МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ

КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

ордена Трудового Красного Знамени федеральное государственное

бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра «Информатика»

Курсовая работа

**«Разработка базы данных и её подключение к веб-сайту конструкторского бюро KURS»**

по дисциплине

**Базы данных**

|  |  |
| --- | --- |
|  | Выполнил: студент гр. БЭИ2202  Кулешов А.С.  Проверил: ст. преп. Юсков И. О., к.т.н., доцент кафедры «Информатика» Шакин В. Н. |

Москва, 2023 г.

Содержание

[Введение 3](#_Toc152532935)

[Техническое задание 5](#_Toc152532936)

[1 Теоретическая часть 7](#_Toc152532937)

[1.1 Выбор базы данных 7](#_Toc152532938)

[2 Реализация базы данных 11](#_Toc152532939)

[3 Взаимодействие базы данных с сайтом 14](#_Toc152532940)

[3.1 Бронирования 14](#_Toc152532941)

[3.2 Пользователи. Аутентификация и авторизация. Профили пользователей 16](#_Toc152532942)

[3.3 Новости 18](#_Toc152532943)

[Заключение 24](#_Toc152532944)

[Список использованных источников 25](#_Toc152532945)

Введение

Современные технологии проникают во все сферы нашей жизни, в том числе и в индустрию архитектуры. Поставки материалов, процесс создания дизайна, покупка, продажа, обратная связь – всё это требует информацию, коммуникацию между участниками архитектурного процесса. Всю информацию хранить в памяти людей невозможно, а переносить на бумагу крайне трудоёмко и непродуктивно. Нужда хранить данные в цифровом формате **актуальна как никогда**. Для этого и будет использована база данных.

Во-вторых, **неоспоримую актуальность** **проблемы** демонстрирует внедрение механизма регистрации пользователя, проверки состояния готовности архитектурного проекта. Благодаря базе данных владельцы бюро смогут эффективно управлять проектами, оптимизировать распределение времени и ресурсов, уменьшить траты и время клиентов.

В-третьих, это **подтверждает актуальность проблемы** в обеспечении безопасности данных. База данных позволяет внедрить меры по защите личной информации клиентов, что становится все более важным в контексте растущей цифровизации и повышенного внимания к вопросам конфиденциальности.

В данной курсовой работе исследуется процесс разработки веб-приложения для архитектурного бюро, который позволит оптимизировать ряд процессов, выполняемых бюро. Проект включает в себя внедрение базы данных, что позволит собирать и хранить информацию о клиентах, их заказах, а также информацию о работниках бюро.

В работе будет рассмотрена архитектура приложения, разработка функциональности для регистрации и аутентификации пользователей, система просмотра состояние готовности проекта клиента, управление штатом сотрудников. Внедрение базы данных позволит эффективно управлять этой информацией, обеспечивая удобное и безопасное взаимодействие клиентов с архитектурным бюро. Вместе с тем будут рассмотрены аспекты безопасности данных и конфиденциальности пользователей, а также возможности для будущего расширения функционала приложения.

Подводя итоги, **цель этой работы** — упростить процесс коммуникации между клиентами и владельцами бюро. Владельцам бюро также смогут более эффективно управлять своим бизнесом.

**Объект исследования** — процесс создания информационной системы с использованием базы данных для упрощения работы с клиентами.

**Предмет исследования** — процесс разработки архитектуры базы данных, подходящих и производительных запросов, а также процесс усиления безопасности в базе данных.

Техническое задание

Целью проекта является расширение функциональности веб-сайта архитектурного бюро «KURS» с помощью добавления базы данных для хранения информации о сотрудниках, проектах, клиентах и их ассоциированных данных.

В настоящий момент веб-сайт «KURS» является статическим и предоставляет посетителям постоянную информацию о бюро. Проект заключается в добавлении базы данных для хранения следующей информации:

1. информация о сотрудниках,
2. информация о пользователях и их профилях,
3. информация о проектах.

**Функциональные требования**

Создание базы данных и реализация следующего функционала:

1. добавление информация о проектах;
2. Аутентификация и авторизация клиентов, а также редактирование его профиля;
3. добавление, редактирование и просмотр проекта, ассоциированного с клиентом.

**Требования к пользовательскому интерфейсу**

Интерфейс должен быть интуитивно понятным и удобным для администраторов сайта и пользователей, позволяя им управлять данными.

**Язык программирования**

Проект может быть реализован с использованием любого языка программирования.

**Система управления базами данных**

По ситуации и в зависимости от требований проекта, оставляется право выбора СУБД. Разработчик должен обосновать свой выбор СУБД, учитывая специфику проекта и его технические характеристики.

**Защита данных**

Обеспечить безопасность данных, включая защиту от SQL-инъекций и других атак.

# 1 Теоретическая часть

Для усовершенствования проекта первым делом нужно определить желаемые объекты для хранения и какое с ними должно производиться взаимодействие. В контексте котокафе такими объектами станут: бронирования, новости, пользователи и их профили. Бронирования будут представлять собой не только актуальную информацию о наличии свободных мест, но и ключевой механизм для гарантирования комфортного пребывания посетителей. Новости, в свою очередь, обеспечат посетителей актуальной информацией о происходящих событиях и предложениях в котокафе. Пользователи и профили представляют собой основу для персонализированного взаимодействия с системой. Исходя из этого, можно предположить, каким функционалом должна обладать итоговая система:

* Бронирования:
  1. Бронирование на конкретные дату и время;
  2. Обеспечение удобного оповещения о доступности мест в котокафе на выбранные дату и время.
* Новости:
  1. Отображение и хранение новостей с помощью Markdown;
  2. Возможность администраторам создавать и редактировать новости.
* Пользователи и их профили:
  1. Просмотр своего и чужого профиля;
  2. Обеспечение возможности редактирования информации о пользователе.

База данных должна будет хранить в себе эти сущности: бронирование, новость, пользователь и профиль пользователя.

## 1.1 Выбор базы данных

Для определения подходящей СУБД для проекта был проведён сравнительный анализ PostgreSQL, MySQL, Oracle, Microsoft Access и MongoDB:

* PostgreSQL:
  1. Открытый исходный код: PostgreSQL является открытой и свободной СУБД, что обеспечивает бесплатный доступ к исходному коду и высокую гибкость в изменении системы под конкретные нужды.
  2. Мощность и расширяемость: PostgreSQL поддерживает сложные SQL-запросы, транзакции, хранимые процедуры и имеет широкий набор встроенных типов данных. Он также позволяет создавать пользовательские функции и расширения.
  3. Надежность и целостность данных: PostgreSQL обеспечивает высокую степень целостности данных и надежности благодаря транзакционной модели и механизмам восстановления после сбоев.
  4. Поддержка JSON и геоданных: Встроенная поддержка JSON-типа данных и геоданных делает PostgreSQL подходящим для разнообразных приложений, включая веб-разработку и анализ больших данных.
* MySQL:
  1. Быстрота и производительность: MySQL известен своей высокой производительностью, особенно в сценариях с частым чтением данных.
  2. Простота использования: MySQL часто выбирают за его простоту использования и относительно низкую стоимость внедрения.
  3. Широкая поддержка: имеется обширное сообщество пользователей MySQL и большое количество сторонних инструментов и библиотек.
  4. Ограниченные возможности для сложных запросов: в некоторых случаях MySQL может оказаться менее мощным для сложных запросов по сравнению с PostgreSQL.
* Oracle:
  1. Масштабируемость и производительность: Oracle часто используется в крупных предприятиях благодаря своей высокой производительности и возможностям масштабирования.
  2. Расширенные функциональные возможности: Oracle предоставляет множество расширенных функций, таких как управление табличными пространствами, аналитические функции и многие другие.
  3. Высокие затраты: Oracle часто сопряжен с высокими затратами на лицензии и обслуживание, что может быть фактором, ограничивающим для малых и средних предприятий.
* Microsoft Access:

1. Легкость использования: Microsoft Access предоставляет простой способ создания баз данных, особенно для небольших проектов.
2. Ограниченные возможности: Access может быть ограничен в производительности и масштабируемости, что делает его менее подходящим для крупных и сложных приложений.
3. Основан на файле: Access использует файловую структуру, что может вызвать проблемы с одновременным доступом нескольких пользователей и безопасностью данных.

* MongoDB:

1. Документно-ориентированная модель: MongoDB хранит данные в формате BSON (бинарный JSON), что удобно для хранения сложных документов и вложенных структур данных.
2. Гибкость схемы: MongoDB предоставляет гибкость в изменении схемы данных, что полезно для проектов с быстро меняющимися требованиями.
3. Масштабируемость: MongoDB хорошо масштабируется горизонтально, что позволяет обрабатывать большие объемы данных и трафика.

Так как разрабатываемая система не требует больших мощностей, не имеет крайне сложной структуры, его поддержка не должна иметь больших затрат, но его реализация должна на основе уже построенных частей позволять расширять систему и реализовывать новые функции, то в качестве СУБД была выбрана PostgreSQL, благодаря отсутствию стоимости обслуживания, высококлассному уровню безопасности, а также огромному количеству расширений и возможностей, что позволяет выполнить любую предназначенную для СУБД задачу без особых затрат.

# 2 Реализация базы данных

Для реализации базы данных было принято решение создать 4 таблицы: «users», «user\_profiles», «news», «bookings».

Запрос для создания таблицы «users»:

create table users  
(  
 id bigint generated always as identity  
 primary key,  
 login varchar(30) not null,  
 password\_hash varchar(72) not null,

role varchar(100) not null default ‘user’  
);  
  
create index users\_login\_idx  
 on users (login);

Так как аутентификация проводится по логину, то был создан индекс на поле «login». «password\_hash» имеет длину 72, потому что максимальная длина хэша BCrypt, выбранного для хэширования паролей равна 72. Роль пользователя отражает его права, всего их две – user – для обычного пользователя, и admin – для администратора.

Запрос для создания таблицы «news»:

create table news  
(  
 id bigint generated always as identity  
 primary key,  
 title varchar(500) not null,  
 content bytea not null,  
 date time with time zone not null,  
 author\_id bigint not null  
 references users  
 on update cascade on delete restrict  
);

Поле идентификатора автора ссылается на таблицу с пользователями, а поле с содержимым — бинарное хранилище Markdown.

Запрос для создания таблицы «bookings»:

create table bookings  
(  
 id bigint generated always as identity  
 primary key,  
 name varchar(100) not null,  
 phone\_number varchar(30) not null,  
 date timestamp with time zone not null,  
 is\_call\_undesirable boolean default false not null,  
 user\_id bigint  
 references users  
 on update restrict on delete restrict  
);

Запрос для создания таблицы «user\_profiles»:

create table user\_profiles  
(  
 user\_id bigint not null  
 primary key  
 references users  
 on update restrict on delete restrict,  
 bio varchar(1024)  
);

«user\_profiles» представляет дополнительную информацию о пользователях.

В итоге получается структура, изображённая на ER-диаграмме, приведённой на рисунке 1.

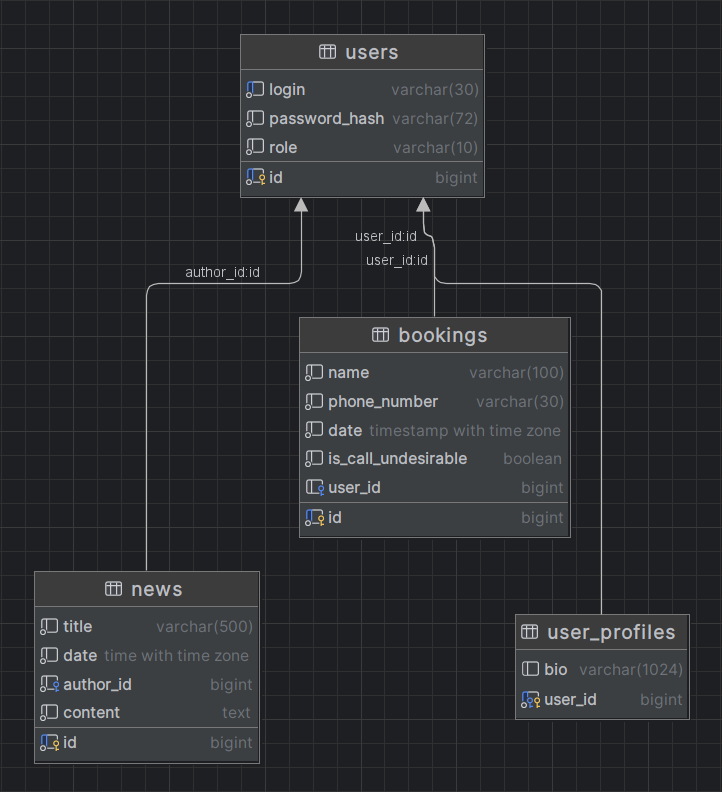


Рисунок 1 — ER-диаграмма полученной базы данных

Запросы будут описаны в главе 3, так как будет использоваться O/RM-система – Entity Framework Core.

# 3 Взаимодействие базы данных с сайтом

## 3.1 Бронирования

Для бронирования была создана отдельная форма, показанная на рисунке 2.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 2 — Форма бронирования

Эта форма при отправке отправляет введённые данные на проверку на сервер, и если проверка пройдена, то создаётся новое бронирование, а также высвечивается благодарность за бронирование. Также если пользователь аутентифицирован, то его идентификатор тоже сохраняется для упрощения идентификации клиента.

Код формы:

***document***.getElementById("bookForm")!.addEventListener("submit", async event => {  
 event.preventDefault();  
  
 const form = event.target as HTMLFormElement;  
  
 const response = await fetch(form.action,  
 {  
 method: "post",  
 body: new FormData(form),  
 credentials: 'include'  
 });  
  
 if (response.status === 201)  
 ***window***.location.href = "/visit/booking-thanks";  
 else if (response.status === 404) {  
 const errorElement = ***document***.getElementById("bookFormError") as HTMLElement;  
 errorElement.classList.add("shown");  
 errorElement.innerText = "На данное время все столики уже забронированы. Пожалуйста, выберите другое время.";  
 }  
 else {  
 const errorElement = ***document***.getElementById("bookFormError") as HTMLElement;  
 errorElement.classList.add("shown");  
 errorElement.innerText = "Что-то не так. Пожалуйста, проверьте правильность введённых данных.";  
 }  
});

Код сервера:

[HttpPost]  
[ValidateAntiForgeryToken]  
public async Task<IActionResult> Post([FromForm] BookingFormModel bookingFormModel)  
{  
 ValidationResult validationResult = await validator.ValidateAsync(bookingFormModel);  
  
 if (!validationResult.IsValid)  
 return UnprocessableEntity(validationResult.Errors);  
  
 Booking booking = (Booking)bookingFormModel;  
  
 string? currentUserIdString = HttpContext.User?.FindFirst(ClaimTypes.NameIdentifier)?.Value;  
  
 if (currentUserIdString is not null)  
 booking.UserId = Int64.Parse(currentUserIdString);  
  
 if (dbContext.Bookings.Count(x => x.Date == booking.Date) >= 7)  
 return NotFound(new { message = "No available for booking tables found." });  
  
 dbContext.Bookings.Add(booking);  
  
 try  
 {  
 await dbContext.SaveChangesAsync();  
 }  
 catch  
 {  
 return BadRequest();  
 }  
  
 return Created();  
}

## 3.2 Пользователи. Аутентификация и авторизация. Профили пользователей

Аутентификация происходит с помощью формы аутентификации, показанной на рисунке 3.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 3 — Форма аутентификации

Эта форма отправляет запрос на сервер с введёнными данными, сравнивает их с сохранёнными в базе данных, и возвращает результат:

public async Task<IActionResult> OnPost(string username, string password, bool remember = false, [FromQuery] Uri? returnUrl = null)  
{  
 User? user = await uniContext.Users.FirstOrDefaultAsync(x => x.Login == username);  
  
 if (user is null)  
 {  
 WrongCredentials = true;  
 return Page();  
 }  
  
 if (!BCrypt.Net.BCrypt.EnhancedVerify(password, user.PasswordHash))  
 {  
 WrongCredentials = true;  
 return Page();  
 }  
  
 List<Claim> claims =  
 [  
 new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),  
 new Claim(ClaimTypes.Name, user.Login),  
 new Claim(ClaimTypes.Role, user.Role)  
 ];  
  
 ClaimsIdentity identity = new(claims, CookieAuthenticationDefaults.AuthenticationScheme);  
 ClaimsPrincipal claimsPrincipal = new(identity);  
  
 await HttpContext.SignInAsync(claimsPrincipal, new AuthenticationProperties  
 {  
 IsPersistent = remember  
 });  
  
 if (returnUrl?.IsAbsoluteUri == false)  
 return Redirect(returnUrl.ToString());  
  
 return RedirectToPage("/Index");  
}

У каждого пользователя есть свой профиль. Его можно посмотреть, а также редактировать. Просмотр профиля доступен для любого пользователя, но кнопка «Выйти» появляется только тогда, когда пользователь аутентифицирован, а кнопка «Редактировать» появляется только тогда, когда пользователь авторизован редактировать профиль. Вид профиля приведён на рисунке 4.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 4 — Профиль пользователя

Код сервера на редактирование профиля:

public async Task<IActionResult> OnPost(string? bio)  
{  
 long userId = Int64.Parse(User.FindFirst(ClaimTypes.NameIdentifier)!.Value);  
 UserProfile? userProfile = await uniContext.UserProfiles  
 .FirstOrDefaultAsync(x => x.UserId == userId);  
  
 if (userProfile == null)  
 return RedirectToPage("/Account/Profile");  
  
 userProfile.Bio = bio?.Trim()[..Math.Min(bio.Length, 1024)];  
  
 await uniContext.SaveChangesAsync();  
  
 return RedirectToPage("/Account/Profile");  
}

## 3.3 Новости

Для реализации новостей требовалось добавить их редактор и отображатель. Для этих целей была выбрана библиотека tui.editor. В нужные моменты сервер либо возвращал данные из базы данных, либо вносил в нее данные.

Для загрузки новостей сначала запускается запрос со страницы на сервер:

async function getArticle(articleIdString: string)  
{  
 const articleResponse = await fetch(`/api/news?id=${articleIdString}`);  
  
 if (articleResponse.status === 200)  
 {  
 return (await articleResponse.json() as ArticleResponse).news;  
 }  
 else  
 {  
 return null;  
 }  
}

А затем сервер получает данные из базы данных:

[HttpGet]  
public async Task<IActionResult> GetArticle(long id)  
{  
 Models.Database.News? news = await dbContext.News.FindAsync(id);  
  
 if (news is null)  
 return NotFound();  
  
 return Ok(new { news });  
}

И отображаются на странице, показанной на рисунке 5.

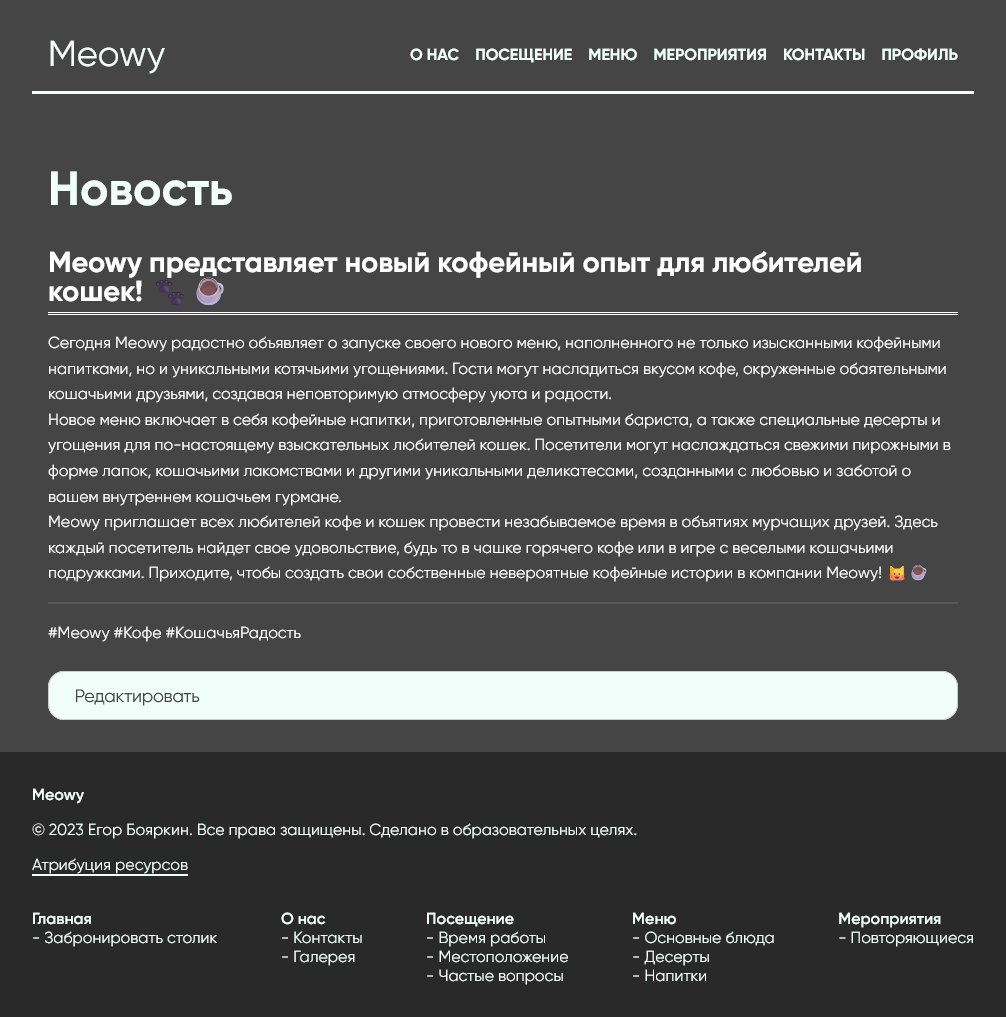


Рисунок 5 – Страница новости

Кнопка «Редактировать» появляется только тогда, когда пользователь авторизован.

Новости сохраняются после редактирования в базу данных следующим образом:

[HttpPut]  
[Authorize(Roles = RoleNames.Admin)]  
public async Task<IActionResult> UpdateArticle([FromQuery] long id, [FromBody] string news)  
{  
 try  
 {  
 await dbContext.News.Where(x => x.Id == id)  
 .ExecuteUpdateAsync(x => x.SetProperty(x => x.Content, news));  
 }  
 catch  
 {  
 return BadRequest();  
 }  
  
 return Ok();  
}

И редактируются на странице, показанной на рисунке 6.

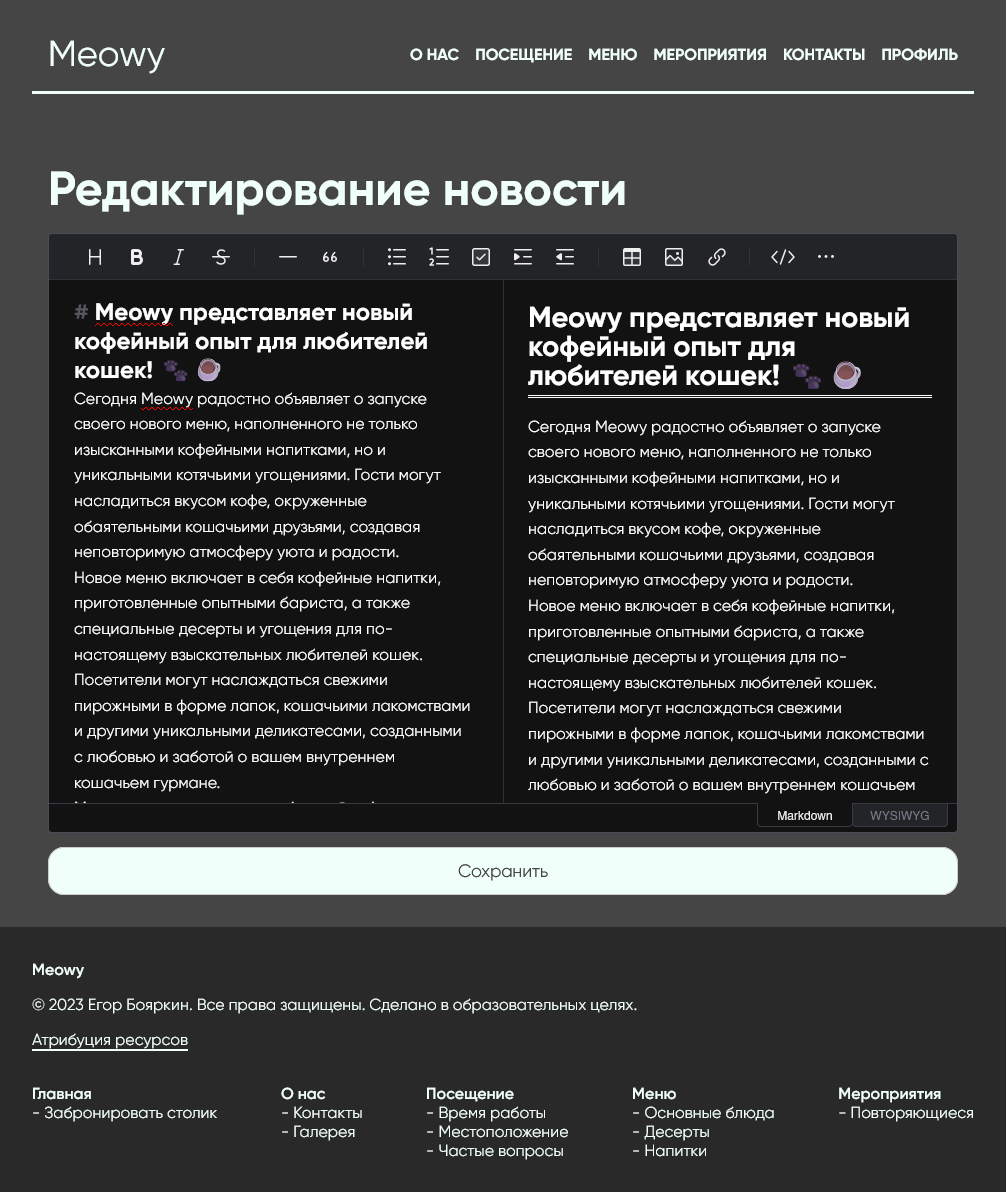


Рисунок 6 – Редактор новостей

Редактор может работать в двух режимах: Markdown и WYSIWYG. Markdown-режим был показан на рисунке 6, а WYSIWYG — What You See Is What You Get — показан на рисунке 7.

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 7 — WYSIWYG-режим

Также новости можно создавать в выше описанных двух режимах на странице, показанной на рисунке 8.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 8 — Страница создания новости

Код создания новой новости в базе данных:

[HttpPost]  
[Authorize(Roles = RoleNames.Admin)]  
public async Task<IActionResult> CreateArticle([FromBody] string news)  
{  
 Models.Database.News newsObj = new()  
 {  
 Title = "Новость",  
 AuthorId = Int64.Parse(User.FindFirst(ClaimTypes.NameIdentifier)!.Value),  
 Date = DateTimeOffset.UtcNow.UtcDateTime,  
 Content = news  
 };  
 dbContext.News.Add(newsObj);  
 try  
 {  
 await dbContext.SaveChangesAsync();  
 }  
 catch  
 {  
 return BadRequest();  
 }  
  
 return Created($"/news/{newsObj.Id}", null);  
}

После создания новости браузер перенаправляет на страницу с созданной новостью.

Страницы с редактированием и созданием новостей перенаправляют на страницу аутентификации, если пользователь не аутентифицирован, и возвращают ошибку 403, если пользователь не авторизован (не достаточно прав), то есть страница не загружается вовсе.

Заключение

В ходе разработки веб-приложения для котокафе были предприняты значительные усилия для создания инновационного и удобного пространства виртуального взаимодействия. Одним из ключевых достижений является внедрение базы данных, что позволило эффективно управлять информацией о посетителях, бронированиях, мероприятиях и других важных аспектах жизни заведения.

Создание объектной модели, отражающей базу данных, в системе, стало краеугольным камнем для унификации разработки, обеспечивая более легкую поддержку и расширение функционала приложения. Понимание взаимосвязей между ключевыми объектами, такими как бронирования, новости, мероприятия, пользователи и профили, позволило создать гибкую и масштабируемую архитектуру.

Формулировка запросов, представленных в теоретической части, способствовала обсуждению абстрактных концепций, что оказало положительное влияние на конечный результат проекта.

Освещение спорных идей в теоретической главе дало возможность взглянуть на проект с разных точек зрения, что позволяет принимать обоснованные решения в ходе разработки. Эти идеи стали катализаторами для размышлений о выборе между реляционными и NoSQL базами данных, стратегиями кэширования и вопросами безопасности данных.

В итоге, разработанное веб-приложение для котокафе представляет собой гармоничное сочетание теоретических концепций и практических решений. Этот проект не только соответствует заявленным целям, но и оставляет пространство для будущего роста и развития, что делает его примером современной индустрии веб-разработки.

Список использованных источников

1. PostgreSQL: Documentation: 15: PostgreSQL 15.5 Documentation // PostgreSQL URL: https://www.postgresql.org/docs/15/index.html (дата обращения: 30.11.2023).
2. Overview of Entity Framework Core - EF Core | Microsoft Learn // Microsoft Learn URL: https://learn.microsoft.com/en-us/ef/core/ (дата обращения: 30.11.2023).
3. Юсков И. О. Конспекты лекций по базам данных. - Москва: МТУСИ, 2023