одним из принципов слоя бизнес логики является отделение данных и их обработки, т.е. данные хранятся и обрабатываются независимо от того, как они визуализуются пользователю. Кроме того, слой бизнес логики должен быть написан таким образом, чтобы его можно было использовать другими частями системы. Это позволяет создавать новые функциональные модули и сервисы, используя единую бизнес-логику. В целом, слой бизнес логики является важнейшим компонентом любого приложения, который обеспечивает его работу в соответствии с требованиями бизнеса и надежностью в долгосрочном периоде.

Третьим слоем является слой доступа к данным или же DAL, расположенный на уровне сервера приложений. Этот слой управляет доступом к данным в приложении и обеспечивает взаимодействие между приложением и системой хранения данных. Он предоставляет интерфейс для выполнения операций чтения, записи, изменения и удаления данных. Слой доступа к данным включает в себя различные компоненты, такие как модели данных и объекты доступа к данным. Одним из основных принципов слоя доступа к данным является разделение логики работы приложения и логики доступа к данным. Это позволяет создавать приложения, которые могут использовать различные системы хранения данных без необходимости изменять код приложения. Кроме того, это облегчает единовременную замену системы хранения данных без влияния на логику приложения.

Четвертым слоем является слой хранения данных, расположенный на уровне сервера БД. Слой хранения данных – это компонент приложения, который отвечает за сохранение данных в постоянном хранилище и их управление. Слой хранения данных может использоваться вместе с другими компонентами, например, со слоем доступа к данным и слоем бизнес-логики, чтобы обеспечить правильное функционирование приложения. Одним из основных принципов слоя хранения данных является выбор правильного типа хранилища данных в соответствии с требованиями приложения. Это может быть реляционная база данных, NoSQL-хранилище, файловое хранилище и т.д. Выбор типа хранилища зависит от требований приложения.

Получившуюся архитектуру можно изобразить в виде диаграммы компонентов с помощью языка UML[30] (рисунок 4)



Рисунок 4 – Диаграмма компонентов

Теперь можно приступить к определению сущностей предметной области и разработке соответствующей диаграммы.

3.2 Диаграмма сущностей

Диаграмма сущностей (или ER-диаграмма)[31] в базах данных используется для визуализации структуры данных, которые используются в системе. Она позволяет описать сущности в системе и отношения между ними, а также описать атрибуты каждой сущности. Главная цель диаграммы сущностей — это создание единого языка для коммуникации между разработчиками и заказчиками в процессе разработки системы.

Для разработки информационной системы организации концертов была создана ER диаграмма, которая показывает предполагаемую для программной реализации структуру базы данных[32]. Всего получилось девять сущностей.

Сущность «Концерт». В данной таблице хранится информация о предстоящих концертах. Каждый концерт имеет свой уникальный идентификатор. Для каждого концерта должны быть указаны: название, дата проведения, минимальная стоимость билетов, предполагаемое количество зрителей, идентификатор сцены, на которой он должен проводиться и идентификатор выступающего на данном концерте артиста.

Сущность – Артист. В данной таблице хранится информация об артистах. У каждого артиста есть свой уникальный идентификатор. Для каждого артиста должны быть указаны: ФИО артиста и менеджера артиста, любой из контактов для связи с менеджером(телефон, почта и т.д.), размер требуемого артистом гонорара и абстрактный индекс популярности.

Сущность – Технический райдер. В данной таблице хранится информация о различных технических райдерах. Каждый технический райдер имеет свой уникальный идентификатор. Для каждого технического райдера должны быть указаны: название концерта, на котором он будет использоваться, идентификатор артиста, заявившего данный райдер, требуемый размер сцены, количество необходимого оборудования в единицах, требования к спецэффектам и декорациям, максимальная сумма возможных расходов, которую артист готов потратить на подготовку к концерту и необходимое для проведения количество работников.

Сущность – Сцена. В данной таблице хранится вся информация о концертных площадках, которые можно использовать для выступления. Каждая сцена имеет свой уникальный идентификатор. Для каждой сцены должны быть указаны: название, цена за выступление на ней, ее размер, количество звукового и светового оборудования, наличие спецэффектов и декораций, а также количество мест для зрителей.

Сущность – Оборудование. В данной таблице хранится информация об оборудовании, имеющемся на определенной сцене. Каждое такое оборудование имеет свой уникальный идентификатор. Для каждого оборудования должны быть указаны: номер сцены, на которой используется данное оборудование, место хранения, количество, тип оборудования и его цена.

Сущность – Рекламное предложение. В данной таблице хранятся рекламные предложения с указанием включенных в это предложение типов рекламы. Каждое рекламное предложение имеет свой уникальный идентификатор. Для каждого рекламного предложения должны быть указаны: цена этого рекламного приложения, время в течении которого данное рекламное предложение будет рекламировать мероприятие, а так же однозначные ответы на вопросы: есть ли реклама на/в интернете/телевиденье/радио/физическом виде.

Сущность – Рекламная кампания. В данной таблице хранятся данные о рекламных кампаниях. Каждая рекламная кампания имеет свой уникальный идентификатор. Для каждой рекламной кампании должны быть указаны: дата ее начала, идентификатор рекламного предложения, которое используется в данной кампании и идентификатор концерта, для которого реализуется рекламная кампания.

Сущность – Концертный работник. В данной таблице хранится информация о всех работниках, которые могут быть задействованы в концерте. Каждый работник имеет свой уникальный идентификатор. Для

каждого работника должны быть указаны: Его ФИО, оклад, специальность, квалификация, опыт работы и номер телефона.

Сущность – Занятость концертного работника. В данной таблице хранятся данные о занятости того или иного работника в том или ином концерте. Каждая такая занятость имеет свой уникальный идентификатор. Для каждой занятости должны быть указаны: время работы, идентификатор работника и идентификатор концерта, в котором работник участвует.

Для того чтобы связать эти сущности необходимо установить отношения между ними.

Одна сцена может использоваться во многих концертах поэтому отношение сцена-концерт равно один ко многим, при этом на сцене могут и не проходить концерты. Так же на одной сцене может быть много оборудования, поэтому отношение сцена-оборудование равно один ко многим, при этом у сцены может не быть оборудования.

Один артист может выступать на множестве концертов и в то же время может использовать много технических райдеров. Поэтому отношение артист-концерт равно один ко многим и отношение технический артист-технический райдер равно один ко многим, при этом у артиста может не быть концертов и технических райдеров.

У одного концерта может быть много рекламных кампаний, а также одно рекламное предложение может использоваться во многих рекламных кампаниях. Поэтому отношение концерт-рекламная кампания равно один ко многим и отношение рекламное предложение - рекламная кампания равно один ко многим, при этом у концерта может не быть рекламы и рекламное предложение в конкретный момент может быть не задействовано ни в одной рекламной кампании.

В одном концерте может быть задействовано много работников, и один работник может быть задействован в множестве разных концертов. Поэтому отношение концерт- занятость концертного работника равно один ко многим и отношение концертный работник - занятость концертного работника равно

один ко многим, при этом у концерта должен быть хотя бы один задействованный работник, а конкретный работник может быть не задействован ни в одном концерте.

Получившаяся диаграмма сущностей представлена на рисунке 5.

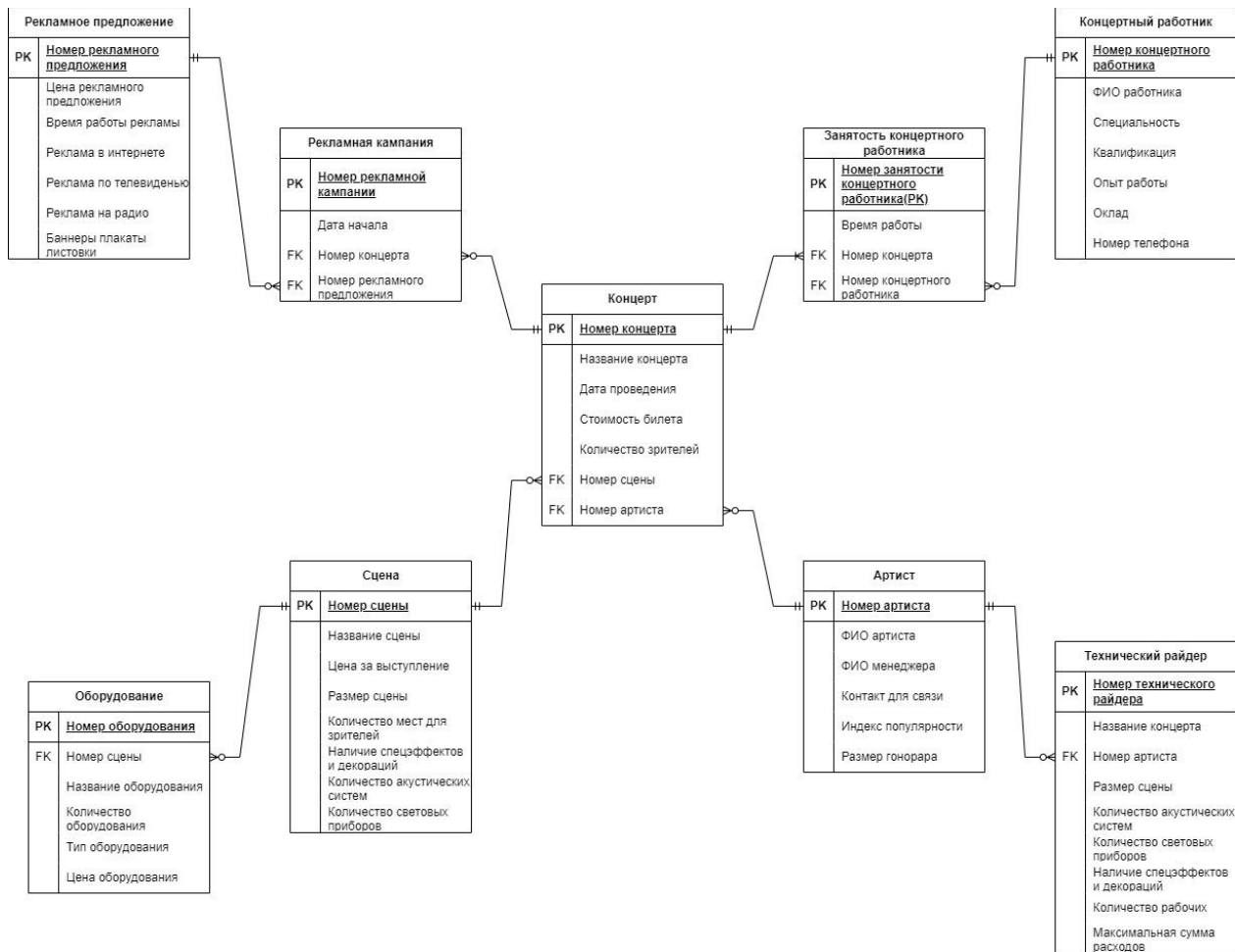


Рисунок 5 – Диаграмма сущностей

3.3 Создание базы данных

Основываясь на спроектированной диаграмме сущностей, можно начинать разработку базы данных в соответствии с определенными ранее сущностями.

База данных[33] – это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. База данных обычно управляется системой управления базами данных (СУБД). Данные вместе с СУБД, а также приложения, которые с ними связаны, называются системой баз данных, или, для краткости, просто базой данных.

Данные в наиболее распространенных типах современных баз данных обычно хранятся в виде строк и столбцов, формирующих таблицу[34]. Этими данными можно легко управлять, изменять, обновлять, контролировать и упорядочивать. В большинстве БД для запросов используется язык (SQL)[35].

Для базы данных обычно требуется комплексное программное обеспечение, которое называется системой управления базами данных (СУБД). СУБД служит интерфейсом между БД и пользователями или программами, предоставляя пользователям возможность получать и обновлять информацию, а также управлять ее упорядочением и оптимизацией. СУБД обеспечивает контроль и управление данными, позволяя выполнять различные административные операции, такие как мониторинг производительности, настройка, а также резервное копирование и восстановление.

Для создания и управление базой данных в работе использовалась реляционная СУБД SQL Server Management Studio (SSMS)[36] от Microsoft. Таблицы в базе данных разрабатывались на основании диаграммы сущностей и в соответствии с принципом нормализации базы данных, с упором на 3 нормальную форму[37].

Для создания таблиц и связей между ними использовался скрипт, часть которого на примере таблиц: концерт и сцена представлена в листинге 1.

Листинг 1 – Скрипт создания таблиц и связей между ними

```
USE master
GO
CREATE DATABASE OrganizingConcerts
ON (
    NAME = 'OrganizingConcerts',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.SQLEXPRESS\MSSQL\DATA\OrganizingConcerts.mdf',
    SIZE = 10,
    MAXSIZE = 50,
    FILEGROWTH = 1
)
LOG ON (
    NAME = 'OrganizingConcerts_log',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.SQLEXPRESS\MSSQL\DATA\OrganizingConcerts_log.ldf',
    SIZE = 10,
    MAXSIZE = 50,
    FILEGROWTH = 1
)
GO

USE OrganizingConcerts
GO

CREATE TABLE Concerts (
    ConcertId int identity (1, 1),
    ConcertTitle varchar(70) not null,
    DateOfEvent datetime not null,
    TicketPrice money not null,
    NumberOfViewers int not null,
    SceneId int not null,
    ArtistId int not null
)
GO

ALTER TABLE Concerts
ADD CONSTRAINT PK_Concert primary key(ConcertId)
GO

CREATE TABLE Scenes (
    SceneId int identity (1, 1),
    SceneName varchar(70) not null,
    PricePerPerformance money not null,
    StageArea float not null,
    NumberOfSeats int not null,
    AvailabilityOfSpecialEffectsAndDecorations varchar(30) not null,
    NumberOfSoundSystems int not null,
    NumberOfLuminaires int not null
)
GO

ALTER TABLE Scenes
ADD CONSTRAINT PK_Scene primary key(SceneId)
GO

ALTER TABLE Concerts
ADD CONSTRAINT FK_Concert_Scene foreign key
(SceneId) references Scenes(SceneId)on delete cascade
GO
```


Общий вид таблиц в получившейся базе данных показан на рисунке 6.

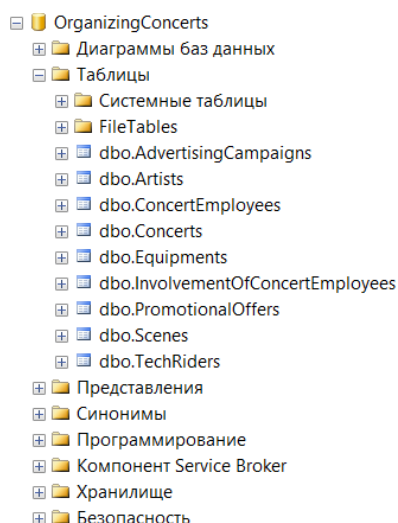


Рисунок 6 – Общий вид таблиц в базе данных

Устройство таблицы в базе данных на примере таблицы концертов представлено на рисунке 7.

	Имя столбца	Тип данных	Разрешить знач...
►	ConcertId	int	<input type="checkbox"/>
	ConcertTitle	varchar(70)	<input type="checkbox"/>
	DateOfEvent	datetime	<input type="checkbox"/>
	TicketPrice	money	<input type="checkbox"/>
	NumberOfViewers	int	<input type="checkbox"/>
	Sceneld	int	<input type="checkbox"/>
	ArtistId	int	<input type="checkbox"/>

Рисунок 7 – Устройство таблицы в базе данных

Пример записей в таблице концертов представлен на рисунке 8.

	ConcertId	ConcertTitle	DateOfEvent	TicketPrice	NumberOf...	Sceneld	ArtistId
►	4	Вечер лунн...	2023-05-29 ...	4000,0000	350	2	3
	6	Авантазия	2023-10-10 ...	3500,0000	350	2	3
	7	Эпика	2023-07-29 ...	2000,0000	150	1	4
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Рисунок 8 – Пример записей в таблице

3.4 Data Access Layer

Следующим шагом в разработке системы является создание DAL - слоя доступа к данным[38].

Слой доступа к данным (DAL) — это компонент архитектуры программного обеспечения, отвечающий за управление хранением и извлечением данных приложения. Он находится между уровнем бизнес-логики и системой хранения данных и обеспечивает уровень абстракции[39], который позволяет уровню бизнес-логики взаимодействовать с системой хранения данных, не зная о ее конкретной реализации.

DAL отвечает за выполнение таких задач, как:

- подключение к системе хранения данных и управление подключением;
- генерация и выполнение SQL-запросов или других команд доступа к данным для извлечения и хранения данных;
- отображение данных из системы хранения данных в объекты данных приложения и наоборот;
- обработка ошибок и исключений, связанных с доступом к данным;
- обеспечение поддержки транзакций и других функций доступа к данным;

DAL предназначен для многократного использования и не зависит от бизнес-логики и реализации хранилища данных. Это позволяет легко модифицировать или расширять приложение, не затрагивая базовый код доступа к данным. Для разработки используется язык C#[40]

Таким образом, уровень доступа к данным — это компонент архитектуры программного обеспечения, который отвечает за управление хранением и извлечением данных приложения. Он находится между уровнем бизнес-логики и системой хранения данных и обеспечивает уровень абстракции, который позволяет уровню бизнес-логики взаимодействовать с системой хранения данных, не зная о ее конкретной реализации.

Для использования сразу на нескольких уровнях модели были вынесены в отдельную библиотеку, которая была подключена к уровню DAL (рисунок 9).

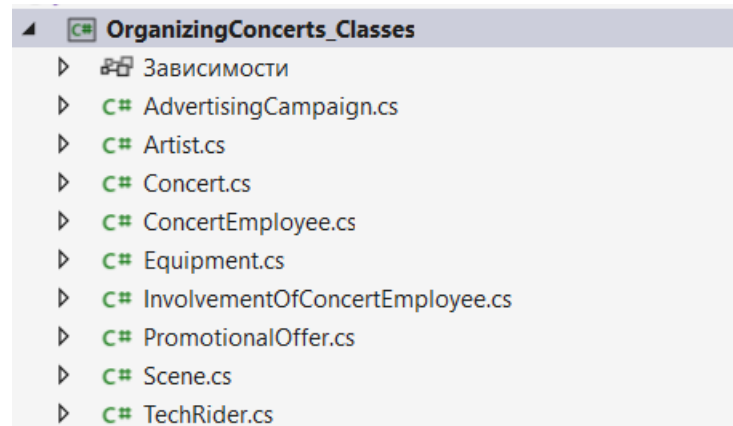


Рисунок 9 - Общий вид библиотеки моделей

Для функционирования слоя DAL в библиотеке были созданы классы, соответствующие таблицам в БД. Названия и типы данных свойств созданных классов соответствуют столбцам соответствующих таблиц. Пример класса концерт представлен в листинге 2.

Листинг 2 – Класс Concert

```
public class Concert
{
    [Key]
    public int ConcertId { get; set; }
    public int ArtistId { get; set; }
    public int SceneId { get; set; }
    public string? ConcertTitle { get; set; }
    public DateTime DateOfEvent { get; set; }
    public decimal TicketPrice { get; set; }
    public int NumberOfViewers { get; set; }
}
```

Для остальных таблиц были также созданы соответствующие классы.

DAL составлен из контроллеров, реализующих CRUD[41] операции, Application context[42] и связи с БД (Рисунок 10).

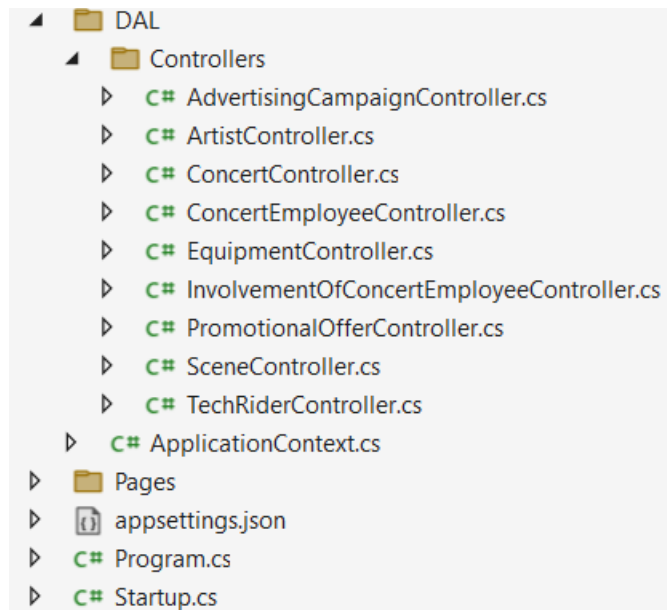


Рисунок 10 - Общий вид слоя доступа к данным

Для взаимодействия с базой данных через Entity Framework Core[43] необходим контекст данных - класс, унаследованный от класса Microsoft.EntityFrameworkCore.DbContext. Поэтому в проект был добавлен класс ApplicationContext (листинг 3).

Листинг 3 – ApplicationContext

```
public class ApplicationContext : DbContext
{
    public DbSet<TechRider> TechRiders { get; set; }
    public DbSet<Artist> Artists { get; set; }
    public DbSet<Scene> Scenes { get; set; }
    public DbSet<Equipment> Equipments { get; set; }
    public DbSet<Concert> Concerts { get; set; }
    public DbSet<PromotionalOffer> PromotionalOffers { get; set; }
    public DbSet<AdvertisingCampaign> AdvertisingCampaigns { get; set; }
    public DbSet<ConcertEmployee> ConcertEmployees { get; set; }
    public DbSet<InvolvementOfConcertEmployee> InvolvementOfConcertEmployees {
get; set; }
    public ApplicationContext(DbContextOptions<ApplicationContext> options)
        : base(options)
    {
        Database.EnsureCreated();
    }
    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
    {
        var builder = new ConfigurationBuilder();
        builder.SetBasePath(Directory.GetCurrentDirectory());
        builder.AddJsonFile("appsettings.json");
        var config = builder.Build();
        string connectionString =
config.GetConnectionString("DefaultConnection");

        optionsBuilder.UseSqlServer(connectionString);
    }
}
```

```
    }
}
```

Свойство DbSet представляет собой коллекцию объектов, которая сопоставляется с определенной таблицей в базе данных. В конструкторе класса ApplicationDbContext через параметр options будут передаваться настройки контекста данных. А в самом конструкторе с помощью вызова Database.EnsureCreated() по определению модели будет создаваться база данных (если она отсутствует). Чтобы подключаться к базе данных, необходимо задать параметры подключения. Это можно сделать в файле конфигурации[44]. Для этого изменили файл appsettings.json, добавив в него строку для подключения к базе данных (листинг 3).

Листинг 4 – Конфигурационный файл appsettings.json

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=DESKTOP-
8DDEC5H\\SQLEXPRESS;Database=OrganizingConcerts;Trusted_Connection=True;MultipleActiveResultSets=true;TrustServerCertificate=True"
  }
}
```

Контекст данных, через который идет взаимодействие с базой данных, передается в приложение в качестве сервиса через механизм внедрения зависимостей. Поэтому для работы с EntityFrameworkCore необходимо добавить класс контекста данных в коллекцию сервисов приложения. Для этого в startup.cs был добавлен следующий код (листинг 5).

Листинг 5 - Подключение класса контекста данных

```
string con = "Server=Localhost\\SQLEXPRESS;Database=OrganizingConcerts; " +
            "Trusted_Connection=True;";
services.AddDbContext<ApplicationContext>(options => options.UseSqlServer(con));
```

В слое DAL через контроллеры были реализованы CRUD операции, позволяющие создавать, редактировать, считывать и удалять записи из соответствующих таблиц в базе данных.

В качестве примера ниже показан класс, реализующий CRUD операции (листинг 6).

Листинг 6 – пример контроллера, реализующего CRUD операции

```
{
  [ApiController]
```

```

[Route("/Concert")]
public class ConcertController : ControllerBase
{
    ApplicationDbContext db;
    public ConcertController(ApplicationContext context)
    {
        db = context;
    }
    [HttpGet]
    public async Task<ActionResult<IEnumerable<Concert>>> Get(
        string? ConcertTitle, int? ArtistId, int? SceneId,int? ConcertId)
    {
        IQueryable<Concert> query = db.Concerts;
        if (!string.IsNullOrEmpty(ConcertTitle))
            query = query.Where(c => c.ConcertTitle != null &&
                c.ConcertTitle.Contains(ConcertTitle));
        if (ArtistId != null)
            query = query.Where(c => c.ArtistId == ArtistId);
        if (SceneId != null)
            query = query.Where(c => c.SceneId == SceneId);
        return await query.ToListAsync();
    }
    [HttpPost]
    public async Task<ActionResult<Concert>> Post(Concert Concert)
    {
        if (Concert == null)
        {
            return BadRequest();
        }
        db.Concerts.Add(Concert);
        await db.SaveChangesAsync();
        return Ok(Concert);
    }
    [HttpPut]
    public async Task<ActionResult<Concert>> Put(Concert Concert)
    {
        if (Concert == null)
        {
            return BadRequest();
        }
        db.Update(Concert);
        await db.SaveChangesAsync();
        return Ok(Concert);
    }
    [HttpDelete]
    [Route("delete")]
    public async Task<ActionResult<Concert>> Delete(int ConcertId)
    {
        Concert Concert = db.Concerts.FirstOrDefault(
            x => x.ConcertId == ConcertId);
        if (Concert == null)
        {
            return NotFound();
        }
        db.Concerts.Remove(Concert);
        await db.SaveChangesAsync();
        return Ok(Concert);
    }
}
}

```

3.5 Слой бизнес логики – Domain

Организация логики приложения с трехзвенной архитектурой по звеньям[45] показана на рисунке 11.



Рисунок 11 – Звенья приложения с трехзвенной архитектурой

Как правило, в приложении определяются слои пользовательского интерфейса, бизнес-логики, доступа к данным и хранения данных. В рамках такой архитектуры пользователи выполняют запросы через слой пользовательского интерфейса, который взаимодействует только со слоем бизнес-логики[46]. Слой бизнес-логики, в свою очередь, может вызывать слой доступа к данным для обработки запросов. Слой пользовательского интерфейса не должен выполнять запросы напрямую к слою доступа к данным и какими-либо другими способами напрямую взаимодействовать с функциями сохраняемости. Аналогичным образом, слой бизнес-логики должен взаимодействовать с функциями сохраняемости только через слой доступа к данным. Таким образом, для каждого слоя четко определена своя обязанность[47].

Стоит отдельно выделить термин HTTP[48], поскольку именно благодаря HTTP запросам происходит взаимодействие внутри системы.

HTTP — широко распространённый протокол передачи данных. Этот протокол предполагает использование клиент-серверной структуры передачи данных. Клиентское приложение формирует запрос и отправляет его на сервер, после чего серверное программное обеспечение обрабатывает данный запрос, формирует ответ и передаёт его обратно клиенту. После этого

клиентское приложение может продолжить отправлять другие запросы, которые будут обработаны аналогичным образом.

Задача, которая традиционно решается с помощью протокола HTTP — обмен данными между пользовательским приложением, осуществляющим доступ к веб-ресурсам (обычно это веб-браузер) и веб-сервером.

HTTP определяет множество методов запроса, которые указывают, какое желаемое действие выполнится для данного ресурса.

GET Метод

GET запрашивает представление ресурса. Запросы с использованием этого метода могут только извлекать данные.

POST Метод

POST используется для отправки сущностей к определённому ресурсу. Часто вызывает изменение состояния или какие-то побочные эффекты на сервере.

DELETE Метод

DELETE используется для удаления ресурса, идентифицированного конкретным URI (ID).

В данном слое используются такие же классы, как и в слое DAL, они были подключены через соответствующую библиотеку.

Для совершения операций с базой были реализованы классы, позволяющие удалять, создавать и считывать записи из соответствующих таблиц в базе данных.

В качестве примера показан класс, реализующий данные операции (листинг 7).

Листинг 7 – Класс ConcertController

```
{
    [ApiController]
    [Microsoft.AspNetCore.Mvc.Route("api/Domain/Concert")]
    public class ConcertController : ControllerBase
    {
        private readonly string? _dalUrl;
        public ConcertController(IConfiguration concertRepository)
```



```

{
    _dalUrl = concertRepository.GetValue<string>("DalUrl");
}

[HttpGet]
public async Task<IEnumerable<Concert>?> GetConcerts()
{
    var client = new HttpClient();
    var response = await client.GetAsync($"{_dalUrl}/Concert");
    response.EnsureSuccessStatusCode();
    var content = await response.Content.ReadAsStringAsync();

    return JsonSerializer.Deserialize<IEnumerable<Concert>>(
        content, new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        });
}

[HttpGet{Id}]
public async Task<IEnumerable<Concert>?> GetConcertsById(int ConcertId)
{
    var client = new HttpClient();
    var response = await client.GetAsync(
        $"{_dalUrl}/Concert?ConcertId={ConcertId}");
    response.EnsureSuccessStatusCode();
    var content = await response.Content.
        ReadFromJsonAsync<IEnumerable<Concert>>();
    return content;
}

[HttpGet("ConcertTitle")]
public async Task<IEnumerable<Concert>?> GetConcertByTitle(
    string? ConcertTitle)
{
    var client = new HttpClient();
    var response = await client.GetAsync(
        $"{_dalUrl}/Concert?ConcertTitle={ConcertTitle}");
    response.EnsureSuccessStatusCode();
    var content = await response.Content.
        ReadFromJsonAsync<IEnumerable<Concert>>();
    return content;
}

[HttpGet("ConcertArtist")]
public async Task<IEnumerable<Concert>?> GetConcertsByArtistId(int ArtistId)
{
    var client = new HttpClient();
    var response = await client.GetAsync(
        $"{_dalUrl}/Concert?ArtistId={ArtistId}");
    response.EnsureSuccessStatusCode();
    var content = await response.Content.
        ReadFromJsonAsync<IEnumerable<Concert>>();
    return content;
}

[HttpGet("ConcertScene")]
public async Task<IEnumerable<Concert>?> GetConcertsBySceneId(int SceneId)
{
    var client = new HttpClient();
    var response = await client.GetAsync(
        $"{_dalUrl}/Concert?SceneId={SceneId}");
    response.EnsureSuccessStatusCode();
    var content = await response.Content.
        ReadFromJsonAsync<IEnumerable<Concert>>();

```

```

        return content;
    }
    [HttpPost("Add")]
    public async Task<ActionResult<Concert>> PostProduct(Concert Concert)
    {
        var client = new HttpClient();
        JsonContent response = JsonContent.Create(Concert);
        using var result = await client.PostAsync(
            $"{_dalUrl}/Concert", response);
        var content = await result.Content.ReadFromJsonAsync<Concert>();

        if (content == null)
            return BadRequest();
        else
            return content;
    }
    [HttpDelete("Delete")]
    public async Task<bool> DeleteConcertsById(int ConcertId)
    {
        var client = new HttpClient();
        var response = await client.DeleteAsync(
            $"{_dalUrl}/Concert/delete?ConcertId={ConcertId}");
        return true;
    }
}
}

```

3.6 Пользовательский интерфейс

После разработки предыдущих двух слоёв и базы данных, можно приступить к созданию сайта[49]. Основой данного веб-сайта являются HTML[50], CSS[51], JavaScript[52].

HTML (Hypertext Markup Language) – это код, который используется для структурирования и отображения веб-страницы и её контента. HTML не является языком программирования. Это язык разметки, и используется, чтобы сообщать браузеру, как отображать веб-страницы, которые пользователь посещает. HTML состоит из ряда элементов, которые используются, чтобы вкладывать или оборачивать различные части контента, чтобы заставить контент отображаться или действовать определённым образом.

CSS или «каскадные таблицы стилей» работают с HTML для создания формата веб-страниц. Он работает поверх основы страницы, созданной с помощью HTML. Это также помогает адаптировать страницы к различным форматам, оптимизированным для настольных компьютеров или мобильных устройств. CSS — это то, что придает веб-сайтам безупречный и профессиональный вид. Это также позволяет добавлять интерактивные элементы, такие как цвет фона, заголовки, формы, графика и многое другое, что делает веб-сайты и веб-приложения гораздо более привлекательными для зрителя.

JavaScript — это интерфейсный язык программирования, который является одной из основных технологий Всемирной паутины, наряду с HTML и CSS. JavaScript используют на стороне клиента для поведения веб-страниц, часто включая сторонние библиотеки. В то время как HTML и CSS используются для управления представлением, форматированием и макетом, JavaScript используется для управления поведением различных веб-элементов.

Несмотря на то, что эти технологии легли в основу сайта, реализован он был в основном через библиотеку вышеупомянутого JavaScript – React[53].

React — это декларативная JavaScript-библиотека для создания пользовательских интерфейсов. Она позволяет собирать сложный UI из маленьких изолированных кусочков кода, называемых «компонентами».

Следующим шагом в разработке веб-интерфейса стало внедрение REST API[54] через Fetch API[55].

В React доступны различные способы использования REST API в приложениях, в том числе использование встроенного JavaScript- метода `fetch()`.

API `fetch()` является встроенным методом JavaScript, предназначенным для получения ресурсов от сервера или конечной точки API. Он определяет такие понятия, как CORS и семантика заголовка HTTP Origin, перенося их отдельные определения в другие места.

Метод `fetch()` API всегда принимает обязательный аргумент, представляющий собой путь или URL-адрес ресурса, который вы хотите получить. Он возвращает промис, который указывает на ответ от запроса, независимо от того, был ли запрос успешным или нет. При желании вы также можете передать объект параметров инициализации в качестве второго аргумента.

После получения ответа доступно несколько встроенных методов, позволяющих определить, каково содержимое тела и как его следует обрабатывать.

На этом слое используется шаблон проектирования MVC[56] («модель-представление-контроллер»). Общая схема архитектурной реализации шаблона представлена на рисунке 12.

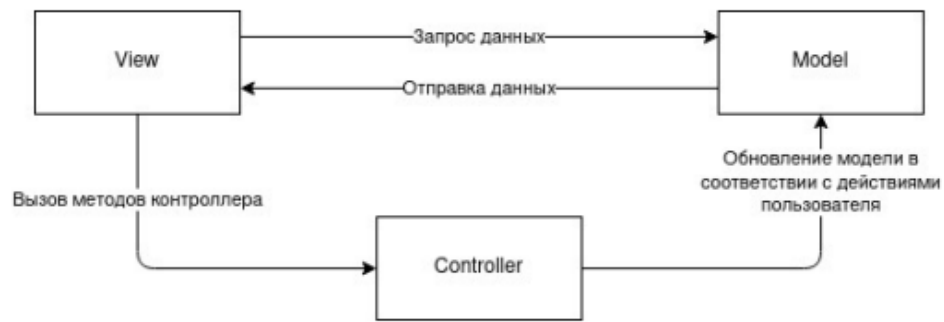


Рисунок 12 – Шаблон проектирования MVC

Шаблон предлагает разделить весь программный код пользовательского интерфейса на три составляющие: модель, представление и контроллер.

Модель содержит программные сущности, полученные из программного шлюза ИС. Представление описывает программные сущности, используемые для построения пользовательского интерфейса. Контроллер связывает между собой представление и модель, преобразуя пользовательские запросы в корректные программные сущности модели. На контроллер также возлагается ответственность за непосредственное взаимодействие с программным шлюзом приложения. Шаблон проектирования MVC позволяет отделить основную бизнес логику, представленную в модели, от логики формирования интерфейсов.

Начало любого компонента сопровождалось импортированием необходимых дополнений, как пример импорт главного компонента App.js (листинг 8).

Листинг 8 – импорт в App.js

```

import './index.css';
import './App.css'
import { Link } from "react-router-dom";
import { FaGrinStars, FaBars, FaMoneyBillWave, FaHandPeace, FaTimes,
FaFortAwesome, FaHeadset, FaVk, FaUser, FaYoutube, FaPhoneAlt, FaTelegram } from
'react-icons/fa';
  
```

Для того, чтобы сделать базовое одностраничное приложение React многостраничным потребовалось внести изменение в файл index.js (листинг 9).

Листинг 9 – изменённый index.js

```

import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
  
```

```

import App from './App';
import reportWebVitals from './reportWebVitals';
import {
  createBrowserRouter,
  RouterProvider,
} from "react-router-dom";
import AboutPage from './AboutPage';
import FormPage from './FormPage';
import PersonalPage from './PersonalPage';
import MiddlePage from './MiddlePage';
import DeletePage from './DeletePage';

const router = createBrowserRouter([
  {
    path: "/",
    element: <App/>,
  },
  {
    path: "about",
    element: <AboutPage/>,
  },
  {
    path: "for",
    element: <FormPage/>,
  },
  {
    path: "workers",
    element: <PersonalPage/>,
  },
  {
    path: "middle",
    element: <MiddlePage/>,
  },
  {
    path: "delete",
    element: <DeletePage/>,
  },
]);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<RouterProvider router={router} />
);

```

Для координирования работы веб-сайта и домена[57] использовался файл setupProxy.js (листинг 10).

Листинг 10 - файл setupProxy.js

```

const { createProxyMiddleware } = require('http-proxy-middleware');
module.exports = function(app) {
  app.use(
    '/api', createProxyMiddleware({
      target: 'http://localhost:5167',
      changeOrigin: true,
    })
  );
};

```

В качестве примера компонента, можно привести MiddlePage.js, который реализует взаимодействие с доменом посредством операции GET

и содержит кнопки перехода на формы, реализующие POST, DELETE (листинг 11).

Листинг 11 – Реализация компонента MiddlePage.js

```
import React, { Component } from 'react';
import { Link } from 'react-router-dom';
import { FaGrinStars, FaBars, FaMoneyBillWave, FaHandPeace, FaTimes,
FaFortAwesome, FaHeadset, FaVk, FaUser, FaYoutube, FaPhoneAlt, FaTelegram } from
'react-icons/fa';
import axios from "axios";
import { useEffect, useState } from "react";

function menuClick() {
  const links = document.getElementById("nav-links-middle");
  links.classList.toggle("show-links-middle");
}

function menuClick2() {
  const links = document.getElementById("nav-links-middle2");
  links.classList.toggle("show-links-middle2");
}

function menuClick3() {
  const links = document.getElementById("middle-btn-main");
  links.classList.toggle("hide-middle-btn-main");
}

function menuClick4() {
  const links = document.getElementById("middle-btn-main");
  links.classList.toggle("show-middle-btn-main");
}

function MiddlePage() {
  const [data, setData] = useState([]);
  const [name, setName] = useState('')
  const [name1, setName1] = useState('')
  const [tempValue, setTempValue] = useState('');
  handleButtonClick = () => {
    setTempValue(name);
  };
  const params = {
    ConcertTitle: name
  };
  const queryString = params
  ? Object.keys(params).map((key)
    => `${encodeURIComponent(key)}=${encodeURIComponent(params[key])}`)
    .join('&'): '';
  const baseUrl = '/api/Domain/Concert/ConcertTitle';
  const url = queryString ? `${baseUrl}?${queryString}` : baseUrl;
  useEffect(() => {
    const fetchData = async() => {
      const response = await axios.get(url)
      setData(response.data)
    }
    fetchData()
  }, [name]);
  return (
    <body>
    <section className="contact4">
    <div className="section-center clearfix">
    <article className="main-container" id='contacts'>
    /* <a><Link to='/middle' className='btn middle-btn middle-btn-main'
    id='middle-btn-main' onClick={
    menuClick3}>Вывести концерты</Link></a> */
  )
```

```

    {/ * <a><Link to='/middle' className='btn middle-btn
      middle-btn-main2'id='middle-btn-main2'
      onClick={menuClick4}>Скрыть</Link></a> */}
    <div className='get-table'>
      <h1 className='regular'>Список концертов:</h1>
      <div >
        <div className='table-head'>
          <th>ConcertId</th>
          {/ * <th>ConcertId</th> */}
          <th>ArtistId</th>
          <th>SceneId</th>
          <th>ConcertTitle</th>
          <th>DateOfEvent</th>
          <th>TicketPrice</th>
          <th>Viewers</th>
        </div>
        <tbody>
          {data.map((post) => {
            return (
              <div className='table-head2'>
                <td style={{ color: "----clr-grey-8" }}>
                  {post.concertId}</td>
                  {/ * <td>{post.concertId}</td> */}
                <td>{post.artistId}</td>
                <td>{post.sceneId}</td>
                <td>{post.concertTitle}</td>
                <td>{post.dateOfEvent}</td>
                <td>{post.ticketPrice}</td>
                <td>{post.numberOfViewers}</td>
              </div>
            );
          })}
        </tbody>
      </div>
    </div>
  </article>
  <div className='buttons'>
    <a><Link to='/middle' className='btn middle-btn first-middle' id="nav-
toggle" onClick={menuClick}>Сортировать</Link>
    <ul className="nav-links-middle" id="nav-links-middle">
      <li>
        <div className="nav-link-middle scroll-link">
          <a href="#home" className="midlli scroll-link">По концерту</a>
          <input
            type="text"
            placeholder="Ввести название концерта"
            style={{width:110,marginTop:5}}
            value={name}
            onChange={event => setName(event.target.value)}
            // className='form-control-3'
          />
        </div>
      </li>
      <li>
        <a href="#about" className="nav-link-middle
          scroll-link">По номеру</a>
      </li>
      <li>
        <a href="#services" className="nav-link-middle
          scroll-link">По артисту</a>
      </li>
      <li>
        <a href="#featured" className="nav-link-middle
          scroll-link">По сцене</a>
      </li>
    </ul>
  </div>

```



```

        <li>
          <a href="#featured" className="nav-link-middle
            scroll-link">По названию</a>
        </li>
      </ul>
    </a>

    <a><Link to='/for' className='btn middle-btn'>Добавить</Link></a>
    <a><Link to='/delete' className='btn middle-btn'>Удалить</Link></a>
  </div>
  <a><Link to="/" className='btn form-back2'>Вернуться на главную</Link></a>
</div>
</section>
<footer className="footer">
  <div className="section-center-form">
    <div className="social-icons">

      <a href="#" className="social-icon">
        <FaVk/>
      </a>

      <a href="#" className="social-icon">
        <FaYoutube/>
      </a>

      <a href="#" className="social-icon">
        <FaTelegram/>
      </a>

    </div>
    <h4 className="footer-text">
      &copy; <span id="date">2023</span>
      <span className="company">Sergbess</span>
      Все права защищены
    </h4>
  </div>
</footer>
</body>
  </>
}
export default MiddlePage;

```

В компоненте `FormPage` используется метод `fetch`, с помощью которого данные, заполненные пользователем в форму, передаются на слой `Domain`. Форма, реализующая удаление, выглядит аналогично. Данный компонент приведен в листинге 12.

Листинг 12 – Компонент `FormPage.js`

```

import React, { useState } from 'react';
import { FaVk, FaYoutube, FaTelegram } from 'react-icons/fa';

function FormPage() {
  const [formValues, setFormValues] = useState({
    ConcertTitle: '',
    TicketPrice: '',
    NumberOfViewers: '',
    ArtistId: '',
    SceneId: '',
    DateOfEvent: ''
  });
  const handleSubmit = async (event) => {

```

```

        event.preventDefault();
        await fetch('/api/Domain/Concert/AddConcert', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(formValues)
        })
        .then((response) => {
            if (response.ok) {
                alert('Ваша заявка отправлена на рассмотрение');
            } else {
                throw new Error('Ошибка HTTP: ' + response.status);
            }
        })
        .catch((error) => {
            console.log(error);
        });
    };

    const handleChange = (event) => {
        const { name, value } = event.target;
        setFormValues((prevState) => ({
            ...prevState,
            [name]: value
        }));
    };

    return (
        <body>
            <section className="contact2">
                <div className="section-center clearfix">
                    <article className="contact-forma" id="contacts">
                        <h3>Уточним данные</h3>
                        <form onSubmit={handleSubmit}>
                            <div className="form-group">
                                <input type="text" placeholder="название концерта"
                                className="form-control" value={formValues.ConcertTitle} name="ConcertTitle"
                                onChange={handleChange}></input>
                                <input type="text" placeholder="стоимость билетов"
                                className="form-control" value={formValues.TicketPrice} name="TicketPrice"
                                onChange={handleChange}></input>
                                <input type="text" placeholder="количество зрителей"
                                className="form-control" value={formValues.NumberOfViewers}
                                name="NumberOfViewers" onChange={handleChange}></input>
                                <input type="text" placeholder="номер артиста"
                                className="form-control" value={formValues.ArtistId} name="ArtistId"
                                onChange={handleChange}></input>
                                <input type="text" placeholder="номер сцены" className="form-
                                control" value={formValues.SceneId} name="SceneId"
                                onChange={handleChange}></input>
                                <p className="form-date">Дата концерта</p>
                                <input type="date" placeholder="Дата проведения"
                                className="form-control" value={formValues.DateOfEvent} name="DateOfEvent"
                                onChange={handleChange}></input>
                            </div>
                            <button type="submit" className="submit-btn
                            btn">Подтвердить</button>
                        </form>
                    </article>
                    <a href="/">
                        <button className="btn form-back">Вернуться на главную</button>
                    </a>
                </div>
            </section>
            <footer className="footer">
                <div className="section-center-form">
                    <div className="social-icons">

```

```

        <a href="#" className="social-icon">
            <FaVk />
        </a>
        <a href="#" className="social-icon">
            <FaYoutube />
        </a>
        <a href="#" className="social-icon">
            <FaTelegram />
        </a>
    </div>
    <h4 className="footer-text">
        &copy; <span id="date">2023</span>
        <span className="company">Sergbess</span>
        Все права защищены
    </h4>
</div>
</footer>
</body>
);
}
export default FormPage;

```

После выполнения всех ранее названных действий можно приступить к тестированию информационно-справочной системы[58].

4 Тестирование системы

Поскольку в проекте реализуется многоуровневое приложение, тестировать его работу стоит соответствующе – отдельно проверяя работу каждого слоя, с последующей проверкой совместной работы всех компонентов вместе. Хорошим инструментом для тестирования слоя данных и бизнес-логики является веб-Фреймворк для интерактивной документации – Swagger UI[59].

Для начала тестированию подвергнется корректность функционирования DAL на примере таблицы Concert (рисунок 13).

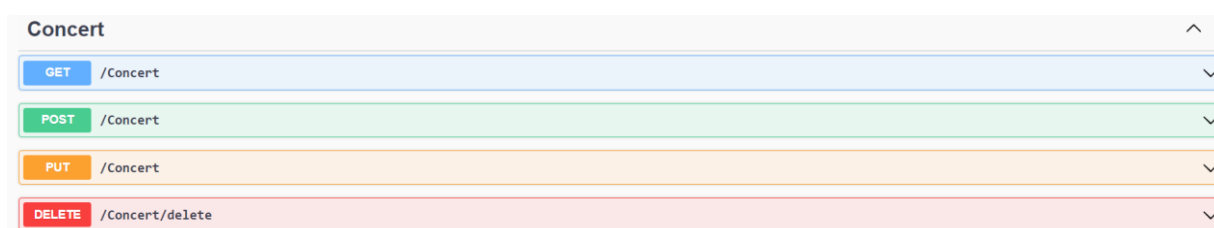


Рисунок 13 – Таблица Concert в Swagger UI

Для этого следует проверить все возможные операции, начиная с POST (рисунок 14).

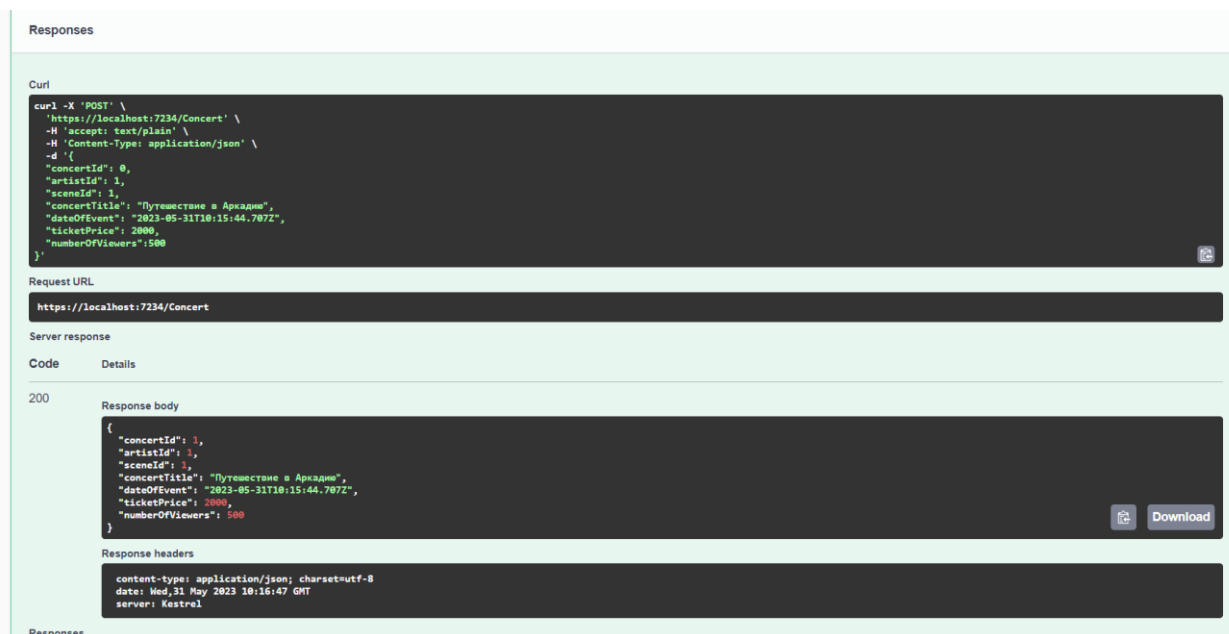


Рисунок 14 – Результат операции POST

В окне, появился результат операции POST. Это значит, что новый концерт добавлен. Далее проверяется операция GET (рисунок 15).



Рисунок 15 – Результат операции GET

Из результата операции видно, что концерт из операции POST добавлен в базу данных. Затем проверяется операция PUT (рисунок 16).



Рисунок 16 – Результат операции PUT

Для того, чтобы проверить выполнение операции PUT, снова выполняется операция GET (рисунок 17).



Рисунок 17 – Результат операции GET

Из результата операции GET видно, что операция PUT работает корректно. Стоимость билетов была изменена. Теперь с помощью операции DELETE концерт удаляется (рисунок 18).

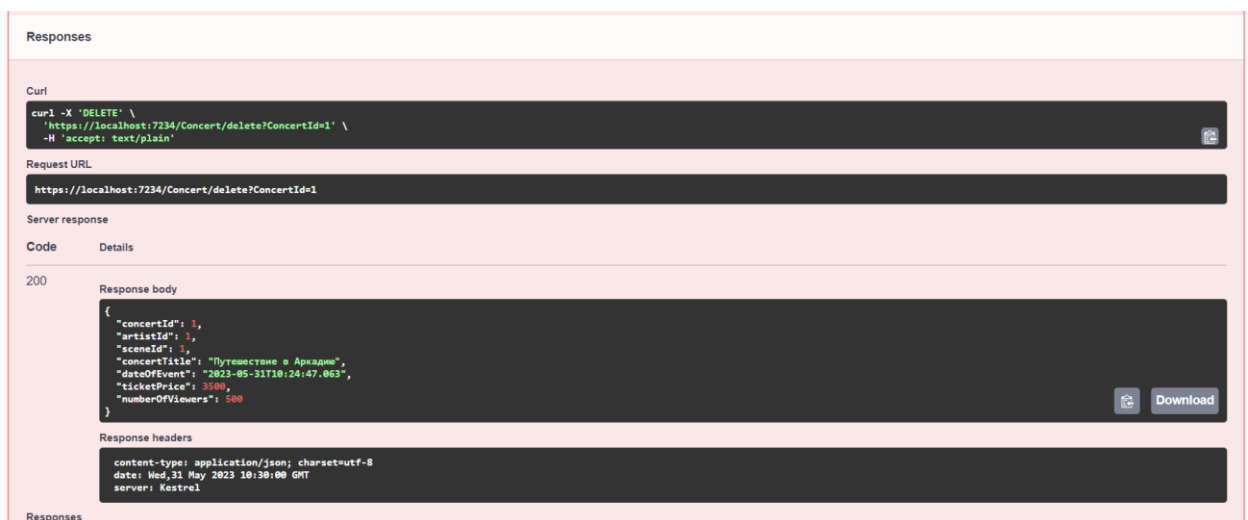


Рисунок 18 – Результат операции DELETE

Для того, чтобы проверить работоспособность операции DELETE, выполняется операция GET и проверяется, удалился ли клиент с введенным идентификатором (рисунок 19).

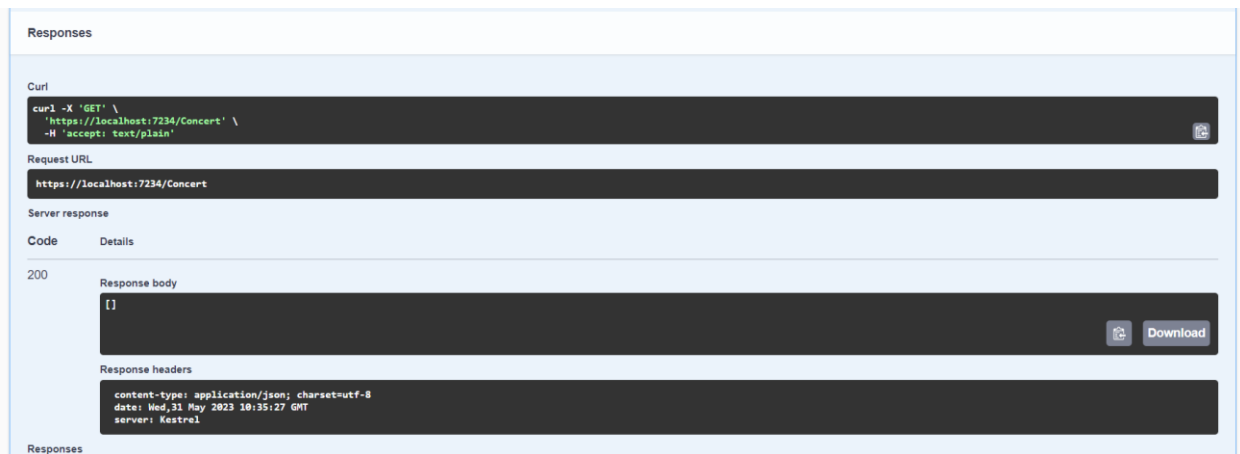


Рисунок 19 – Результат операции GET

В данном случае вернулось пустое множество, что значит, что в базе данных в таблице Concerts отсутствуют записи, что означает, что операция DELETE работает правильно.

Выполнение вышеуказанных действий позволяет сделать вывод, что слой доступа к данным работает корректно.

Проверка слоя бизнес-логики будет выполняться схожим образом, через веб-Фреймворк Swagger UI на примере таблицы Concert (рисунок 20).

Concert		^
GET	/api/Domain/Concert	▼
GET	/api/Domain/Concert/{Id}	▼
GET	/api/Domain/Concert/ConcertTitle	▼
GET	/api/Domain/Concert/ConcertArtist	▼
GET	/api/Domain/Concert/ConcertScene	▼
POST	/api/Domain/Concert/Add	▼
DELETE	/api/Domain/Concert/Delete	▼

Рисунок 20 – Таблица Concert в Swagger UI

С помощью операции POST добавляется новый концерт (рисунок 21).

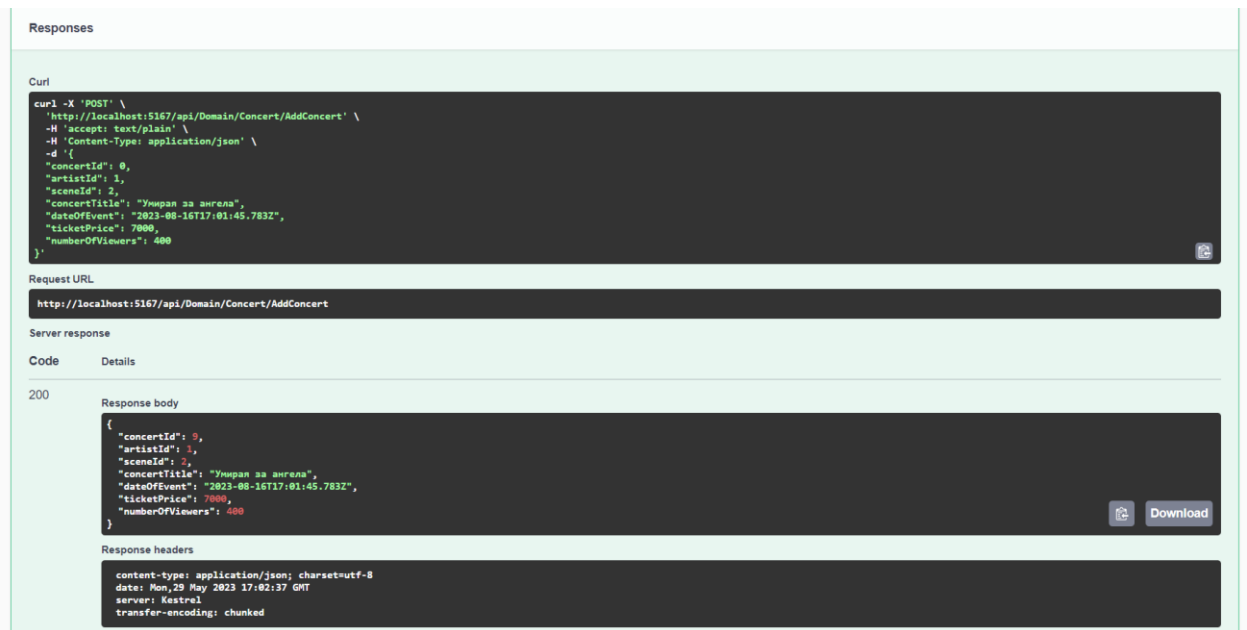


Рисунок 21 – Операция POST

Затем выполняется GET запрос по названию концерта (рисунок 22).

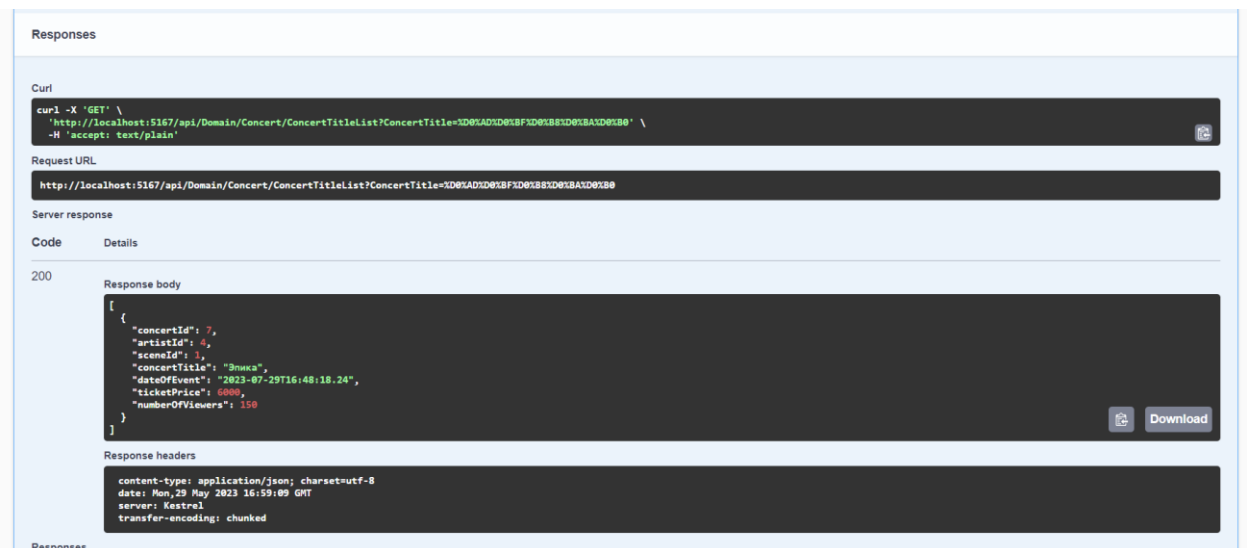


Рисунок 22 – Операция GET по названию концерта

После для удаления концерта используется операция DELETE (рисунок 23).



Рисунок 23 – Операция DELETE

Следующим этапом тестирования является просмотр веб-сайта. Стоит начать с главного раздела (рисунок 24).

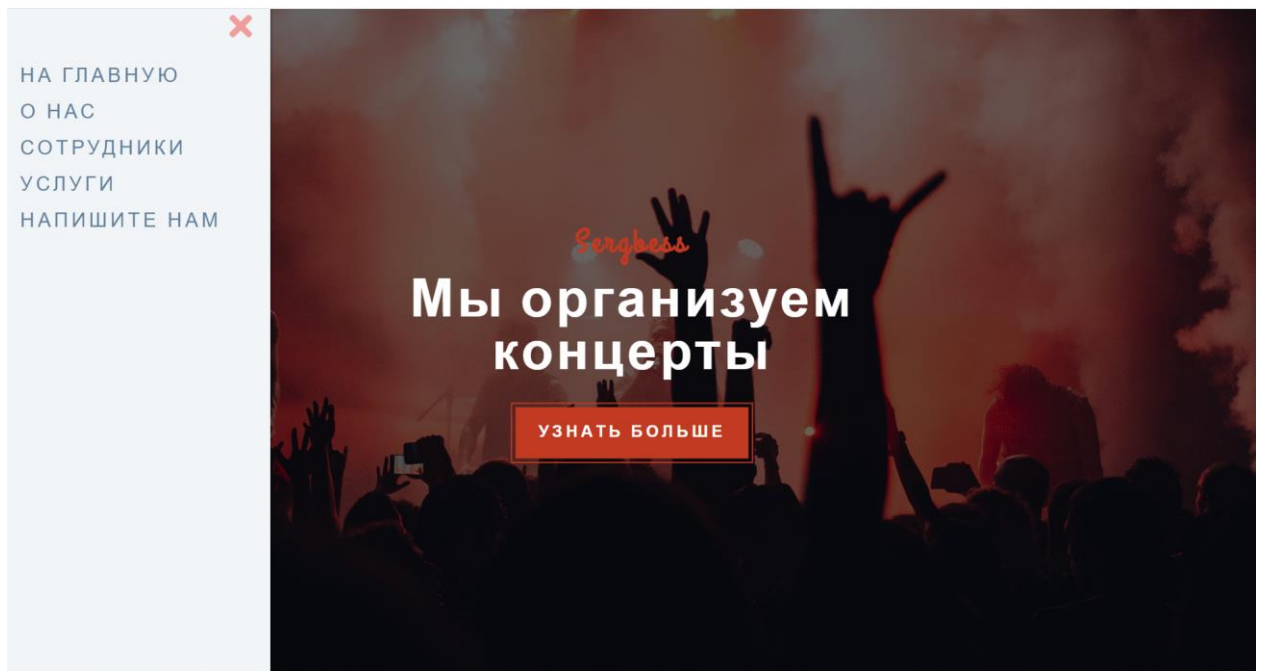


Рисунок 24 – Начало главной страницы

Общая информация о компании содержится в последующем разделе (рисунок 25).

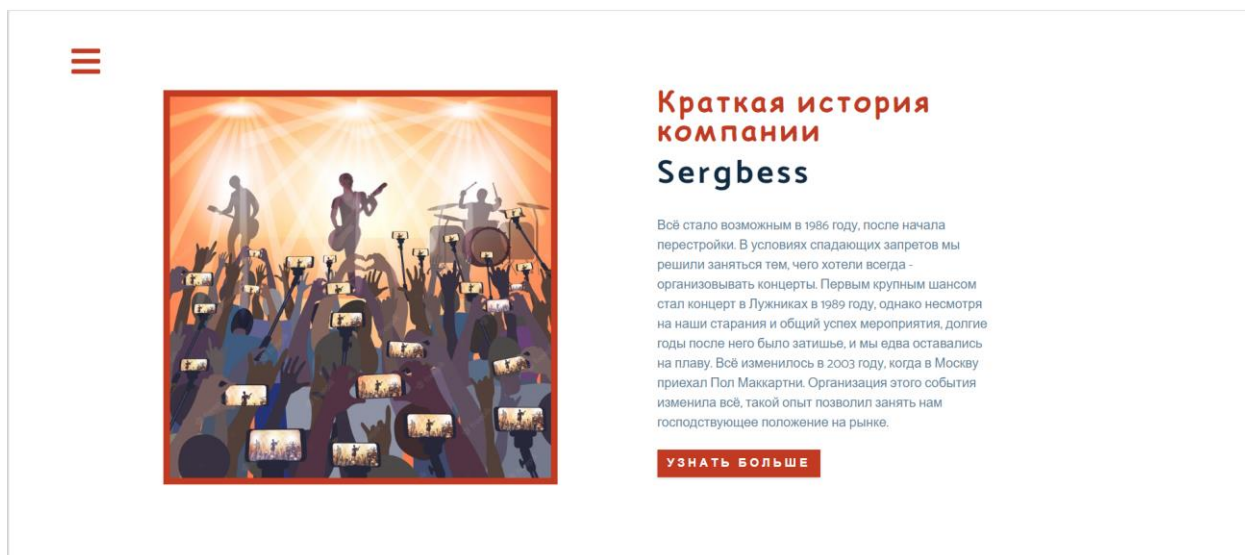


Рисунок 25 – Раздел с историей компании

Общая информация о сотрудниках содержится в последующем разделе (рисунок 26).

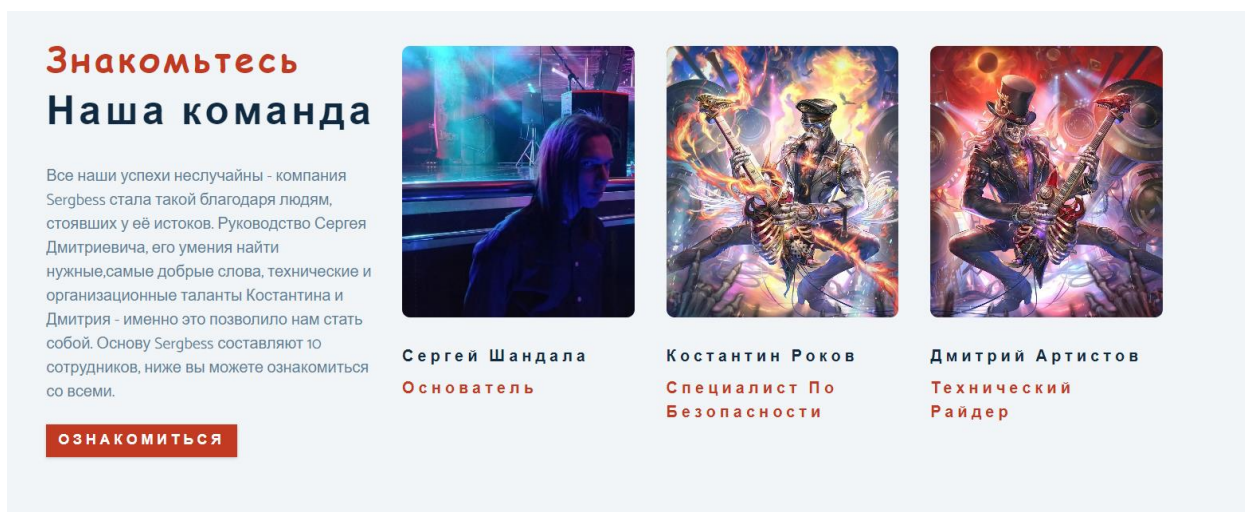



Рисунок 26 – Раздел с сотрудниками компании

Список услуг кампании (рисунок 27).




Ознакомьтесь С Нашими Услугами



Организация Концерта

Организуем ваш концерт, обеспечив полный комплект услуг, сопровождая вас на всём протяжении процесса

[Подробнее](#)



Консультация

Проинструктируем вас как собственными силами организовать концерт, если у вас есть всё необходимое

[Подробнее](#)

Рисунок 27 – Список услуг компании

Крайним на сайте является раздел обратной связи (рисунок 28).



Адрес
308 NEGRA AROYA LANE
ALBUQUERQUE, NEW MEXICO, 87104.

Почта
EMAIL@EMAIL.COM

Телефон
+ 8 800 555 35 35

Свяжитесь С Нами

[ПОДТВЕРДИТЬ](#)

© 2023 SERGBESS ВСЕ ПРАВА ЗАЩИЩЕНЫ

Рисунок 28 – Раздел обратной связи

Сайт позволяет отслеживать уже запланированные концерты (рисунок 29).

Список концертов:

ConcertId	ArtistId	ScenelId	ConcertTitle	DateOfEvent	TicketPrice	Viewers
4	3	2	Вечер лунных цветов	2023-05- 29T00:00:00	4000	350
8	4	2	Одной холодной зимней ночью	2023-05- 29T16:44:57.367	3000	150
9	1	2	Умирая за ангела	2023-08- 16T17:01:45.783	7000	400

СОРТИРОВАТЬ

ДОБАВИТЬ

УДАЛИТЬ

ВЕРНУТЬСЯ НА ГЛАВНУЮ

Рисунок 29 – Окно для отображения концертов

Имеется возможность поиска концертов по названию (рисунок 30)

Список концертов:

ConcertId	ArtistId	ScenelId	ConcertTitle	DateOfEvent	TicketPrice	Viewers
4	3	2	Вечер лунных цветов	2023-05- 29T00:00:00	4000	350

СОРТИРОВАТЬ

ДОБАВИТЬ

УДАЛИТЬ

По
названию

Вечер лунных цв

ВЕРНУТЬСЯ НА ГЛАВНУЮ

Рисунок 30 – Результат поиска

Для создания нового концерта следует заполнить форму, в которой требуется указать свои данные. После ввода данных следует нажать на кнопку «Подтвердить». После этого данные перенесутся в базу данных (рисунок 31).

Подтвердите действие на странице localhost:3000
Ваша заявка отправлена на рассмотрение

ОК

Уточним данные

ПРИЗРАЧНЫЕ ОГНИ

4500

300

3

2

ДАТА КОНЦЕРТА

06.06.2023

ПОДТВЕРДИТЬ

ВЕРНУТЬСЯ НА ГЛАВНУЮ

Рисунок 31 – Заполненная форма и уведомление о принятии заявки
Новый концерт в списке (рисунок 32).

Список концертов:

ConcertId	ArtistId	ScenelId	ConcertTitle	DateOfEvent	TicketPrice	Viewers
4	3	2	Вечер лунных цветов	2023-05- 29T00:00:00	4000	350
8	4	2	Одной холодной зимней ночью	2023-05- 29T16:44:57.367	3000	150
9	1	2	Умирая за ангела	2023-08- 16T17:01:45.783	7000	400
10	3	2	Призрачные огни	2023-06- 06T00:00:00	4500	300

СОРТИРОВАТЬ

ДОБАВИТЬ

УДАЛИТЬ

ВЕРНУТЬСЯ НА ГЛАВНУЮ

Рисунок 32 – Новый концерт в списке

Для удаления концерта следует заполнить форму, в которой требуется указать номер концерта. После ввода номера следует нажать на кнопку «Подтвердить». После этого концерт будет удален (рисунок 33).

Подтвердите действие на странице localhost:3000
Концерт успешно удален!

OK

номер концерта для
Удаления

8

ПОДТВЕРДИТЬ

ВЕРНУТЬСЯ НА ГЛАВНУЮ

Рисунок 33 – Заполненная форма и уведомление об удалении
Список без удаленного концерта (рисунок 34).

Список концертов:

ConcertId	ArtistId	SceneId	ConcertTitle	DateOfEvent	TicketPrice	Viewers
4	3	2	Вечер лунных цветов	2023-05- 29T00:00:00	4000	350
9	1	2	Умирая за ангела	2023-08- 16T17:01:45.783	7000	400
10	3	2	Призрачные огни	2023-06- 06T00:00:00	4500	300

СОРТИРОВАТЬ ДОБАВИТЬ УДАЛИТЬ

ВЕРНУТЬСЯ НА ГЛАВНУЮ

Рисунок 34 – Список без удаленного концерта

После проверки работы всех модулей можно заключить, что система работает корректно в соответствии с поставленными задачами.

Заключение

Создание Информационно-справочной системы представляет собой сложный процесс проектирования, разработки и тестирования. Целью проектирования являются обозначение контуров и деталей будущей системы и создание соответствующей документации. В процессе разработки выстраивается сам объект, устанавливается его архитектура, связи между компонентами, характерные особенности. Конечным этапом является тестирование системы, в ходе которого выявляются все возможные недостатки и ошибки, с последующим их устранением.

В данной курсовой работе был разработан проект информационной системы «Организация концертов»

В ходе курсовой работы были решены следующие поставленные задачи:

- выполнен анализ предметной области;
- смоделирована предметная область;
- разработаны все слои системы;
- протестирован функционал конечного продукта.

Список используемых источников

1. Румянцев Д. В., Франкель Н. Event-маркетинг. Все об организации и продвижении событий. – "Издательский дом" "Питер", 2017.
2. Бастрыкин К. А. ТРУДНОСТИ В ОРГАНИЗАЦИИ КОНЦЕРТОВ НАЧИНАЮЩИХ АРТИСТОВ В РОССИИ //Электронная наука. – 2020. – Т. 1. – №. 1. – С. 2-7.
3. Кобелев Е. В., Кульназарова А. В. Специфика организации концертов в период ограничений, связанных с пандемией COVID-19 //ВЕСТНИК. – 2020. – С. 52.
4. Андрущенко Е. Ю. Менеджмент в сфере академической музыкальной культуры и современные event-технологии //Ростовская государственная консерватория им. С.В. Рахманинова. – 2013– С. 80.
5. Черняк Е. Ф., Черняк М. В., Белов В. Ф. Технологические аспекты организации онлайн-мероприятий в учреждениях культуры //Вестник Кемеровского государственного университета культуры и искусств. – 2021. – №. 54. – С. 231-236.
6. Касаткина С. А. Менеджмент в сфере культуры как вид управленческой деятельности //Вестник Московского государственного университета культуры и искусств. – 2011. – №. 6. – С. 199-204.
7. Буторина Н. И., Шекунова М. В. ХАРАКТЕРИСТИКА ОРГАНИЗАЦИИ КОНЦЕРТНЫХ МЕРОПРИЯТИЙ //Акмеология профессионального образования. – 2020. – С. 279-282.
8. Коротченков Д. А. Организация административно-правовой охраны общественного порядка и обеспечения общественной безопасности при проведении массовых мероприятий //Хабаровск: Изд-во Тихоокеанского гос. ун-та. – 2006. – Т. 7. – №. 3.
9. Ермаков С. Г., Макаренко Ю. А., Соколов Н. Е. Event-менеджмент: обзор и систематизация подходов к организации мероприятий //Управленческое консультирование. – 2017. – №. 9 (105). – С. 140-148.

10. Старостина С. А. РОЛЬ ПУБЛИЧНОСТИ В ПРОФЕССИОНАЛЬНОЙ СФЕРЕ ЭСТРАДНЫХ АРТИСТОВ Г. ЯКУТСК //Концепции фундаментальных и прикладных научных исследований. – 2017. – С. 178.
11. Bittner K., Spence I. Use case modeling. – Addison-Wesley Professional, 2003.
12. Chinosi M., Trombetta A. BPMN: An introduction to the standard //Computer Standards & Interfaces. – 2012. – Т. 34. – №. 1. – С. 124-134.
13. Waddell R. D., Barnet R., Berry J. This business of concert promotion and touring: a practical guide to creating, selling, organizing, and staging concerts. – Billboard Books, 2010.
14. Синкина Я. М. ПРОЕКТИРОВАНИЕ ДЕЯТЕЛЬНОСТИ КОНЦЕРТНОГО АГЕНТСТВА: ПРОБЛЕМЫ И СПЕЦИФИКА //Диалоги о культуре и искусстве. – 2014. – С. 125-129.
15. Бердников С. Ю., Цветкова Г. С. АКТУАЛЬНОСТЬ МАРКЕТИНГ-МЕНЕДЖМЕНТА ДЛЯ КОНЦЕРТНОЙ ДЕЯТЕЛЬНОСТИ //Редакционная коллегия. – 2016. – С. 76.
16. Kang S., Kang S., Hur S. A design of the conceptual architecture for a multitenant saas application platform //2011 First ACIS/JNU International Conference on Computers, Networks, Systems and Industrial Engineering. – IEEE, 2011. – С. 462-467.
17. Опацкий В. А., Краснопахтова Л. И. Преимущества использования мобильных приложений для организации массовых мероприятий //Аллея науки. – 2017. – Т. 3. – №. 10. – С. 742-745.
18. Orozco D. Strategically astute contracting: The Ticketmaster case study //Business Horizons. – 2021. – Т. 64. – №. 2. – С. 171-180.
19. Yılmaz Ö., Easley R. F., Ferguson M. E. The future of sports ticketing: Technologies, data, and new strategies //Journal of Revenue and Pricing Management. – 2023. – С. 1-12.

20. Fahl S. et al. Rethinking SSL development in an appified world //Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. – 2013. – C. 49-60.
21. Gu X. et al. Deep API learning //Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering. – 2016. – C. 631-642.
22. Kamins M. et al. Winning Strategies for Online Purchases (eBay, Priceline and StubHub) //World Scientific Book Chapters. – 2018. – C. 209-224.
23. Reges S., Stepp M. Building Java Programs. – Pearson, 2014.
24. Greenwald R., Stackowiak R., Stern J. Oracle essentials: Oracle database 12c. – " O'Reilly Media, Inc.", 2013.
25. Meng G., Su-ping H., Hong-ji O. Java Web application based on Spring MVC framework //Computer and Modernization. – 2018. – №. 08. – C. 97.
26. Newton D. Apache Struts 2 Web Application Development. – Packt Publishing Ltd, 2009.
27. Coleman L. J. et al. Marketing the Performing Arts: Efficacy of Web 2.0 Social Networks //Proceedings of the Northeast Business & Economics Association. – 2018.
28. Mane M. R. et al. Electronic shopping using barcode scanner //Int. Res. J. Eng. Technol. – 2016. – T. 3. – №. 4. – C. 1-5.
29. Hartono S. et al. Smart hybrid learning framework based on three-layer architecture to bolster up education 4.0 //2018 international conference on ict for smart society (iciss). – IEEE, 2018. – C. 1-5.
30. Petre M. UML in practice //2013 35th international conference on software engineering (icse). – IEEE, 2013. – C. 722-731.
31. Li Q., Chen Y. L. Entity-relationship diagram //Modeling and analysis of enterprise and information systems. – Springer, Berlin, Heidelberg, 2009. – C. 125-139.
32. Bagui S., Earp R. Database design using entity-relationship diagrams. – Crc Press, 2011.

33. Gaulton A. et al. The ChEMBL database in 2017 //Nucleic acids research. – 2017. – T. 45. – №. D1. – C. D945-D954.
34. Kontinen J., Link S., Väänänen J. Independence in database relations //Logic, Language, Information, and Computation: 20th International Workshop, WoLLIC 2013, Darmstadt, Germany, August 20-23, 2013. Proceedings 20. – Springer Berlin Heidelberg, 2013. – C. 179-193.
35. Khan W. et al. SQL and NoSQL Database Software Architecture Performance Analysis and Assessments—A Systematic Literature Review //Big Data and Cognitive Computing. – 2023. – T. 7. – №. 2. – C. 97.
36. Mistry R., Misner S. Introducing Microsoft SQL Server 2014. – Microsoft Press, 2014.
37. Date C. J. Database design and relational theory: normal forms and all that jazz. – Apress, 2019.
38. Choegyen T. et al. Data access rules in a database layer : пат. 10380334 CIIIA. – 2019.
39. Pulier E., Martinez F., Hill D. C. System and method for a cloud computing abstraction layer : пат. 8931038 CIIIA. – 2015.
40. Albahari J. C# 10 in a Nutshell. – " O'Reilly Media, Inc.", 2022.
41. Gómez O. S., Rosero R. H., Cortés-Verdín K. CRUDyLeaf: a DSL for generating spring boot REST APIs from entity CRUD operations //Cybernetics and Information Technologies. – 2020. – T. 20. – №. 3. – C. 3-14.
42. Addepalli S. K. et al. Application context transfer for distributed computing resources : пат. 9507630 CIIIA. – 2016.
43. Smith J. Entity Framework core in action. – Simon and Schuster, 2021.
44. Nos K. Configuration of life cycle management for configuration files for an application : пат. 9075633 CIIIA. – 2015.
45. Litvinov A. A. ON BUSINESS LOGIC LAYER DESIGN AND ARCHITECTURE //System technologies. – 2020. – T. 1. – №. 126. – C. 86-95.

46. Khalil M. E., Ghani K., Khalil W. Onion architecture: a new approach for xaas (every-thing-as-a service) based virtual collaborations //2016 13th Learning and Technology Conference (L&T). – IEEE, 2016. – C. 1-7.
47. Boukhary S., Colmenares E. A clean approach to flutter development through the flutter clean architecture package //2019 International Conference on Computational Science and Computational Intelligence (CSCI). – IEEE, 2019. – C. 1115-1120.
48. Jestratjew A., Kwiecien A. Performance of HTTP protocol in networked control systems //IEEE Transactions on Industrial Informatics. – 2012. – T. 9. – №. 1. – C. 271-276.
49. Al-Hawari F. et al. The GJU website development process and best practices //Journal of Cases on Information Technology (JCIT). – 2021. – T. 23. – №. 1. – C. 21-48.
50. Ganapathy A. et al. HTML Content and Cascading Tree Sheets: Overview of Improving Web Content Visualization //Turkish Online Journal of Qualitative Inquiry. – 2021. – T. 12. – №. 3. – C. 2428-2438.
51. Duckett J. Web design with HTML, CSS, JavaScript and jQuery set. – IN : Wiley, 2014. – T. 1.
52. Brown E. Web development with node and express: leveraging the JavaScript stack. – O'Reilly Media, 2019.
53. Saks E. JavaScript Frameworks: Angular vs React vs Vue. – 2019.
54. Sohan S. M. et al. A study of the effectiveness of usage examples in REST API documentation //2017 IEEE symposium on visual languages and human-centric computing (VL/HCC). – IEEE, 2017. – C. 53-61.
55. Gambhir A., Raj G. Analysis of cache in service worker and performance scoring of progressive web application //2018 International Conference on Advances in Computing and Communication Engineering (ICACCE). – IEEE, 2018. – C. 294-299.
56. Freeman A., Freeman A. Pro asp. net mvc 5 platform. – Apress, 2014. – C. 3-8.

57. Zhang Y. et al. Network traffic identification of several open source secure proxy protocols //International Journal of Network Management. – 2021. – T. 31. – №. 2. – C. e2090.

58. Rainer R. K., Prince B. Introduction to information systems: Supporting and transforming business. – John Wiley & Sons, 2022.

59. Haupt F. et al. A framework for the structural analysis of REST APIs //2017 IEEE International Conference on Software Architecture (ICSA). – IEEE, 2017. – C. 55-58.