Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____О.В. Непомнящий
«___»_____2024 г.

БАКАЛАВРСКАЯ РАБОТА

090301 Информатика и вычислительная техника

Мобильное приложение – трекер дня. Серверная часть.

Руководитель	_____	_____	доцент, канд. тех. наук	С.Н. Титовский
	подпись	*дата*	*должность, ученая степень*	
Выпускник	_____	_____		А.В. Чередниченко
	подпись	*дата*		
Нормоконтролёр	_____	_____	доцент, канд. тех. наук	С.Н. Титовский
	подпись	*дата*	*должность, ученая степень*	

Красноярск 2024

РЕФЕРАТ

Выпускная квалификационная работа по теме «Мобильное приложение – трекер дня. Серверная часть» содержит 31 страницу текстового документа, 9 рисунков, 4 таблицы, 11 использованных источников.

ТРЕКЕР ДНЯ, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ПЕРСОНАЛИЗАЦИЯ, УПРАВЛЕНИЕ ВРЕМЕНЕМ, POSTGRESQL, GO, REST API.

В условиях современной жизни, где информация и технологии играют ключевую роль, люди сталкиваются с потребностью эффективного управления временем, поддержание здорового образа жизни и достижение личных и профессиональных целей. Большинство существующих приложений предлагают стандартные решения, которые не учитывают индивидуальные потребности каждого пользователя. Это приводит к недостаточной мотивации и эффективности в использовании таких приложений.

Объектом исследования является система управления задачами и временем.

Целью выпускной квалификационной работы является разработка персонализированного трекера дня, который сможет удовлетворить потребности пользователя. Разработка должна помочь в дальнейшем для использования в личном проекте.

Задачи:

- провести анализ существующих решений;
- выявить функциональные требования;
- спроектировать структуру приложения;
- реализовать функциональные требования;
- реализовать серверную часть приложения.

Результатом выпускной квалификационной работы является разработанная серверная часть для мобильного приложения «Трекер дня».

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С появлением мобильных технологий и персональных электронных устройств произошли значительные изменения в жизни людей. Одной из ключевых задач современного человека стало эффективное управление своим временем и задачами. В условиях постоянно растущих требований и множества отвлекающих факторов люди все больше нуждаются в инструментах, которые помогут им структурировать и оптимизировать свои повседневные дела.

Персонализированные трекеры дня или трекеры задач стали важными инструментами для достижения этой цели. Они позволяют пользователям отслеживать свои задачи, планировать время и контролировать различные условия своей жизни, учитывая индивидуальные предпочтения и потребности. Такие приложения особенно полезны для тех, кто стремится повысить свою продуктивность, улучшить организацию времени и достичь поставленных целей.

Целью данной выпускной квалификационной работы является разработка персонализированного трекера дня, который сможет удовлетворить потребности пользователя. Приложение будет разрабатываться с использованием языка программирования Go и фреймворка Gin, а также базы данных PostgreSQL. Это позволит создать надежное и масштабируемое решение, обеспечивающее высокую производительность и удобство использования.

Для достижения поставленной цели были сформулированы следующие задачи:

- провести анализ существующих решений;
- выявить функциональные требования;
- спроектировать структуру приложения;
- реализовать функциональные требования;
- реализовать серверную часть приложения.

1 Анализ предметной области

В первую очередь необходимо провести анализ целевой аудитории, их потребности и предпочтения. Это включает в себя понимание того, какие функции и возможности пользователи хотели бы видеть в приложении «Трекер дня», а также выявление недостатков и ограничений существующих аналогов. Важно учитывать разнообразие целей и ожиданий различных групп пользователей, таких как подростки, молодежь и взрослые, чтобы создать действительно персонализированное и универсальное решение. Изучение мнений и обратной связи от пользователей поможет выявить ключевые функции, которые необходимы для повышения личной эффективности, управления временем и достижения личных и профессиональных целей.

1.1 Анализ целевой аудитории

Понимать, кто целевая аудитория продукта, важно на любом этапе проекта: от идеи до продвижения. При этом важно понимать, что целевая аудитория – это группа людей, которым потенциально может быть интересен продукт или услуга и которые имеют возможность его приобрести [1].

Результатом анализа стала следующая целевая аудитория:

– подростки (12-17 лет), у которых есть необходимость в эффективном управлении временем, для отслеживания выполнения домашних заданий, проектов и подготовку к экзаменам;

– молодежь (17-35 лет), которая планирует учебные, рабочие и личные задачи, а также прогресс в достижении поставленных целей;

– взрослые (35+ лет), которым важно планирование рабочих задач, встреч и личных дел, а также поддержка профессионального развития. Обзор существующих решений.

Большой популярностью пользуются разработки, направленные на организацию, улучшение и упрощение управления временем и задачами. Как

пример можно привести множество приложений для отслеживания времени, планирования задач и инструменты для совместного управления проектами. На данный момент можно уверенно утверждать, что такие сервисы востребованы у людей различных профессий, возрастов и сфер деятельности.

Однако в открытых источниках не удалось обнаружить универсального инструмента для комплексного управления временем, который бы полностью соответствовал всем требованиям пользователей. Существующие решения, такие как отдельные приложения для треккинга времени, планировщик задач и отслеживания самочувствия имеют свои преимущества и недостатки, но не всегда обеспечивают необходимую гибкость.

Таким образом, возможно выделить следующие критерии для сравнения существующих решений:

1) информативность – возможность полного и точного учета времени, затраченного на различные задачи и проекты, с детальным описанием каждой активности, указанием даты и времени начала и завершения, а также возможностью добавления примечаний и тегов;

2) полнота – возможность для пользователя получить полную информацию о своей занятости, включая время, затраченное на работу, отдых, личные дела и другие активности;

3) доступность – удобство использования приложения, доступность на разных устройствах (мобильных, веб-версиях), а также простота регистрации и интеграции с другими сервисами и приложениями.

1.2 Обзор существующих решений

Мобильное приложение «To-do List»

"To-do List" – мобильное приложение, предоставляющее удобный инструмент для управления задачами и планирования времени. Пользователи могут создавать заметки, которые автоматически привязываются к календарю, обеспечивая систематизацию и хронологию задач.

Мобильное приложение «TickTime»

"TickTime" представляет собой мобильное приложение, фокусирующееся на точном измерении времени и предоставление пользователям удобных инструментов для эффективного управления своим рабочим временем. Основные характеристики приложения включают:

Мобильное приложение «Tusk»

"Tusk" представляет собой многофункциональное мобильное приложение, ориентированное на эффективное управление задачами и повышение продуктивности пользователей.

Сравнительный анализ представлен в таблице 1:

Таблица 1 – Анализ существующих решений

Характеристика	Трекер дня	To-do List	TickTime	Tusk
Напоминания задач	Есть	Есть	Нет	Есть
Виджеты для главного экрана	Есть	Есть	Нет	Нет
Список задач	Есть	Есть	Нет	Есть
Просмотр календаря	Есть	Есть	Есть	Есть
Изменение настроек	Есть	Есть	Нет	Есть

1.3 Выявление функциональных требований

Личный кабинет пользователя. После авторизации пользователь получает доступ в личный кабинет, в котором может редактировать информацию о себе. Личный кабинет позволяет пользователю управлять своими данными и настройками для более точной персонализации приложения.

Лента задач и мероприятий. Каждая задача или мероприятие содержит поля: название, категория (учеба, работа, здоровье, отдых и т.д.), дата и время выполнения, продолжительность, приоритет, напоминания, описание, ссылки на связанные ресурсы или документы. Лента задач и мероприятий доступна

всем зарегистрированным пользователям, пользователи могут фильтровать сортировать задачи и мероприятия по параметрам.

Функциональные требования администратора:

– управление пользователями (создание, редактирование, удаление). После авторизации администратор получает доступ к панели управления пользователями. В этой панели администратор может создавать новых пользователей, редактировать информацию существующих пользователей и удалять пользователей из системы. Для создания пользователя администратор нужно ввести необходимую информацию: имя, адрес электронной почты, роль (пользователь или администратор), пароль. При редактировании информации администратора доступны поля: имя, адрес электронной почты, роль и пароль. Удаление пользователя из системы требует подтверждения от администратора для предотвращения случайных удалений;

– управление задачами и мероприятиями. Администратор имеет возможность управлять задачами и мероприятиями всех пользователей. Это включает создание, редактирование и удаление задач и мероприятий. Каждая задача или мероприятие содержит поля для описания: название, категория (учеба, работа и т.д.), дата и время выполнения, продолжительность, приоритет, напоминания. Администратор может фильтровать и сортировать задачи по различным параметрам для удобства управления. Также администратор может назначать задачи определённым пользователям или группам пользователей;

– настройки системы и прав доступа. Администратор управляет настройками системы и правами доступа пользователей. Это включает настройку глобальных параметров системы, таких как язык интерфейса, часовой пояс, параметры уведомлений и другие общие настройки. Администратор также управляет правами доступа пользователей, устанавливая уровни доступа и разрешения для различных ролей (пользователь, администратор). Это позволяет контролировать, какие

функции и данные доступны различным категориям пользователей, обеспечивая безопасность и конфиденциальность информации.

1.4 Выводы

В данной главе был проведен анализ предметной области, определены целевая аудитория и их потребности, а также рассмотрены существующие решения в области тайм-трекеров. На основе анализа целевой аудитории были выявлены ключевые группы пользователей: подростки, молодежь и взрослые, каждая из которых имеет свои специфические потребности и предпочтения в управлении временем и задачами.

2 Проектирование сервера

2.1 Выбор основного инструмента

Для разработки серверной части мобильного приложения «Трекер дня» был выбран язык программирования Go. Этот выбор обусловлен несколькими ключевыми факторами, которые способствуют достижению целей данного проекта.

Язык программирования Go был выбран из-за его высокой производительности, поддержки параллелизма, простоты и надежности. Будучи компилируемым языком, Go обеспечивает высокую скорость выполнения кода, что особенно важно для серверных приложений, обрабатывающих большое количество запросов. Встроенная поддержка параллелизма через каналы позволяет эффективно управлять многозадачностью и использовать возможности многопроцессорных систем. Простота синтаксиса Go снижает вероятность ошибок и делает код более читаемым и поддерживаемым. Строгая типизация и механизмы управления памятью, такие как сборка мусора, повышают надежность и безопасность приложения. Активное сообщество и богатая экосистема библиотек и инструментов ускоряют разработку и облегчают решение различных задач проекта.

2.2 Среда разработки

Поскольку для разработки проекта был выбран язык Go, необходимы IDE, поддерживающие именно этот язык. Самыми популярными IDE для Go стали Visual Studio Code и GoLand.

Visual Studio Code – это бесплатная, открытая и простая среда разработки, предназначенная для различных языков программирования, включая Go. VS Code завоевал популярность благодаря своей простоте, настраиваемому интерфейсу, простому дизайну и обширным библиотекам.

Самым большим преимуществом VS Code является его модульность и возможность настройки под конкретные нужды разработчика.

GoLand, с другой стороны, представляет собой очень сложную среду разработки со множеством различных панелей и инструментов. Эта среда разработки от JetBrains разработана специально для языка Go, что позволяет предлагать мощные и специализированные инструменты разработки.

Сравнив оба IDE был сделан выбор в сторону Visual Studio Code благодаря своей универсальности, расширяемости и удобству использования. VS Code поддерживает множество языков программирования, включая Go, через плагины, что делает его универсальным инструментом для разработки. Встроенная поддержка Git облегчает управление версиями кода и координацию работы в команде. Расширяемость редактора позволяет добавлять новые функции и интеграции, такие как авто дополнение, отладка и тестирование кода на Go, что значительно упрощает процесс разработки. Удобный и интуитивно понятный интерфейс VS Code повышает эффективность работы с кодом. Бесплатное предоставление лицензий и кроссплатформенность редактора позволяют разработчикам работать на любой операционной системе, что способствует гибкости и удобству работы над проектом.

2.3 Разработка функциональных возможностей

Диаграммы прецедентов составляют модель прецедентов (вариантов использования). Прецедент – это функциональность системы, позволяющая пользователю получить некий значимый для него, ощутимый и измеримый результат [5]. На рисунке 1 и 2 представлена диаграмма прецедентов разрабатываемого приложения.

Рисунок 1 – Диаграмма прецедентов пользователя

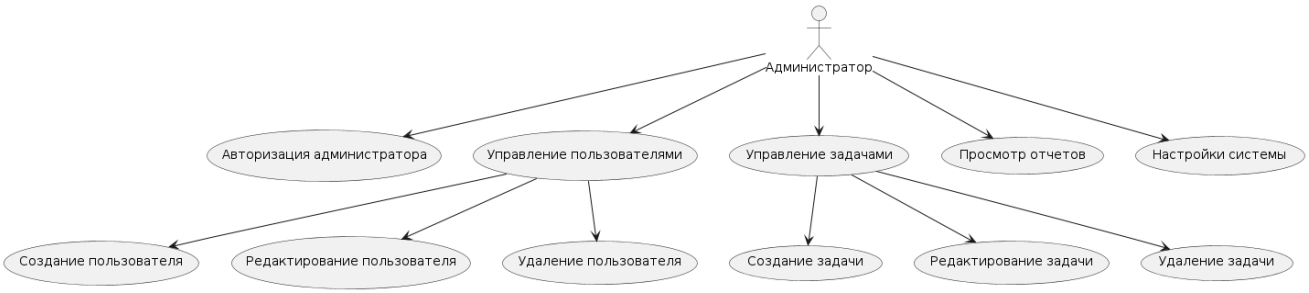

Рисунок 2 – Диаграмма прецедентов администратор

В таблице 2 представлено описание блоков диаграммы прецедентов.

Таблица 2 – Описание прецедентов.

Название прецедента	**Описание**	**Основной сценарий**	**Альтернативный сценарий**
Вход в систему	Пользователь осуществляет подключение к системе	Успешный вход в систему	Ничего не введено
Предоставление расписания	Пользователь открывает свое расписание	Пользователю выводится расписание	Нет
Предоставление профиля	Пользователь открывает информацию о своем профиле	Пользователю выводится данные профиля пользователя	Нет
Изменение настроек	Пользователь изменяет настройки	Система применяет изменения и подтверждает	Нет
Управление данными	Администратор открывает управлению данными	Система предоставляет доступ к данным	Сообщение об ошибке

Окончание таблицы 2

Название прецедента	Описание	Основной сценарий	Альтернативный сценарий
Просмотр данных	Администратор просматривает данные	Администратору выводятся данные для просмотра	Данные не найдены
Обновление данных	Администратор обновляет данные	Система обновляет данные и подтверждает	Выводится об ошибке при обновлении данных
Удаление данных	Администратор удаляет данные	Система удаляет данные и подтверждает	Выводится об ошибке при удалении данных
Предоставление блоков интересов	Пользователь запрашивает информацию о блоках интересов	Открывается блоки с его интересом	Нет

2.4 Разработка структуры базы данных

Для реализации проекта была выбрана PostgreSQL – это объектно-реляционная система управления базами данных. СУБД позволяет гибко управлять базами данных (БД). С ее помощью можно создавать, модифицировать или удалять записи, отправлять транзакцию – набор из нескольких последовательных запросов на особом языке запросов SQL [6].

Атрибуты сущностей

Сущность – тип объектов, которые должны храниться в базе данных. Каждая таблица в базе данных должна представлять одну сущность. Как правило, сущности соответствуют объектам из реального мира.

У каждой сущности определяют набор атрибутов. Атрибут представляет свойство, которое описывает некоторую характеристику объекта.

Каждый столбец должен хранить один атрибут сущности. А каждая строка представляет отдельный объект или экземпляр сущности [7].

Атрибуты сущностей представлены в таблица 3.

Таблица 3 – Атрибуты сущностей

Атрибут	Назначение	Тип	Ограничения
Users			
ID	Идентификатор	Bigint	Primary key
name	Имя	Text	Not null
email	Логин	Text	Not null
password	Пароль	Text	Not null
settings			
ID	Идентификатор	Bigint	Primary key
user_id	Идентификатор пользователя	Bigint	Foreign key
theme	Тема интерфейса	Text	Not null
notifications	Уведомления	Text	Not null
profile			
ID	Идентификатор	Bigint	Primary key
user_id	Идентификатор пользователя	Bigint	Foreign key
bio	Биография пользователя	Text	Not null
avatar_url	Аватар пользователя	Text	Not null
interest_blocks			
ID	Идентификатор	Bigint	Primary key
user_id	Идентификатор пользователя	Bigint	Foreign key
title	Название	Text	Not null
daily_schedule			
ID	Идентификатор	Bigint	Primary key
user_id	Идентификатор пользователя	Bigint	Foreign key
date	Дата	Timestamp without time zone	Not null
task	Описание	Text	Not null
notes			
ID	Идентификатор	Bigint	Primary key
user_id	Идентификатор пользователя	Bigint	Foreign key
interest_block_id	Идентификатор блока с интересами	Bigint	Not null
content	Заметки	Text	Not null

На рисунке 3 приведено изображение взаимосвязи базы данных.

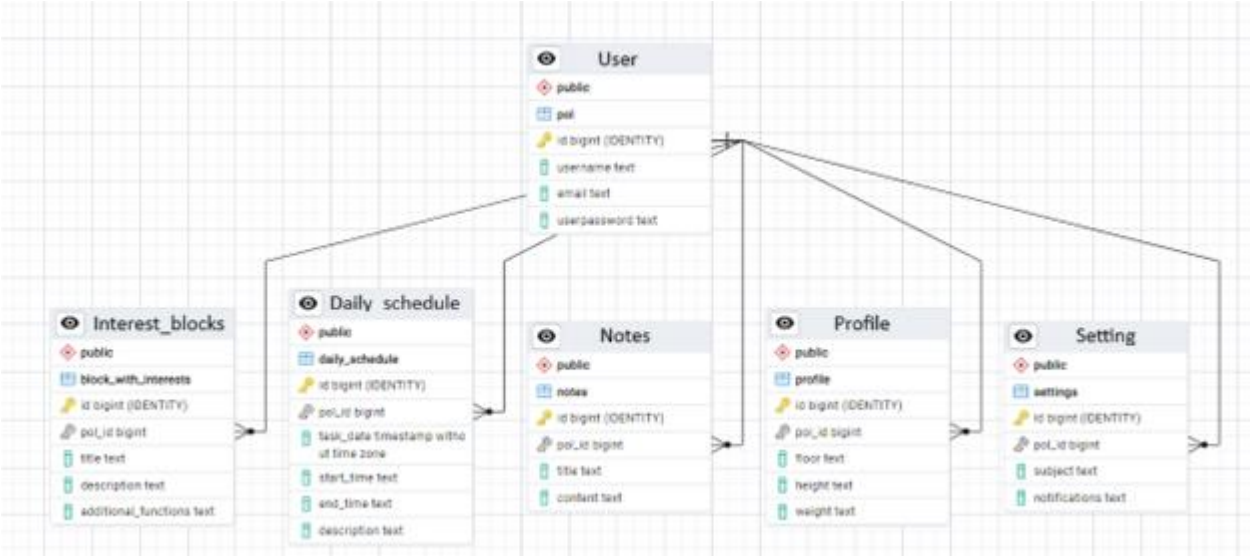

Рисунок 3 – Взаимосвязь базы данных

Связующая таблица содержит столбцы-ссылки на первичные ключи и позволяет связать между собой несколько записей из одной таблицы с несколькими записями из другой таблицы.

Использование связующей таблицы позволяет эффективно управлять данными об интересах и статистиках и обеспечивает гибкость при построении запросов и анализе данных. Это удобно для визуализации своих блоков, которые содержат определенные интересы пользователя.

Индексы

Индекс – объект базы данных, создаваемый с целью повышения производительности поиска данных.

В базу данных входят индексы:

а) Таблица «pol»:

– pol_id – индекс для ускорения поиска по внешнему ключу;

– email – индекс для ускорения поиска по email.

б) Таблица «block_with_interests»:

– pol_id – индекс для ускорения поиска по внешнему ключу.

в) Таблица «daily_schedule»:

– pol_id – индекс для ускорения поиска по внешнему ключу;

– task_date – индекс для ускорения поиска по дате задачи.

г) Таблица «notes»:

– pol_id – индекс для ускорения поиска по внешнему ключу.

д) Таблица «profile»:

– pol_id – индекс для ускорения поиска по внешнему ключу.

е) Таблица «settings»:

– pol_id – Индекс для ускорения поиска по внешнему ключу.

Триггеры

Триггер – хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением INSERT, удалением DELETE строки в заданной таблице, или изменением UPDATE данных в определённом столбце заданной таблицы реляционной базы данных.

Данные триггеры используются в:

– daily_schedule – (UPDATE) Обновление поля end_time текущей датой и временем;

– profile – (INSERT) Создание новой записи в profile при добавлении новой записи в pol.

2.5 Архитектура приложения

Архитектура серверной части мобильного приложения построена по принципу клиент-серверной архитектуры – это система распределения, в которой одни устройства обращаются к другим для получения необходимой информации или выполнения определенных задач. Первые из них называются клиентами, вторые – серверами [8].

К числу основных достоинств архитектуры относятся:

– централизованное управление. Предполагает надежный контроль над данными и ресурсами, что упрощает администрирование, обновление информации;

– распределение нагрузки. Обрабатывается несколько запросов от различных клиентов одновременно. Это позволяет рационализировать ресурсы;

– безопасность. Доступ к данным контролируется центральной системой, что обеспечивает авторизацию и аутентификацию пользователей.

На рисунке 4 показана схема взаимодействия компонентов в клиент-серверной архитектуре.

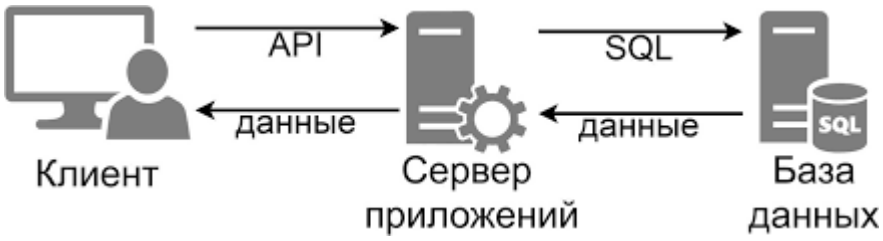

Рисунок 4 – Клиент серверная архитектура

Эта архитектура обеспечивает разделение ответственности между различными компонентами системы, что упрощает разработку, тестирование и сопровождение приложения.

Внутри сервера используется многослойная архитектура, которая помогает структурировать и организовать внутреннюю логику сервера.

Многоуровневая клиент-серверная архитектура имеет следующие отличительные особенности:

– более сложная структура с несколькими серверными и клиентскими уровнями;

– каждый из уровней отвечает за определенные функции и способен быть физически разделен на разные компоненты;

– позволяет гибко масштабировать систему и распределять нагрузку между серверами.

Эта архитектура разбивает сервер на несколько логических слоев, каждый из которых выполняет определенные функции. Подобное разделение позволяет лучше управлять сложностью системы, повышает модульность и упрощает процесс разработки, тестирования и сопровождения приложения.

В данной архитектуре можно выделить несколько ключевых слоев и компонентов:

– слой маршрутизации (Routing Layer) – этот слой отвечает за обработку входящих HTTP-запросов и их маршрутизацию к соответствующим обработчикам;

– слой обработчиков (Handler Layer) – этот слой выполняют роль контроллеров в архитектуре приложения. Они обрабатывают запросы, взаимодействуют с сервисами и формируют HTTP-ответы. Каждый обработчик отвечает за определенный тип запросов, таких как создание пользователя, получение задач и т.д;

– слой сервисов (Service Layer) – сервисы содержат бизнес-логику приложения. Они взаимодействуют с репозиториями для выполнения операций с данными и предоставляют обработчикам результаты выполнения этих операций;

– слой репозиториев (Repository Layer) – репозитории отвечают за доступ к базе данных. Они содержат методы для выполнения CRUD-операций (создание, чтение, обновление, удаление) над различными сущностями. Взаимодействие с базой данных осуществляется через стандартные драйверы и библиотеки для работы с PostgreSQL;

– слой моделей (Model Layer) – модели представляют собой структуры данных, соответствующие таблицам базы данных. Они используются для обмена данными между слоями репозиториев, сервисов и обработчиков. Модели также могут содержать методы для валидации данных.

2.6 Взаимодействие клиент–сервер

Практически все современное ПО построено на принципе взаимодействия клиент-сервер – это способ обмена информацией между двумя компьютерами, где одна сторона (клиент) запрашивает данные у серверной части. Последняя формирует ответ, направляя его в обмен [9].

Одно из главных преимуществ – это возможность распределения функций между сторонами в связке. Клиент может выполнять легкие задачи, такие как отображение информации на экране, в то время как сервер может обрабатывать более сложные задачи, к примеру, вычисления и хранение данных. Другими же плюсами являются:

– централизованное управление данными и услугами, что упрощает их обслуживание, обновление и безопасность;

– клиенты могут пользоваться любыми системами и устройствами, что делает архитектуру универсальной и доступной для широкого круга пользователей;

– клиенты вправе обратиться к серверу через сеть, что обеспечивает географическую гибкость и позволяет пользователям работать удаленно;

– эффективное управление помогает оптимизировать нагрузку и гарантирует быстрый отклик приложений;

– сервер может предоставлять общий доступ к данным, что облегчает совместное использование информации между пользователями;

– архитектура активно контролирует безопасность и доступность информации, так как централизованный сервер может реализовывать строгие политики шифрования.

Главным недостатком данного решения считается возможность сбоев в работе сервера или сети.

Механизм же взаимодействия сторон выглядит следующим образом:

а) клиент при соединении отсылает запрос: например, через сеть Интернет;

б) сервер его принимает и производит анализ, пытаясь понять, что нужно отправителю;

в) сервер формирует ответ или извлекает информацию из базы данных;

г) итоговые данные отправляются обратно клиенту;

д) клиент принимает информацию и выводит ее на экран пользователя, может продолжить взаимодействие.

Такой подход позволяет эффективно организовать обмен информацией и ресурсами в сети. Диаграмма последовательности взаимодействия клиента и сервера представлена на рисунке 5.

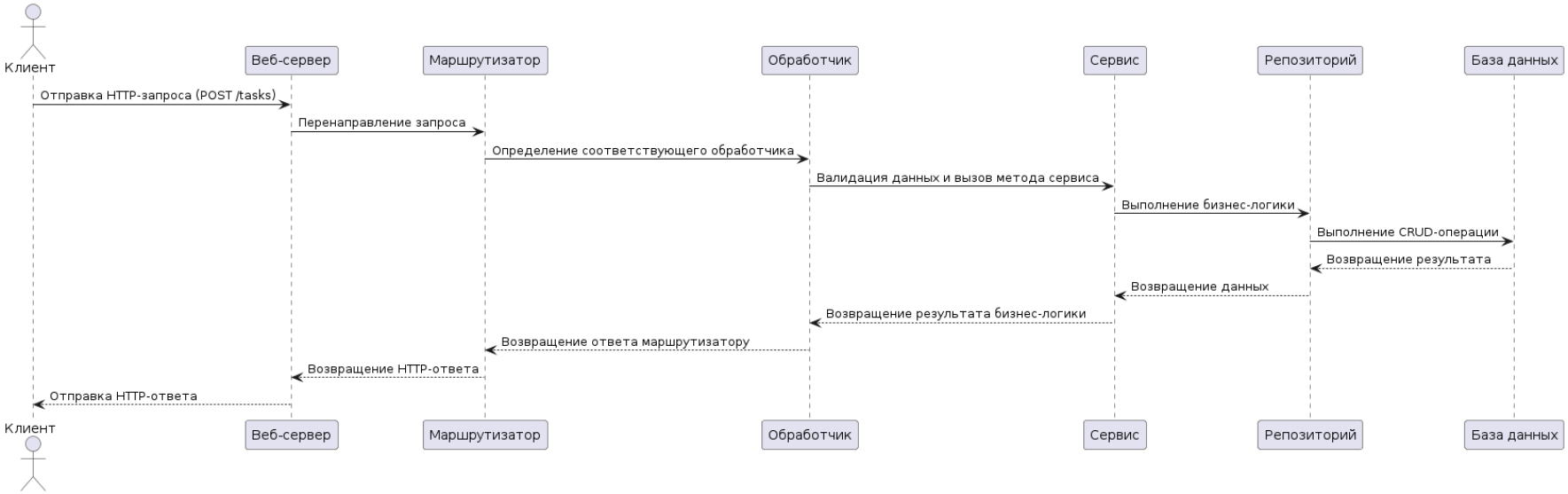

Рисунок 5 – Диаграмма последовательности взаимодействия клиент-сервер

Методы взаимодействия клиента и сервера представлена в таблице 4.

Таблица 4 – Таблица взаимодействия клиента и сервера

Метод	**Маршрут**	**Описание**
Get	/tasks	Получение всех задач
POST	/tasks	Добавление новой задачи
PATCH	/tasks/:id	Обновление задач
DELETE	/tasks/:id	Удаление задач

2.7 Выводы

В данной главе была разработана и описана структура базы данных и архитектура серверной части. Приведенные диаграммы классов и описания слоев архитектуры иллюстрируют, как различные компоненты системы взаимодействуют друг с другом для обеспечения функциональности приложения.

3 Реализация серверной части приложения

В этой главе подробно рассмотрен процесс реализации серверной части персонализированного трекера дня, включая методы, алгоритмы и структуры данных, используемые для достижения целей проекта.

3.1 REST API

REST API – это архитектурный подход, который устанавливает ограничения для API: как они должны быть устроены и какие функции поддерживать. Позволяет стандартизировать работу программных интерфейсов, сделать их более удобными и производительными [11]. Система, отвечающая принципам REST, называется RESTful – системой, и должна удовлетворять следующим ограничениям:

– клиент-серверная модель. RESTful-система должна производить операции в клиент-серверной модели, разделяя интерфейсы и возможности масштабирования;

– отсутствие состояния. Взаимодействие между клиентом и сервером происходит таким образом, что сервер не хранит состояние клиента между запросами;

– единообразие интерфейса. Сервер и клиент взаимодействуют через стандартизированные интерфейсы;

– кэширование. Ответы сервера должны быть помечены как кэшируемые или некэшируемые;

– система слоёв. RESTful – система может иметь большее количество компонентов, чем клиент и сервер, при этом каждый компонент ограничен способностью видеть только соседний слой;

– код по требованию (опционально). Сервер может передавать исполняемый код клиенту.

Обмен данных между frontend и backend через REST API позволяет приложению функционировать гладко и эффективно. REST API определяет правила для создания, отправки и обработки запросов и ответов, делая процесс взаимодействия стандартизированным и надежным.

3.2 Реализация сервера

Любой код начинается с определения необходимых инструментов. Для реализации поставленной задачи был выбран язык программирования Go и фреймворк Gin для создания веб-приложений. Go (или Golang). Известен своей высокой производительностью, эффективной работой с параллельными процессами, что было освещено в предыдущей главе. Gin, в свою очередь, предоставляет удобные функции для маршрутизации и обработки HTTP-запросов, облегчая создание RESTful API.

Основным файлом сервера является «main.go». Является основной точкой входа, инициализирует сервер, подключается к базе данных и настраивает маршруты для обработки HTTP-запросов.

В основной файл входят:

– log. Стандартная библиотека Go для логирования. Используется для вывода логов и сообщений об ошибках;

– tracker/controllers. Пользовательский пакет, содержащий контроллеры, которые обрабатывают HTTP-запросы;

– tracker/database. Пользовательский пакет для инициализации подключения к базе данных PostgreSQL;

– github.com/gin-gonic/gin. Фреймворк для создания веб-приложений и API на Go. Предоставляет удобные функции для маршрутизации и обработки HTTP-запросов;

– gin.Default(). Создает новый экземпляр маршрутизатора Gin с настройками по умолчанию;

– database.InitDB(). Вызов функции для инициализации подключения к базе данных PostgreSQL. Эта функция возвращает подключение к базе данных и ошибку, если подключение не удалось;

– log.Fatalf. Выводит сообщение об ошибке и завершает программу, если подключение к базе данных не удалось;

– defer db.Close(). Гарантирует закрытие подключения к базе данных при завершении работы программы. Это важно для освобождения ресурсов и предотвращения утечек памяти;

– r.Use(gin.Logger()). Подключает мидлваре для логирования всех входящих HTTP-запросов. Это нужно для мониторинга и отладки приложения;

– r.Use(gin.Recovery()). Подключает мидлваре для восстановления, чтобы сервер не завершался при возникновении ошибок. Это помогает поддерживать стабильность приложения;

– r.GET("/tasks", controllers.GetTasks(db)). Настраивает маршрут для обработки GET-запросов на получение всех задач. Контроллер GetTasks принимает подключение к базе данных db и возвращает список задач;

– r.POST("/tasks", controllers.CreateTask(db)). Настраивает маршрут для обработки POST-запросов на создание новой задачи. Контроллер CreateTask принимает подключение к базе данных db и добавляет новую задачу в базу данных;

– r.PATCH("/tasks/:id", controllers.UpdateTask(db)). Настраивает маршрут для обработки PATCH-запросов на обновление существующей задачи по идентификатору id. Контроллер UpdateTask принимает подключение к базе данных db и обновляет информацию о задаче;

– r.DELETE("/tasks/:id", controllers.DeleteTask(db)). Настраивает маршрут для обработки DELETE-запросов на удаление задачи по идентификатору id. Контроллер DeleteTask принимает подключение к базе данных db и удаляет задачу;

– r.Run(":3000"). Запускает сервер на порту 3000. Если порт уже занят или произошла ошибка при запуске, программа выведет соответствующее сообщение об ошибке. Это позволяет серверу начать прослушивание входящих запросов и обрабатывать их. Так же можно указать произвольный порт.

Описанный код представлен на рисунке 6.

```go
package main

import (
    "log"                      // Пакет для логирования
    "tracker/controllers"      // Контроллеры для обработки запросов
    "tracker/database"         // Подключение к базе данных
    "github.com/gin-gonic/gin" // Фреймворк для веб-приложений
)

func main() {
    r := gin.Default() // Создание маршрутизатора Gin

    // Инициализация базы данных
    db, err := database.InitDB()
    if err != nil {
        log.Fatalf("Не удалось подключиться к базе данных: %v", err)
    }
    defer db.Close() // Закрытие подключения к базе данных при завершении работы программы

    // Подключение мидлваре
    r.Use(gin.Logger())
    r.Use(gin.Recovery())

    // Настройка маршрутов
    r.GET("/tasks", controllers.GetTasks(db))
    r.POST("/tasks", controllers.CreateTask(db))
    r.PATCH("/tasks/:id", controllers.UpdateTask(db))
    r.DELETE("/tasks/:id", controllers.DeleteTask(db))

    // Запуск сервера
    r.Run(":3000")
}
```

Рисунок 5 – Файл «main.go»

Следующим шагом нужно наладить взаимодействие базы данных и сервера. Для этого необходимо настроить подключение к базе данных PostgreSQL и предоставить функции для выполнения операций над данными. В проекте используется библиотека sqlx для работы с базой данных, которая

расширяет стандартный пакет database/sql и предоставляет дополнительные удобные функции.

В файл входят такие импортированные библиотеки как:

– log: Стандартная библиотека Go для логирования сообщений об ошибках;

– github.com/jmoiron/sqlx: Библиотека для работы с базами данных, которая расширяет стандартный пакет database/sql;

– github.com/lib/pq: Импорт драйвера PostgreSQL. Импортируется с префиксом _ для регистрации драйвера без явного использования.

А также фукция InitDB инициализирует подключение к базе данных PostgreSQL и возвращает объект подключения и ошибку, если подключение не удалось. Переменная dsn содержит строку подключения к базе данных, включающую имя пользователя, пароль, имя базы данных и режим SSL. sqlx.Connect("postgres", dsn) выполняет подключение к базе данных PostgreSQL. Если подключение не удалось, логируется сообщение об ошибке и программа завершается. Функция возвращает объект базы данных db и ошибку err.

Описанный код представлен на рисунке 7.

```go
package database

import (
    "log" // Пакет для логирования сообщений об ошибках
    "github.com/jmoiron/sqlx" // Библиотека для работы с базами данных
    _ "github.com/lib/pq" // Драйвер для работы с PostgreSQL
)

// InitDB инициализирует подключение к базе данных PostgreSQL
func InitDB() (*sqlx.DB, error) {
    // Строка подключения к базе данных
    dsn := "user=your_database_user password=your_database_password dbname=your_database_name sslmode=disable"
    // Подключение к базе данных
    db, err := sqlx.Connect("postgres", dsn)
    if err != nil {
        // Логирование ошибки и завершение программы при неудачном подключении
        log.Fatalln(err)
    }
    // Возвращение объекта базы данных и ошибки (если есть)
    return db, err
}
```

Рисунок 6 – Файл «database/database.go»

Структура файла Task представляет модель данных задачи и включает такие поля как:

– ID. Целочисленный идентификатор задачи, отмеченый для работы с базой данных (db:"id") и JSON (json:"id");

– title. Строка, представляющая заголовок задачи, отмеченый для работы с базой данных (db:"title") и JSON (json:"title");

– description. Строка, представляющая описание задачи, отмеченый для работы с базой данных (db:"description") и JSON (json:"description");

– status. Строка, представляющая статус задачи, отмеченый для работы с базой данных (db:"status") и JSON (json:"status").

Модель Task используется в контроллерах и репозиториях для обмена данными между различными слоями приложения. Она определяет структуру данных, которая сохраняется в базе данных и передается через API.

Описанный код представлен на рисунке 8.

```
type Task struct {
    ID          int    `db:"id" json:"id"`
    Title       string `db:"title" json:"title"`
    Description string `db:"description" json:"description"`
    Status      string `db:"status" json:"status"`
}
```

Рисунок 7 – Файл «Task»

Для взаимодействия с базой данных и управлением задачами из файла Task был реализован репозиторий, который отвечает за создание, чтение, обновление, удаление. В нем были реализованы функции:

– GetTasks. Возвращает все задачи из базы данных. Она выполняет SQL-запрос SELECT * FROM tasks и заполняет срез tasks объектами models.Task. Функция возвращает срез задач или ошибку, если она произошла;

– CreateTask. Создает новую задачу в базе данных. Она выполняет SQL-запрос INSERT INTO tasks (title, description, status) VALUES ($1, $2, $3)

RETURNING id, передавая значения заголовка, описания и статуса задачи. Идентификатор новой задачи считывается и присваивается полю ID структуры task. Функция возвращает обновленный объект task и ошибку, если она произошла;

– UpdateTask. Обновляет существующую задачу в базе данных. Она выполняет SQL-запрос UPDATE tasks SET title=$1, description=$2, status=$3 WHERE id=$4, передавая значения заголовка, описания и статуса задачи, а также идентификатор задачи для обновления. Функция возвращает обновленный объект task или ошибку, если она произошла;

– DeleteTask. Удаляет задачу из базы данных. Она выполняет SQL-запрос DELETE FROM tasks WHERE id=$1, передавая идентификатор задачи для удаления. Функция возвращает ошибку, если она произошла.

Описанный код представлен на рисунке 9.

```go
package repositories

import (
    "tracker/models"
    "github.com/jmoiron/sqlx"
)

// GetTasks возвращает все задачи из базы данных
func GetTasks(db *sqlx.DB) ([]models.Task, error) {
    tasks := []models.Task{}
    err := db.Select(&tasks, "SELECT * FROM tasks")
    return tasks, err
}

// CreateTask создает новую задачу в базе данных
func CreateTask(db *sqlx.DB, task models.Task) (models.Task, error) {
    var id int
    query := `INSERT INTO tasks (title, description, status) VALUES ($1, $2, $3) RETURNING id`
    err := db.QueryRow(query, task.Title, task.Description, task.Status).Scan(&id)
    task.ID = id
    return task, err
}

// UpdateTask обновляет существующую задачу в базе данных
func UpdateTask(db *sqlx.DB, id int, task models.Task) (models.Task, error) {
    query := `UPDATE tasks SET title=$1, description=$2, status=$3 WHERE id=$4`
    _, err := db.Exec(query, task.Title, task.Description, task.Status, id)
    task.ID = id
    return task, err
}

// DeleteTask удаляет задачу из базы данных
func DeleteTask(db *sqlx.DB, id int) error {
    query := `DELETE FROM tasks WHERE id=$1`
    _, err := db.Exec(query, id)
    return err
}
```

Рисунок 8 – Файл «taskRepository.go»

3.3 Выводы

В этой главе были определены цели и задачи, которые необходимо выполнить при реализации выпускной квалификационной работы. Сравнение существующих средств разработки мобильных приложений помогло определить наиболее подходящие инструменты и технологии для создания мобильного приложения в соответствии с требованиями и целями работы. Также разработан концепт интерфейса и описаны варианты использования приложения.

ЗАКЛЮЧЕНИЕ

В первой главе проанализированы существующие аналоги и составлены функциональные требования приложения.

Во второй главе были выбраны основные инструменты для реализации сервера, а также описана архитектура приложения и взаимодействие клиента с сервером.

В третьей главе были представлены API и реализация ключевых файлов серверной части приложения.

В результате выполненной работы был разработан сервер и база данных для взаимодействия с клиентской частью приложения. Также разработан концепт интерфейса и описаны варианты использования приложения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Как определять и анализировать целевую аудиторию // inSales: сайт. URL: https://www.insales.ru/blogs/university/celevaja-auditorija (дата обращения 25.05.2024)

2. ТОП-40 лучших программ тайм-трекеров в 2024 году: управление персоналом и учет рабочего времени // APPTASK управление проектами: сайт. URL: https://apptask.ru/blog/chto-takoe-time-treker-i-kak-on-rabotaet (дата обращения 27.05.2024)

3. Проектирование // Wikipedia: Официальный сайт. URL: https://en.wikipedia.org/wiki/Project (дата обращения 17.02.2024)

4. Разновидности диаграмм UML // lucidchart: сайт. URL: https://www.lucidchart.com/blog/ru/types-of-UML-diagrams (дата обращения 22. 04.2024)

5. Введение в UML // ИНТУИТ национальный открытый университет: сайт. URL: https://intuit.ru/studies/courses/1007/229/lecture/5962 (дата обращения 23.05.2024)

6. PostgreSQL // Skillfactory: Официальный сайт. URL: https://blog.skillfactory.ru/glossary/postgresql/ (дата обращения)

7. Основы проектирования баз данных // METANIT.COM: сайт. URL: https://metanit.com/sql/tutorial/1.1.php (дата обращения 18.05.2024)

8. Основные особенности и виды архитектур клиент-сервер // Ittelo: сайт. URL: https://www.ittelo.ru/news/osnovnye-osobennosti-i-vidy-arkhitektur-klient-server/#tag-9 (дата обращения 02.06.2024)

9. Клиент-серверное взаимодействие // Ittelo: сайт. URL: https://www.ittelo.ru/news/klient-servernoe-vzaimodeystvie/ (дата обращения 02.06.2024)

10. СТУ 7.5–07–2021. Система менеджмента качества. Общие требования к построению, изложению и оформлению документов учебной деятельности : стандарт университета : издание официальное : утверждён и

введён в действие приказом ректора Сибирского федерального университета М.В. Румянцевым от 07 декабря 2021г. №1301 : введён взамен СТО 4.2–07–2014 : дата введения 2021-12-20 / разработан СФУ – Красноярск : СФУ, 2021 – 61с.

11. REST API: что это такое и как работает // Skillbox: сайт. URL: https://skillbox.ru/media/code/rest-api-chto-eto-takoe-i-kak-rabotaet/ (дата обращения 03.06.2024)

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
«20» 06 2024 г.

БАКАЛАВРСКАЯ РАБОТА

090301 Информатика и вычислительная техника

Мобильное приложение – трекер дня. Серверная часть.

Руководитель _____ 20.06.24 доцент, канд. тех. наук С.Н. Титовский
подпись дата должность, ученая степень

Выпускник _____ 20.06.24 А.В. Чередниченко
подпись дата

Нормоконтролёр _____ 20.06.24 доцент, канд. тех. наук С.Н. Титовский
подпись дата должность, ученая степень

Красноярск 2024