Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Жизненный цикл разработки программного обеспечения

ОТЧЕТ по лабораторной работе № 3
«Исследование архитектурного решения»

Студенты: Д.С. Тарбаев
А.С. Гутковский
Д.В. Кабачевский

Преподаватель: Д.А. Жалейко

МИНСК 2025

ВВЕДЕНИЕ

В данной лабораторной работе проводится исследование архитектурного решения для разрабатываемой системы. Архитектура программного обеспечения играет важную роль в создании надежных, масштабируемых и поддерживаемых приложений, определяя структуру системы, взаимодействие её компонентов и обеспечивая выполнение функциональных и нефункциональных требований. В рамках работы будут рассмотрены как теоретические аспекты проектирования архитектуры, так и практические шаги по анализу и улучшению существующего решения.

Работа разделена на три основные части. В первой части основное внимание уделяется проектированию архитектуры системы на высоком уровне абстракции. Будут изучены теоретические основы, описанные в «Руководстве Microsoft по моделированию приложений», а также определены ключевые аспекты, такие как тип приложения, стратегия развёртывания, выбор технологий, показатели качества и пути реализации сквозной функциональности. Результатом этой части станет структурная схема приложения, представленная в виде функциональных блоков или диаграмм UML, которая отражает архитектуру «To Be».

Во второй части работы проводится анализ существующей архитектуры на основе реального кода, используемого в системе. С помощью автоматизированных средств обратной инженерии будут сгенерированы диаграммы классов, отражающие текущее состояние системы. Это позволит получить представление об архитектуре «As Is» и выявить её особенности.

Третья часть работы посвящена сравнению архитектур «As Is» и «To Be». На основе выявленных различий будут предложены пути улучшения архитектуры, учитывающие принципы проектирования, архитектурные стили и шаблоны. Это позволит не только проанализировать текущее состояние системы, но и наметить направления для её дальнейшего развития и оптимизации.

1 ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ

1.1 Определения типа приложения

В данном случае разрабатывается веб-сервис "Туристическое агентство", состоящий из двух основных компонентов: веб-приложения (бэк-энд) и клиентского интерфейса (фронт-энд). Выбор данного типа приложения обусловлен следующими факторами:

- Веб-приложение (бэк-энд) отвечает за обработку данных, хранение информации о турах, клиентах и транзакциях, а также за предоставление API для взаимодействия с фронт-эндом.
- Клиентское приложение (фронт-энд) обеспечивает интуитивно понятный интерфейс для пользователя, позволяя ему искать и бронировать туры, получать рекомендации и управлять своими заказами.
- Основная логика обработки данных выполняется на серверной стороне, что снижает нагрузку на клиентское устройство и обеспечивает быструю реакцию системы.
- Хранение информации о турах и клиентах в централизованной базе данных позволяет легко управлять и обновлять данные.
- Бэк-энд разворачивается на сервере, что обеспечивает надежный доступ к данным через API для фронт-энда.
- Фронт-энд доступен для пользователей через веб-браузер, что исключает необходимость установки дополнительного ПО на устройства.
- Веб-архитектура позволяет легко добавлять новые функции и расширять функциональность приложения, не затрагивая клиентскую часть.

Таким образом, выбранное решение с разделением на веб-приложение (бэк-энд) и клиентский интерфейс (фронт-энд) обеспечивает удобство использования, надежность и масштабируемость системы "Туристическое агентство".

1.2 Выбор стратегии развёртывания

Использование Docker позволит создать изолированные контейнеры для фронтенда, бэкенда и базы данных, обеспечивая консистентную среду для разработки и развертывания. Бэкенд будет служить API, к которому фронтенд будет обращаться для получения данных и выполнения операций, что обеспечит четкое разделение логики приложения. Для автоматизации

процессов тестирования и развертывания целесообразно внедрить CI/CD, что позволит быстро интегрировать изменения и запускать тесты после каждого коммита. Хранение конфигураций и секретов в переменных окружения Docker поможет обеспечить безопасность и гибкость настройки приложения. При развертывании можно использовать облачные платформы, такие как AWS или Azure, что обеспечит масштабируемость и высокую доступность. Также, важно учесть мониторинг и логирование для отслеживания работы приложения и быстрого реагирования на возможные проблемы.

1.3 Обоснование выбора технологии

Обоснование выбора технологий для веб-приложения, использующего PostgreSQL, Spring Boot и React, основывается на нескольких ключевых моментах. PostgreSQL была выбрана в качестве системы управления базами данных из-за ее мощных возможностей и расширенной функциональности. Она поддерживает сложные запросы, транзакции и различные типы индексации, что особенно важно для приложений с большим объемом данных и требующих высокой производительности. Кроме того, PostgreSQL является открытым и бесплатным решением, что снижает затраты на лицензирование.

Spring Boot выбран для разработки бэкенда благодаря своей способности ускорять процесс разработки и упрощать конфигурацию приложений. Он предоставляет готовые шаблоны, которые позволяют сосредоточиться на бизнес-логике, а не на конфигурационных задачах. Spring Boot также поддерживает создание RESTful API, что позволяет легко интегрироваться с фронтенд-приложениями и сторонними сервисами. Его экосистема включает множество полезных библиотек и инструментов, что делает его идеальным выбором для создания стабильных и масштабируемых приложений.

React использован для фронтенда из-за своего компонентного подхода и высокой производительности при обновлении пользовательского интерфейса. Это позволяет создавать динамичные интерфейсы, которые быстро реагируют на действия пользователя. Компоненты React могут быть повторно использованы, что значительно ускоряет разработку и упрощает сопровождение кода. Кроме того, React имеет большую и активную сообщество, обеспечивая доступ к обширной экосистеме библиотек и ресурсов, что дополнительно упрощает процесс разработки.

1.4 Показатели качества

Качество веб-приложения можно оценивать по нескольким показателям, каждый из которых отражает различные аспекты его работы и пользовательского опыта.

Во-первых, производительность — это критически важный показатель, который включает время загрузки страниц, скорость отклика приложения и общую эффективность работы с данными. Чем быстрее приложение отвечает на запросы пользователей, тем выше его воспринимаемое качество.

Во-вторых, надежность — это способность приложения функционировать без сбоев и ошибок в течение длительного времени. Надежное приложение должно быть устойчивым к сбоям и обеспечивать сохранность данных, а также предоставлять пользователям четкую обратную связь в случае возникновения проблем.

В-третьих, безопасность — это ключевой аспект, особенно для приложений, работающих с конфиденциальной информацией. Приложение должно быть защищено от несанкционированного доступа, атак и утечек данных.

Четвертым показателем является удобство использования (юзабилити), которое определяет, насколько просто и интуитивно пользователю взаимодействовать с приложением. Хороший интерфейс должен быть понятным, доступным и обеспечивать максимальную эффективность выполнения задач.

Пятый показатель — это масштабируемость, которая определяет способность приложения справляться с увеличением нагрузки и числа пользователей без ухудшения производительности.

1.5 Решение о путях реализации сквозной функциональности

Для достижения надежности, безопасности и производительности веб-приложения можно использовать следующие механизмы и методики:

Протоколирование**:** Для централизованного журнала событий может использоваться фиксирование ключевых операций в системе, что упростит отладку и анализ поведения приложения. Это поможет быстро выявлять проблемы и улучшать функциональность.

Обработка исключений**:** Для обработки ошибок можно предусмотреть перехват их на границах слоев и централизованную обработку, что поможет предотвратить утечки данных и обеспечит равномерный процесс обработки ошибок для пользователей.

Связь между слоями**:** Для связи между фронтендом и бэкендом можно использовать REST API с поддержкой HTTP/HTTPS протоколов. Это

гарантирует защищенность передаваемых данных и минимизирует временные задержки в сетевых вызовах.

Асинхронные запросы**:** Для повышения производительности можно использовать асинхронные запросы, что улучшит отзывчивость интерфейса и сократит время ожидания от пользователей.

Кэширование**:** Для ускорения работы клиента можно применять кэширование, что позволит снизить нагрузку на сервер и минимизировать время отклика при повторных запросах к часто используемым данным.

Аутентификация и авторизация**:** Для обеспечения безопасности можно внедрить меры с использованием Spring Security, что позволит реализовать надежную аутентификацию пользователей и управление доступом к ресурсам.

Эти механизмы помогут повысить стабильность, безопасность и производительность веб-приложения, а также упростить поддержку и дальнейшее масштабирование.

1.6 Структурная схема приложения

Структурная схема приложения в виде функциональных блоков представлена на рисунке 1.1.

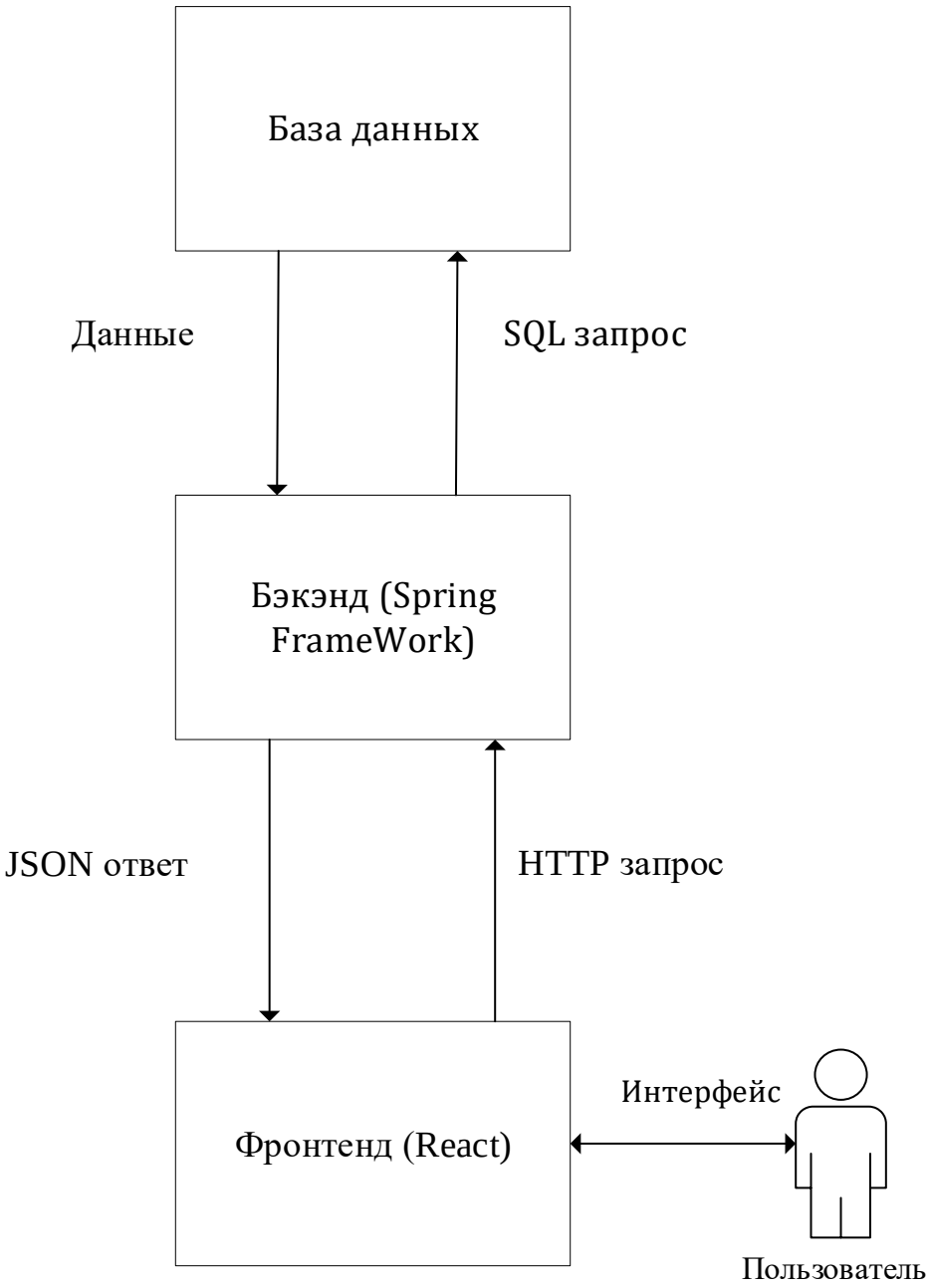

Рисунок 1.1 – Схема проекта

2 АНАЛИЗ АРХИТЕКТУРЫ

На данном этапе проведён анализ существующей архитектурысистемы, сформированной в ходе первого Sprint Review. С использованием инструмента обратной инженерии была сгенерирована диаграмма, отображающая структуру базы данных. Эта диаграмма представлена на рисунке 2.1.

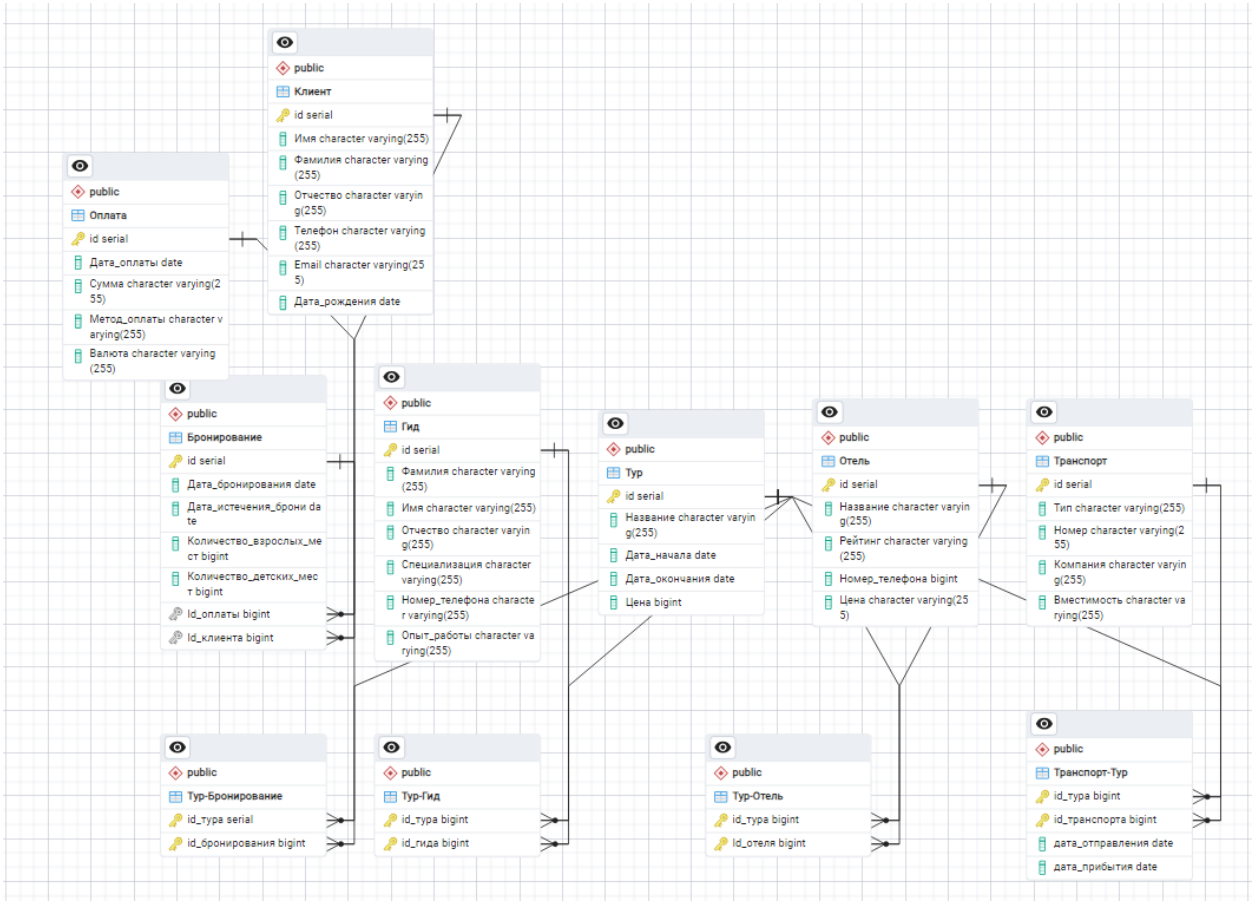

Рисунок 2.1 – EAR диаграмма базы данных

3 СРАВНЕНИЕ И РЕФАКТОРИНГ

3.1 Сравнение «As is» и «To be»

As is (текущая архитектура)

Фронтенд:

- отсутствует

Бэкенд:

- Принимает запросы от фронтенда и обрабатывает их.
- Взаимодействует с базой данных для хранения и извлечения данных.

База данных:

- Хранит все данные, связанные с диаграммами и пользователями.
- Обеспечивает надежное сохранение и извлечение данных.

Взаимодействие между компонентами:

- Пользователь обращается к бэкэнду посредством HTTP и получает в ответ сущность в виде json файла.
- Бэкэнд обращается к базе данных с помощью sql запросов.

Производительность:

- отсутствуют механизмы улучшающие производительность

Безопасность и управление доступом:

- отсутствуют механизмы обеспечения безопасности и контроля прав доступа

To be (целевая архитектура)

Структурные компоненты:

Фронтенд:

- Ответственный за взаимодействие с пользователем.
- Обрабатывает ввод данных и отображает результаты.
- Использует асинхронные запросы для взаимодействия с бэкендом.

Бэкенд:

- Принимает запросы от фронтенда и обрабатывает их.
- Взаимодействует с базой данных для хранения и извлечения данных.
- Реализует централизованную обработку ошибок и протоколирование для улучшения отладки.

База данных:

- Хранит все данные, связанные с диаграммами и пользователями.
- Обеспечивает надежное сохранение и извлечение данных.

Взаимодействие между компонентами:

- Пользователь вводит данные во фронтенде.
- Фронтенд отправляет запросы на создание или обновление диаграммы к бэкенду.
- Бэкенд обрабатывает эти запросы, взаимодействует с базой данных для сохранения или извлечения диаграмм.
- Бэкенд отправляет результаты (подтверждения или данные диаграммы) обратно на фронтенд.
- Фронтенд отображает полученные данные пользователю.

Безопасность и управление доступом:

- Внедряется аутентификация и авторизация с использованием подходов, таких как Spring Security, для защиты данных и управления доступом пользователей.

Производительность:

- Применяются механизмы кэширования для ускорения доступа к часто запрашиваемым данным.
- Асинхронные вызовы в фронтенде для улучшения отзывчивости интерфейса.

Расширяемость и поддержка:

- Код комментируется и документируется для упрощения понимания и поддержки.
- Реализуется модульная архитектура, позволяющая добавлять новые функции или компоненты без значительных изменений в существующем коде.

Протоколирование и мониторинг:

- Настраивается централизованное логирование, что упрощает отладку и анализ событий в приложении.

3.2 Выделенные отличия и их причины

Отличие	Причина
Отсутствие безопасности	Нехватка времени
Отсутствие механизмов улучшения производительности	Нехватка времени
Отсутствие фронтенда	Нехватка времени

Отсутствие документирования	Нехватка времени
Отсутствие логирования	Нехватка времени

3.3 Пути улучшения архитектуры

Для преодоления существующих недостатков и достижения желаемого состояния ("to be") архитектуры веб-сервиса, можно реализовать следующие пути улучшения:

1. **Обеспечение безопасности:**
   - Внедрение аутентификации и авторизации на уровне бэкенда с использованием таких технологий, как Spring Security.
   - Реализация HTTPS для всех сетевых взаимодействий для защиты данных на этапе передачи.
   - Регулярное обновление зависимостей и компонентов для защиты от известных уязвимостей.
2. **Улучшение производительности:**
   - Внедрение кэширования для ускорения доступа к часто запрашиваемым данным и снижению нагрузки на сервер.
   - Оптимизация запросов к базе данных для уменьшения времени отклика и повышения эффективности.
   - Использование асинхронных запросов в интерфейсе для повышения отзывчивости пользовательского опыта.
3. **Разработка фронтенда:**
   - Создание интуитивно понятного и отзывчивого пользовательского интерфейса с современными библиотеками и фреймворками (например, React, Vue.js).
   - Интеграция с бэкендом через REST API или GraphQL для более гибкого взаимодействия.
4. **Документирование кода и архитектуры:**
   - Создание документации для архитектурных решений, API и классов кода с использованием инструментов, таких как Swagger или Javadoc.
   - Регулярные обновления документации по мере внесения изменений, чтобы сохранить актуальность информации.
5. **Внедрение логирования:**
   - Реализация централизованного логирования с использованием таких инструментов, как Log4j или SLF4J, для отслеживания событий и ошибок.

- Настройка мониторинга логов с использованием платформ для анализа логов (например, ELK Stack), что позволит легко находить и устранять проблемы.

6. **Оптимизация управления временем:**
   - Разработка четкого плана разработки с выделением временных рамок для каждого компонента и реализуемой функции.
   - Применение методик управления проектами, таких как Agile или Scrum, для увеличения прозрачности работ и улучшения командной работы.