Ши Чэнюэ, магистрант,
Амурский государственный университет

ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ СИСТЕМЫ УПРАВЛЕНИЯ ЖИЛЫМ КОМПЛЕКСОМ НА БАЗЕ ASP.NET И VUE.JS

Аннотация. В этой статье подробно описан процесс проектирования, внедрения и проверки системы управления общественной собственностью на основе ASP.NET Core и Vue.js. Благодаря использованию архитектуры B/S с разделением внешнего и внутреннего интерфейса было успешно создано модульное, масштабируемое и удобное для пользователя веб-приложение.

Ключевые слова: Система управления общественным имуществом, разделение фронтенда и бэкенда, модульная архитектура, ASP.NET Core, Vue.js.

Эта система использует архитектуру разделения клиентской и серверной частей на основе модели B/S. Эта архитектура четко разделяет логику приложения на уровень представления, уровень бизнес-логики и уровень сохранения данных, что повышает масштабируемость, удобство обслуживания и эффективность совместной работы системы.

Общая архитектура системы состоит из трех частей: клиента, интерфейсного сервера и внутреннего сервера. Клиент – это веб-браузер, используемый пользователем. На интерфейсном сервере размещается одностраничное приложение, созданное с помощью Vue.js, и оно обычно развертывается с использованием статического файлового сервера, такого как Nginx. На внутреннем сервере размещается веб-API RESTful, построенный на ASP.NET Core, и он отвечает за обработку основной бизнес-логики, проверку данных и операции сохранения. База данных работает как отдельный сервис. Общая архитектура системы представлена в таблице 1.

Таблица 1.

Архитектура системы

Слой	**Технологии**	**Ответственность**
Клиентский слой	Vue.js 3, Vue Router, Pinia, Axios, Element Plus	Пользовательский интерфейс, маршрутизация, управление состоянием
Сервер представления	Nginx	Обслуживание статических файлов, балансировка нагрузки
Сервер приложений	ASP.NET Core 6+, Web API, JWT Bearer	Бизнес-логика, аутентификация, RESTful API
Слой данных	Entity Framework Core, SQL Server	Работа с данными, миграции, ORM
Слой безопасности	JWT, RBAC, CORS, Валидация	Аутентификация, авторизация, защита данных

Конкретный рабочий процесс выглядит следующим образом: конечный пользователь получает доступ к системному URL-адресу в браузере, и браузер загружает статические ресурсы Vue.js SPA с интерфейсного сервера. После запуска SPA Vue Router отображает соответствующие компоненты страницы на основе текущего URL-пути. Когда компоненту требуются данные, он инициирует HTTP-запрос к веб-API ASP.NET Core, работающему на внутреннем сервере, через Axios. После получения запроса контроллер веб-API

взаимодействует с базой данных SQL Server через Entity Framework Core для выполнения операций запроса, вставки, обновления или удаления. Контроллер инкапсулирует результаты обработки в данные в формате JSON и возвращает их во внешний интерфейс через HTTP-ответ. После того как клиентское приложение получает данные JSON, оно использует адаптивную систему Vue.js для обновления статуса компонента, тем самым обеспечивая динамическое обновление пользовательского интерфейса. Весь процесс взаимодействия соединяет переднюю и внутреннюю части посредством четкого контракта API, обеспечивая разделение задач. На рисунке 1 представлена архитектура системы.

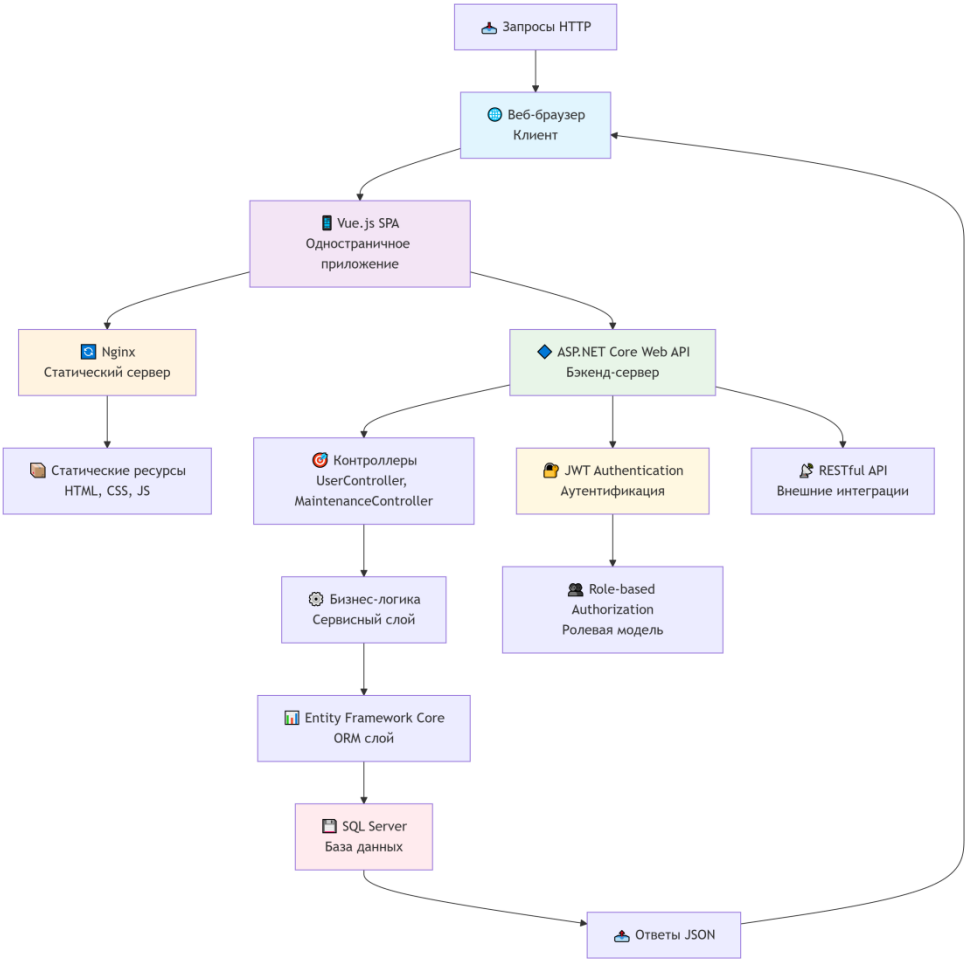

Рисунок 1.

Система разделена на три функциональных модуля в зависимости от ролевых разрешений: внешний модуль для владельцев и внутренний модуль управления для администраторов и управляющих недвижимостью. Функции каждого модуля разработаны вокруг основных бизнес-процессов управления недвижимостью.

Если взять в качестве примера основной модуль управления техническим обслуживанием, то его конструкция и реализация заключаются в следующем. На уровне базы данных таблица информации об обслуживании предназначена для хранения данных обо всем процессе ремонта. Таблица содержит такие поля, как уникальный идентификатор, связанный

идентификатор владельца, номера зданий и помещений, описание неисправности, время ремонта, статус ремонта, назначенный идентификатор администратора, время завершения и примечания по ремонту. Отслеживайте полный жизненный цикл заказа на ремонт с помощью перечисляемого значения поля статуса.

Во внутренней реализации был создан выделенный контроллер веб-API для предоставления ряда конечных точек RESTful. Например, конечная точка POST/api/maintenance используется для обработки нового заказа на ремонт, отправленного владельцем. Контроллер автоматически установит время ремонта на текущее время и инициализирует статус «Отправлено». Конечная точка PUT /api/maintenance/{id}/assign используется администраторами или управляющими недвижимостью для назначения задач обслуживания. Этот метод проверяет разрешения оператора и обновляет назначенный персонал и статус заказа на обслуживание. Все эти операции взаимодействуют с базой данных через Entity Framework Core и используют встроенный механизм привязки и проверки модели ASP.NET Core для обеспечения точности данных.

Что касается внешней реализации, были разработаны соответствующие компоненты Vue. Например, после установки компонента управления обслуживанием он вызовет интерфейс GET /api/maintenance через Axios для получения данных списка обслуживания и будет использовать компонент таблицы Element Plus для рендеринга. В таблице не только отображается основная информация, но и визуально отображается текущий прогресс с помощью меток состояния. Для каждой записи предусмотрены кнопки управления, такие как «Назначить». При срабатывании будет вызван соответствующий интерфейс API, а компонент Message будет использоваться для обратной связи с пользователем о результатах операции. Весь внешний интерфейс отзывчив и может адаптироваться к экранам разных размеров.

В процессе внедрения системы мы предложили решения ряда ключевых технических вопросов.

Аутентификация и авторизация личности являются основными проблемами в архитектуре разделения клиентской и внутренней частей. Эта система использует JSON Web Token в качестве механизма аутентификации без сохранения состояния. Конкретный процесс таков: когда пользователь входит в систему, внешний интерфейс отправляет учетные данные в API аутентификации; после прохождения внутренней проверки ключ используется для создания JWT, содержащего идентификатор пользователя и объявление роли, и возвращается во внешний интерфейс; внешний интерфейс хранит этот токен локально и переносит его в заголовок авторизации HTTP всех последующих запросов API; серверная часть автоматически проверяет достоверность токена в запросе через промежуточное программное обеспечение носителя JWT и создает контекст идентификации пользователя. На основании этого вы можете использовать атрибут Authorize в контроллере или методе действия для выполнения детального управления доступом с помощью ролевых политик.

Чтобы обеспечить стандартизацию и удобство обслуживания взаимодействия внешних и внутренних данных, разработан единый формат инкапсуляции ответов API. Все интерфейсы API возвращают стандартизированный объект JSON, содержащий статус успеха, подсказку и фактические полезные данные. Это позволяет интерфейсу обрабатывать все ответы API согласованным образом, упрощая обработку ошибок и логику управления состоянием.

Что касается безопасности, система была усилена на нескольких уровнях. Используя Entity Framework Core в качестве ORM, генерируемые им параметризованные запросы принципиально исключают риск атак с использованием SQL-инъекций. Vue.js по умолчанию экранирует HTML-данные, отображаемые в шаблоне, эффективно предотвращая атаки с использованием межсайтовых сценариев. Для сценариев, требующих ввода форматированного текста, для обработки вводится библиотека фильтрации содержимого

HTML на основе белого списка. Благодаря использованию аутентификации JWT без сохранения состояния система имеет естественный иммунитет к атакам с подделкой межсайтовых запросов.

Для обеспечения качества системы мы реализуем стратегию многоуровневого тестирования. На серверной стороне были написаны модульные тесты с использованием тестовой среды xUnit, а изолированные тесты проводились для основной бизнес-логики, такой как расчет стоимости, поток конечных автоматов и т. д. В то же время был создан интеграционный тест с использованием пакета Microsoft.AspNetCore.Mvc.Testing, имитирующий HTTP-клиент для выполнения сквозных вызовов к конечной точке API и проверяющий полную связь от запроса к операции с базой данных.

Для оценки производительности система прошла стресс-тестирование с использованием инструмента Apache JMeter. В тестовом сценарии моделируется, что 100 одновременно работающих пользователей постоянно получают доступ к типичному интерфейсу чтения с высокой степенью параллелизма «Получить список объявлений» в течение 5 минут. Результаты тестирования показывают, что при такой нагрузке среднее время ответа системы стабильно в пределах 200 миллисекунд, пропускная способность достигает более 500 запросов в секунду, а частота ошибок равна нулю. Этот результат доказывает, что система может бесперебойно поддерживать повседневные потребности доступа и использования сообщества среднего размера при текущей архитектуре и выборе технологий, а ее производительность соответствует ожиданиям.

Список литературы:

1. Макфедрис П. Веб-дизайн с нуля: HTML + CSS на практике, 2-е изд. / П. Макфедрис // Питер. – 2025.
2. Гоф Дж. Проектирование архитектуры API / Дж. Гоф // АЛИСТ. – 2024.