СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Система бронирования автомобилей – это удобный сервис, который позволяет пользователям арендовать автомобили на определенный срок без необходимости покупки. Благодаря этой системе, клиенты могут выбирать из широкого ассортимента автомобилей различных марок и моделей, а также настраивать условия аренды в соответствии с их потребностями. Система бронирования автомобилей обеспечивает удобство и гибкость в использовании транспорта, что делает ее популярным выбором как для путешествий, так и для повседневных нужд.

Данный проект представляет собой техническое задание на разработку портала бронирования автомобилей, функциональность которого включает в себя возможность поиска и выбора подходящего автомобиля по различным критериям (бренд, модель, мощность двигателя, стоимость и тип автомобиля), возможность выбора периода и оплаты бронирования, его завершения и отмены, а также оставление отзывов на автомобили по завершению аренды (с возможностью их просмотра для будущих арендаторов).

Целью данного курсового проекта является разработка веб-приложения аренды автомобилей.

Для достижения цели необходимо решить следующие задачи:

- провести анализ предметной области;
- спроектировать архитектуру распределенной системы;
- произвести выбор стека технологий для реализации;
- реализовать распределенную систему аренды автомобилей.

1 Аналитический раздел

1.1 Существующие аналоги

На данный момент существует некоторое количество сервисов, нацеленных на аренды автомобилей, но они имеют ряд недостатков. Данные о сравнении сервисов сведены в таблице 1.

Таблица 1.1 – Обзор существующих аналогов

Название	**Архитектура**	**Наличие отзывов**
rentcars.ru	Микросервисная	Нет
prostoprokat.ru	Монолитная	Нет
avtomaxi.ru	Монолитная	Есть

Не все представленные сервисы имеют возможность оставлять отзывы на арендуемые автомобили, из-за чего арендатор не может узнать об актуальном состянии арендуемого автомобиля.

Также не все представленные сервисы основаны на микросервисной архитектуре, из-за чего возникают сложности с масштабированием, обслуживанием и внесением изменений в функциональность.

1.2 Описание системы

Разрабатываемый сервис должен представлять собой распределенную систему для бронирования автомобилей. Если пользователь хочет забронировать выбранный автомобиль, ему необходимо зарегистрироваться, указав информацию: фамилия, имя, отчество, номер телефона, электронная почта, пароль. После регистрации для активации аккаунта необходимо подтвердить почту путем перехода по ссылке, присланной на почту. В случае, если зарегистрированному ранее пользователю нужно получить информацию о свободных для бронирования автомобилей и выбрать один из них, ему нужно авторизоваться. Для неавторизованных пользователей доступен только просмотр списка свободных для бронирования автомобилей.

На рисунке 1.1 отображена схема предметной области.

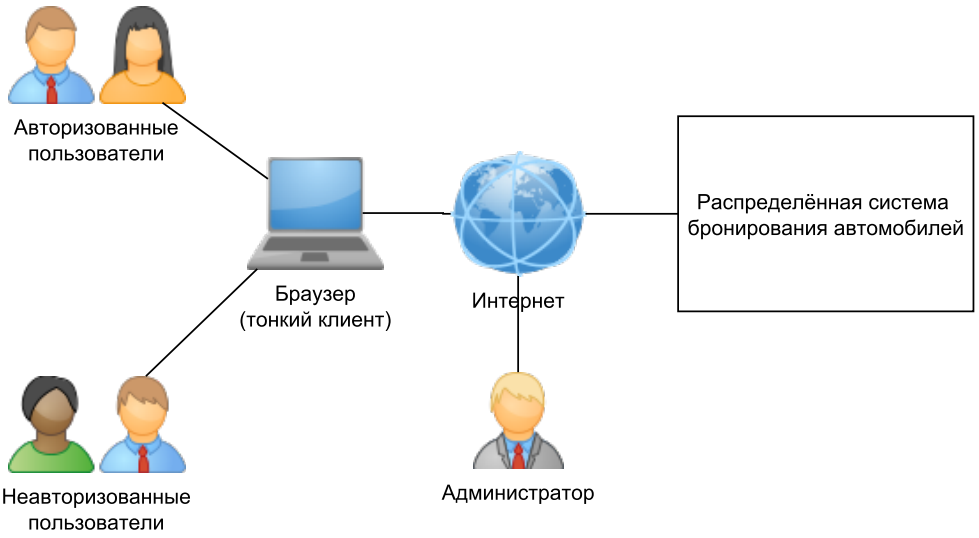

Рисунок 1.1 – Схема предметной области.

1.3 Общие требования к системе

Требования к системе следующие.

1. Разрабатываемое ПО должно поддерживать функционирование системы в режиме 24 часов, 7 дней в неделю, 365 дней в году (24/7/365) со среднегодовым временем доступности не менее 99.9%. Допустимое время, в течении которого система недоступна, за год должна составлять $24 \cdot 365 \cdot 0.001 = 8.76$ ч.

2. Время восстановления системы после сбоя не должно превышать 15 минут.

3. Каждый узел должен автоматически восстанавливаться после сбоя.

4. Система должна поддерживать возможность «горячего» переконфигурирования системы. Необходимо предусмотреть поддержку добавления нового узла во время работы системы без рестарта.

5. Обеспечить безопасность работоспособности за счет отказоустойчивости узлов.

1.4 Требования к функциональным характеристикам

1. По результатам работы модуля сбора статистики медиана времени отклика системы на запросы пользователя на получение информации не

должна превышать 3 секунд.

2. По результатам работы модуля сбора статистики медиана времени отклика системы на запросы, добавляющие или изменяющие информацию на портале не должна превышать 7 секунд.

3. Система должна обеспечивать возможность запуска в современных браузерах: не менее 85% пользователей Интернета должны пользоваться ей без какой-либо деградации функционала.

1.5 Функциональные требования к системе с точки зрения пользователя

Система должна обеспечивать реализацию следующих функций.

1. Регистрация и авторизация пользователей с валидацией вводимых данных (процесс регистрации включает в себя подтверждение адреса электронной почты).

2. Аутентификация пользователей (в том числе двойная, на усмотрение пользователя).

3. Разделение всех пользователей на три роли:

   - Неавторизированный пользователь (гость));
   - Авторизированный пользователь (клиент));
   - Администратор.

4. Предоставление возможностей **Гостю, Клиенту, Администратору** представленных в таблице 1.2.

Таблица 1.2 – Функции пользователей

Гость	1. просмотр списка доступных автомобилей; 2. получение информации о выбранном автомобиле; 3. просмотр отзывов на выбранный автомобиль; 4. регистрация в системе; 5. авторизация в системе;
Клиент	1. функции гостя; 2. создание отзыва на ранее забронированный автомобиль; 3. просмотр собственной истории бронирования автомобилей; 4. бронирование выбранного автомобиля на указанный период; 5. оплата аренды автомобиля; 6. завершение и отмена аренды автомобиля.
Администратор	1. функции клиента; 2. просмотр информации об арендах автомобилей указанного пользователя; 3. создание, редактирование и удалении информации о бронируемых автомобилях; 4. редактирование данных и удаление пользователей; 5. отслеживание транзакций.

1.6 Входные данные

Входные параметры системы представлены в таблице 1.3.

Таблица 1.3 – Входные данные

Сущность	**Входные данные**
Клиент / Администратор	1. *фамилия, имя* и *отчество* не более 256 символов каждое поле; 2. *дата рождения* в формате д/м/гггг; 3. *логин* не менее 10 символов и не более 128; 4. *пароль* не менее 8 символов и не более 128, как минимум одна заглавная и одна строчная буква, только латинские буквы, без пробелов, как минимум одна цифра; 5. *номер телефона*; 6. *электронная почта*; 7. *роль* (CLIENT, ADMIN);
Автомобиль	1. *бренд* не более 80 символов; 2. *модель* не более 80 символов; 3. *регистрационный номер* не более 20 символов; 4. *мощность*; 5. *цена*; 6. *тип* (SEDAN, SUV, MINIVAN, ROADSTER); 7. *наличие*;
Аренда	1. *имя пользователя*; 2. *дата начала аренды*; 3. *дата конца аренды*; 4. *статус* (IN_PROGRESS, FINISHED, CANCELED);
Платеж	1. *статус* (PAID, CANCELED); 2. *цена*;
Отзыв	1. *имя пользователя*; 2. *оценка*; 3. *дата публикации* ; 4. *отзыв* не более 500 символов.

1.7 Выходные параметры

Выходными параметрами системы являются web-страницы. В зависимости от запроса и текущей роли пользователя они содержат следующую

информацию (таблица 1.4).

Таблица 1.4 – Выходные параметры

Гость	1. Список доступных автомобилей, указывается: • *бренд*; • *модель*; • *регион*; • *цена*; • *тип*; • *рейтинг*;
	2. Подробное описание выбранного автомобиля: • *бренд*; • *модель*; • *регистрационный номер*; • *мощность*; • *цена*; • *тип*; • *рейтинг*;
	3. Отзывы выбранного автомобиля: • *имя пользователя*; • *оценка*; • *дата*; • *отзыв*;
	4. Общая информация о сайте;
	5. Контактная информация поддержки;
Клиент	1. Список доступных автомобилей, указывается: • *бренд*; • *модель*; • *регион*; • *цена*; • *тип*; • *рейтинг*;

	2. Подробное описание выбранного автомобиля: • *бренд*; • *модель*; • *регистрационный номер*; • *мощность*; • *цена*; • *тип*; • *рейтинг*;
	3. Параметры аренды выбранного автомобиля: • *дата начала аренды*; • *дата конца аренды*; • *суммарная стоимость аренды*;
	4. Отзывы выбранного автомобиля: • *имя пользователя*; • *оценка*; • *дата*; • *отзыв*;
	5. Общая информация о сайте;
	6. Контактная информация поддержки;
	7. История бронирования автомобилей: • *дата начала аренды*; • *дата конца аренды*; • *суммарная стоимость аренды*; • *статус аренды*;
Администратор	1. Список автомобилей, указывается: • *бренд*; • *модель*; • *регион*; • *цена*; • *тип*; • *рейтинг*; • *наличие*;

	2. Подробное описание выбранного автомобиля: • *бренд*; • *модель*; • *регистрационный номер*; • *мощность*; • *цена*; • *тип*; • *рейтинг*;
	3. Отзывы выбранного автомобиля: • *имя пользователя*; • *оценка*; • *дата*; • *отзыв*;
	4. Общая информация о сайте;
	5. Контактная информация поддержки;
	6. Список зарегистрированных пользователей: • *фамилия, имя* и *отчество*; • *дата рождения*; • *номер телефона*; • *электронная почта*;
	7. История бронирования автомобилей указанного пользователя: • *дата начала аренды*; • *дата конца аренды*; • *суммарная стоимость аренды*; • *статус аренды*;
	8. Список всех пользователей с возможностью их удаления или ограничения доступа на время, а также возможность редактирования информации о пользователе в соответствии с полями из пункта 6;
	9. Список всех автомобилей для бронирования с возможностью их удаления, а также возможность редактирования информации об автомобиле в соответствии с полями из пункта 2;
	10. Статистика по порталу, собранная через сервис статистики.

1.8 Топология Системы

На рисунке 1.2 изображен один из возможных вариантов топологии разрабатываемой распределенной Системы.

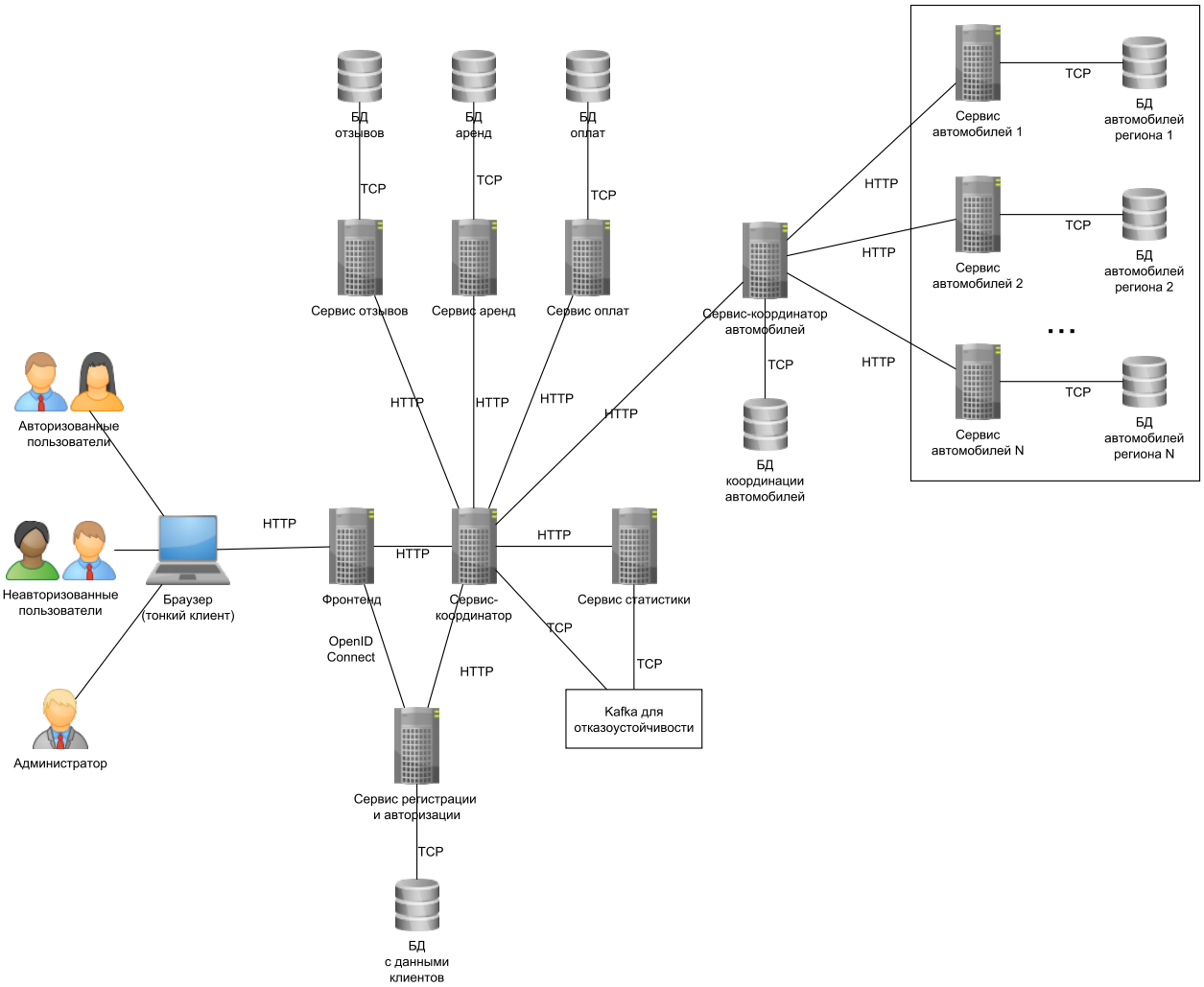

Рисунок 1.2 – Топология системы.

Система будет состоять из фронтенда и 8 подсистем:

- сервис-координатор;
- сервис регистрации и авторизации;
- сервис-координатор автомобилей;
- сервис автомобилей;
- сервис аренд;

- сервис оплат;
- сервис отзывов;
- сервис статистики.

Фротенд принимает запросы от пользователей по протоколу HTTP и анализирует их. На основе проведенного анализа выполняет запросы к микросервисам бекенда, агрегирует ответы и отсылает их пользователю.
Сервис-координатор – сервис, ответственный за координацию запросов внутри системы. Для реализации балансировки запросов используется инфраструктура Kubernetes.
Сервис-регистрации и авторизации отвечает за:

- возможность регистрации нового клиента;
- аутентификацию пользователя (клиента и администратора);
- авторизацию пользователя;
- двухфазную авторизацию;
- подтверждение почты;
- восстановление пароля к странице;
- выход из сессии.

Сервис регистрации и авторазации в своей работе использует базу данных, которая хранит следующую инфомацию:

- Пользователь
  - *идентификатор*;
  - *фамилия, имя* и *отчество*;
  - *дата рождения*;
  - *логин*;
  - *пароль*;
  - *номер телефона*;

- *электронная почта*;
- *роль*.

Сервис-координатор автомобилей – сервис, ответственный за координацию запросов к соответствующим сервисам автомобилей, каждый из которых привязан к своему региону по регистрационному номеру. То есть, если, например, для поиска был выбрана московская область, то сервис определяет код региона для этого субъекта и перенаправляет запрос к тому сервису, ответственному за этот регион.

Сервис использует в своей работе базу данных, в которой хранится следующая информация:

- Адрес сервиса:
  - *идентификатор*;
  - *название региона*
  - *код региона*;
  - *ip-адрес*.

Сервис автомобилей реализует функции:

- получение списка автомобилей, как доступных для бронирования, так и занятых;
- получение информации об автомобиле;
- создание записи о новом автомобиле;
- редактирование информации об автомобиле;
- удаление автомобиля.

Сервис автомобилей в своей работе использует базу данных, которая хранит следующую инфомацию:

- Автомобиль
  - *идентификатор*;

  - *бренд*;
  - *модель*;
  - *регистрационный номер*;
  - *мощность*;
  - *цена*;
  - *тип*;
  - *наличие*.

Сервис аренд реализует функции:

- получение списка аренд указанного пользователя;
- получение информации по конкретной аренде пользователя;
- создание записи о бронировании автомобиля;
- редактирование записи о бронировании автомобиля;
- завершение аренды автомобиля;
- отмена аренды автомобиля;
- удаление записи о бронировании автомобиля.

Сервис аренд в своей работе использует базу данных, которая хранит следующую инфомацию:

- Аренда
  - *идентификатор*;
  - *имя пользователя*;
  - *идентификатор* соответствующей *оплаты*;
  - *идентификатор* соответствующего *автомобиля*;
  - *дата начала аренды*;
  - *дата конца аренды*;
  - *статус*.

Сервис оплат реализует функции:

- получение списка оплат;
- получение информации о конкретной оплате;
- создание записи о новой оплате;
- отмена платежа;
- редактирование информации об оплате;
- удаление оплаты.

Сервис оплат в своей работе использует базу данных, которая хранит следующую инфомацию:

- Оплата
  - *идентификатор*;
  - *статус*;
  - *цена*.

Сервис отзывов реализует функции:

- получение списка всех отзывов;
- получение списка личных отзывов пользователя;
- получение отзывов о конкретном автомобиле;
- изменение отзыва;
- удаление отзыва;
- получение выбранного отзыва.

Сервис отзывов в своей работе использует базу данных, которая хранит следующую инфомацию:

- Отзыв

- *идентификатор*;
- *идентификатор* соответствующего *автомобиля*;
- *идентификатор* соответствующего *пользователя*;
- *числовая оценка* от 1 до 5;
- *дата публикации*;
- *отзыв*.

Сервис статистики отвечает за логирование событий во всей системе для осуществления возможности быстрого детектирования, локализации и воспроизведения ошибки в случае её возникновения.

1.9 Требования к программной реализации

1. Требуется использовать архитектуру SPA (Single Page Application) для реализации системы. Использование CSS обязательно.
2. Требуется создать сервис, выполняющий функцию Identity Provider [1], реализовать протокол OpenID Connect.
3. Система состоит из микросервисов. Каждый микросервис отвечает за свою область логики работы приложения и должны быть запущены изолированно друг от друга (один сервис – один docker-контейнер [2]).
4. При необходимости, каждый сервис имеет своё собственное хранилище, запросы между базами запрещены.
5. Каждый сервис при получении запроса выполняет валидацию JWT токена с помощью JWKs, которые он получает из Identity Provider.
6. При разработке базы данных необходимо учитывать, что доступ к ней должен осуществляться по протоколу TCP.
7. Для межсервисного взаимодействия использовать HTTP (придерживаться RESTful).
8. Выделить Gateway Service как единую точку входа и межсервисной коммуникации для исключения горизонтальных запросов между сервисами.

9. При недоступности систем портала должна осуществляться деградация функционала или выдача пользователю сообщения об ошибке.

10. Необходимо предусмотреть авторизацию пользователей, как через интерфейс приложения, так и через приложения двухфазной авторизации.

11. Валидация входных данных должна производиться и на стороне пользователя с помощью TypeScript скриптов, и на стороне фронтенда. Бекенды не должны валидировать входные данные, так как пользователь не может к ним обращаться напрямую, бекенды должны получать уже отфильтрованные входные данные от фронтенда.

12. Для запросов, выполняющих обновление данных на нескольких узлах распределенной системы, в случае недоступности одной из систем, необходимо выполнять полный откат транзакции.

13. На сервисе-координаторе для всех операций чтения реализовать паттерн Circuit Breaker. Накапливать статистику в памяти, и если система не ответила N раз, то в $N + 1$ раз вместо запроса сразу отдавать fallback. Через небольшой timeout выполнить запрос к реальной системе, чтобы проверить ее состояние.

14. В случае недоступности данных из некритичного источника (не основного), возвращается fallback-ответ. В зависимости от ситуации, это может быть:

    - пустой объект или массив;
    - объект, с заполненным полем (uid или подобным), по которому идет связь с другой системой;
    - строка по умолчанию (если при этом не меняется тип переменной).

15. Код хранить на Github, для сборки использовать Github Actions.

1.10 Функциональные требования к подсистемам

Фронтенд – серверное приложение, предоставляет пользовательский интерфейс и внешний API системы, при разработке которого нужно учитывать следующее:

- должен принимать запросы по протоколу HTTP и формировать ответы пользователям в формате HTML;
- в зависимости от типа запроса должен отправлять последовательные запросы в соответствующие микросервисы;
- запросы к микросервисам необходимо осуществлять по протоколу HTTP;
- данные необходимо передавать в формате JSON.

Сервис-координатор – серверное приложение, через которое проходит весь поток запросов и ответов, должен соответствовать следующим требованиям разработки:

- обрабатывать запросы в соответствии со своим назначением, описанным в топологии системы;
- принимать и возвращать данные в формате JSON по протоколу HTTP;
- накапливать статистику запросов, в случае, если система не ответила N раз, то в $N + 1$ раз вместо запроса сразу отдавать fallback. Через некоторое время выполнить запрос к реальной системе, чтобы проверить ее состояние;
- выполнять проверку существования клиента, также регистрацию и аутентификацию пользователей;
- при получении запроса выполнять валидацию JWT токена с помощью JWKs, которые он получает из Identity Provider;
- осуществлять деградацию функциональности в случае отказа некритического сервиса (зависит от семантики запроса);
- существовать в нескольких экземплярах, чтобы координация запросов не была узким местом приложения;
- уведомлять сервис статистики о событиях в системем.

Сервис регистрации и авторизации, сервис-координатор автомобилей, сервис автомобилей, сервис аренд, оплат и сервис отзывов

– это серверные приложения, которые должны отвечать следующим требованиям по разработке:

- обрабатывать запросы в соответствии со своим назначением, описанным в топологии системы;
- принимать и возвращать данные в формате JSON по протоколу HTTP;
- осуществлять доступ к СУБД по протоколу TCP.

Сервис статистики – это серверное приложение, которое должно отвечать следующим требованиям по разработке:

- обрабатывать запросы в соответствии со своим назначением, описанным в топологии системы;
- принимать и возвращать данные в формате JSON по протоколу HTTP.

2 Конструкторский раздел

2.1 Концептуальный дизайн

Для создания функциональной модели портала, отражающей его основные функции и потоки информации наиболее наглядно использовать нотацию IDEF0. На рисунке 2.1 приведена концептуальная модель системы. На рисунке 2.2 представлена декомпозиция функциональной модели системы.

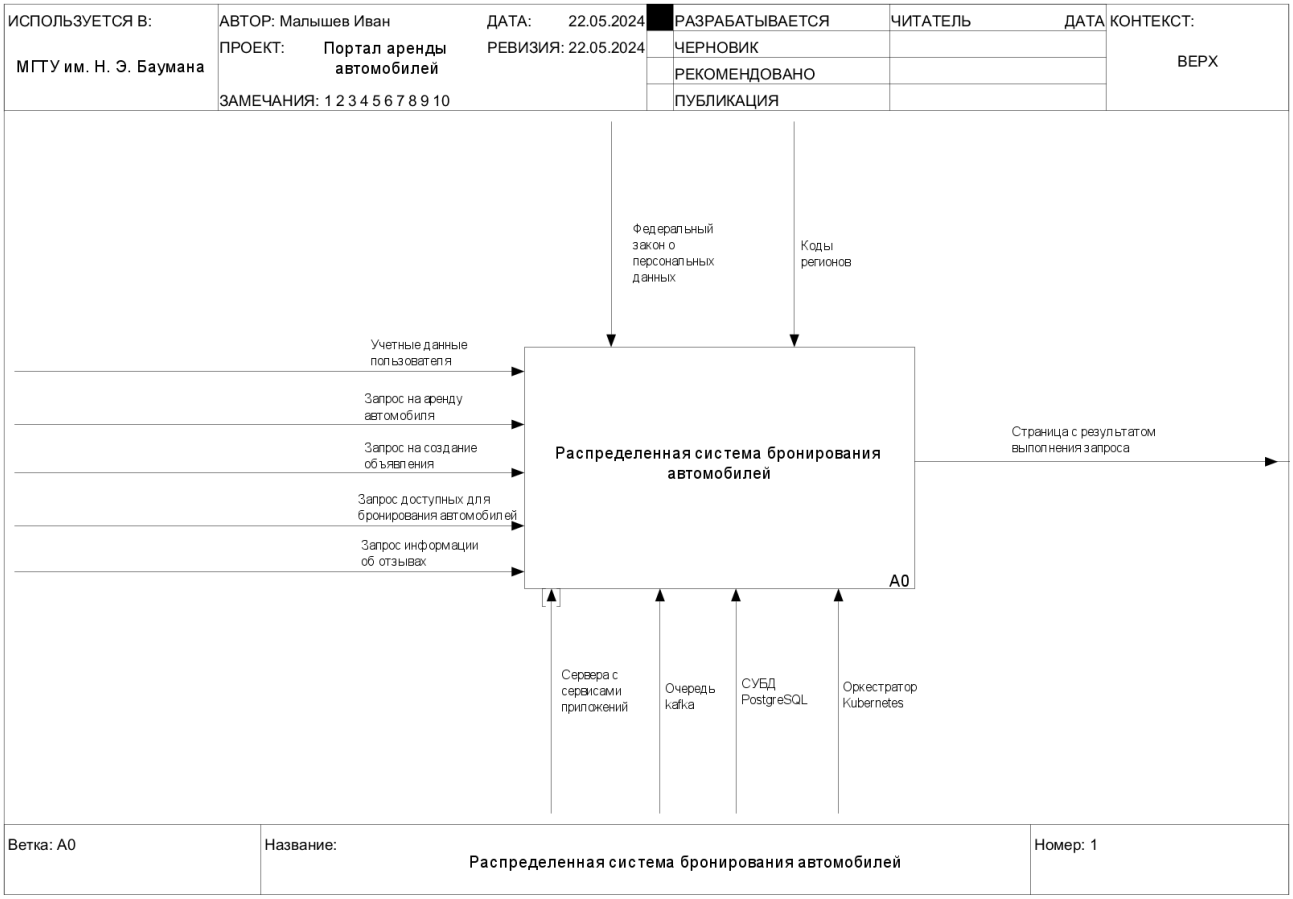

Рисунок 2.1 – Концептуальная модуль системы в нотации IDEF0

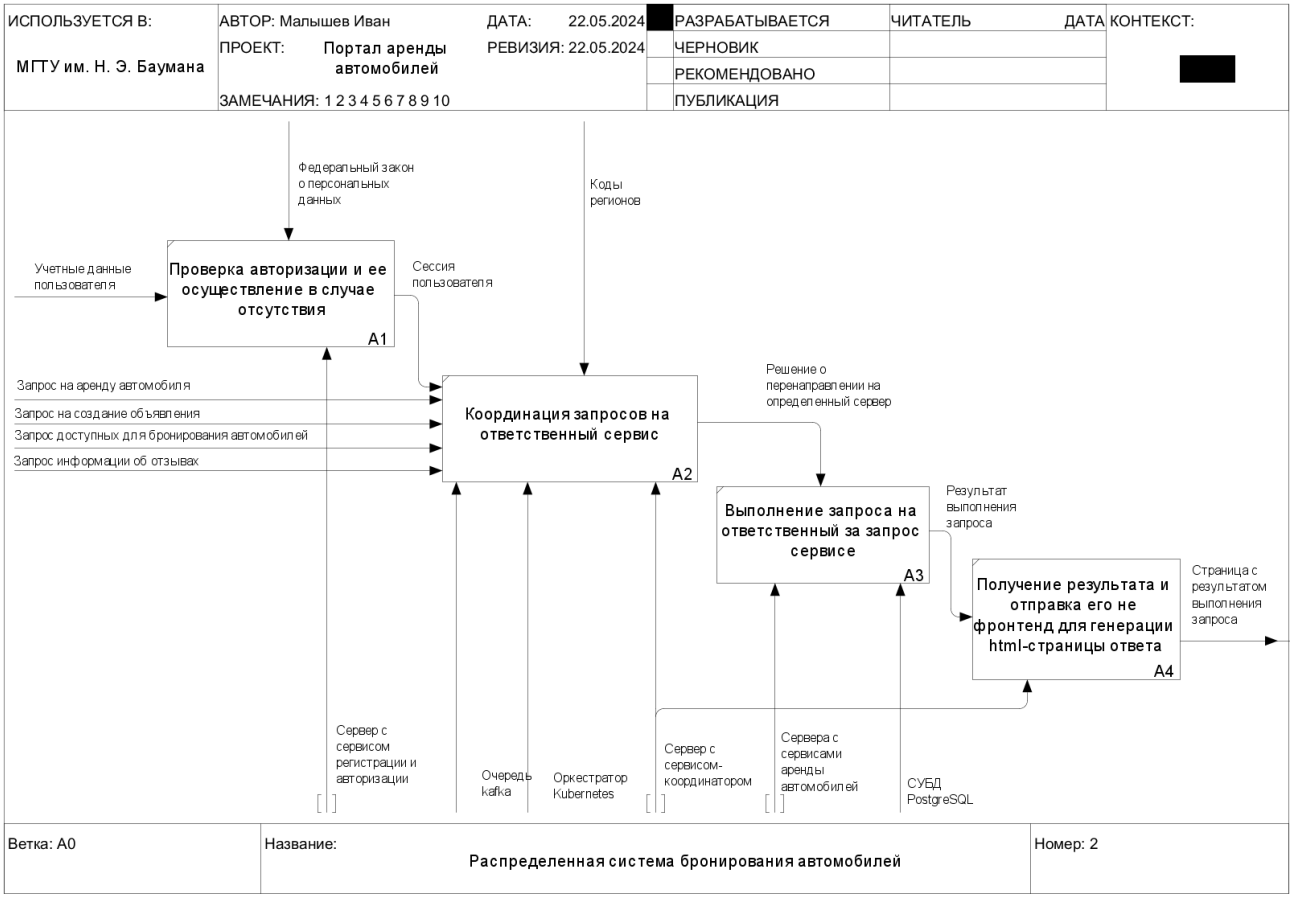

Рисунок 2.2 – Детализированная концептуальная модель системы в нотации IDEF0

2.2 Сценарии функционирования системы

Регистрация пользователя

1. Пользователь переходит на страницу регистрации с помощью кнопки «Регистрация», либо автоматически перенаправляется на страницу авторизации, где ему будет предложено перейти к регистрации, при попытке совершения действий, которые невозможно совершить без регистрации (например, забронировать автомобиль). После чего пользователь будет перенаправлен на страницу регистрации.

2. Пользователь заполняет различные наборы полей на странице регистрации. Валидация входных данных осуществляется «на лету» на стороне пользователя. При отправке данных на фронтенд, он тоже производит валидацию.

3. Пользователь нажимает кнопку «Регистрация», после чего пользователь

перенаправляется на главную страницу портала.

Авторизация на портале пользователя

1. Пользователь переходит на страницу авторизации с помощью кнопки «Авторизация», либо автоматически перенаправляется на соответствующую страницу при попытке совершения действий, которые невозможно совершить без регистрации (например, забронировать автомобиль).

2. Вводит учётные данные, нажимает кнопку «Войти».

3. Пользователь даёт согласие на использование его данных. Если пользователь не дает согласия, то он перенаправляется на страницу с ошибкой.

4. Пользователь перенаправляется на главную страницу портала.

Просмотр доступных для бронирования автомобилей

Сценарий доступен как для авторизованного (Клиент), так и для неавторизованного (Гость) пользователя.

1. При переходе на страницу «Аренда авто» пользователю на экране предоставляется список доступных для аренды автомобилей.

2. Пользователь задает конкретезированные и диапазонные параметры поиска (бренд, регион, цена, тип) и после нажатия кнопки «Обновить» получает список доступных для аренды автомобилей в соответствии с заданными им фильтрами просмотра.

Аренда автомобиля

1. При переходе на страницу «Аренда авто» пользователю на экране предоставляется список доступных для аренды автомобилей.

2. Пользователь задает конкретезированные и диапазонные параметры поиска (бренд, регион, мощность, цена, тип) и после нажатия кнопки «Обновить» получает список доступных для аренды автомобилей в соответствии с заданными им фильтрами просмотра.

3. Пользователь выбирает автомобиль и переходит на страницу с детальной информацией об автомобиле.

4. Нажав кнопку «Арендовать», пользователь должен выбрать срок аренды (дата начала и окончания аренды), после чего пользователю предоставляется информация о сумме оплаты за аренду.

5. Пользователь нажимает кнопку «Оплатить», после чего происходит операция оплаты (денежная транзакция).

6. В случае успешной оплаты, пользователь арендует автомобиль, т. е. помечает автомобиль как занятый, а информация об аренде попадает в историю аренд пользователя.

7. В случае завершения аренды, пользователь оставляет оценку автомобилю и опционально заполняет отзыв об автомобиле. В случае отмены аренды, необходимо указать причину.

Создание объявления об аренде автомобиля

Сценарий доступен только пользователю с ролью «Администратор».

1. При переходе на страницу «Аренда авто» пользователю на экране предоставляется список доступных для аренды автомобилей.

2. Пользователь переходит на страницу создания объявления с помощью кнопки «Создать новое объявление».

3. Пользователь вводит данные о питомце (бренд, модель, регистрационный номер, мощность, цена, тип) в соответствующие поля, которые валидируются «на лету» на стороне пользователя.

4. Пользователь переходит на страницу для прикрепления фотографий автомобиля с помощью кнопки «Далее».

5. Пользователь прикрепляет фотографии автомобиля в соответствии с требованиями к формату. После чего происходит проверка регистрационных номеров в ГИБДД.

6. Пользователь перенаправляется на страницу созданного им объявления, где помимо информации по объявлению отображается статус ожидания проверки регистрационных номеров и место в очереди на проверку. После успешной проверки объявление публикуется для общего доступа.

Получение статистики

Сценарий доступен только пользователю с ролью «Администратор».

1. Пользователь нажимает на кнопку «Посмотреть историю запросов» и перенаправляется на соответствующую страницу.
2. Пользователь нажимает на кнопку «Получить статистику».
3. Пользователь перенаправляется на страницу просмотра статистики о запросах.

2.3 Диаграммы прецедентов

Для детальной разработки портала используется унифицированный язык моделирования UML. В системе выделены три роли: Гость, Клиент, Администратор. На рисунках 2.3-2.5 представлены диаграммы прецедентов для каждой из ролей.

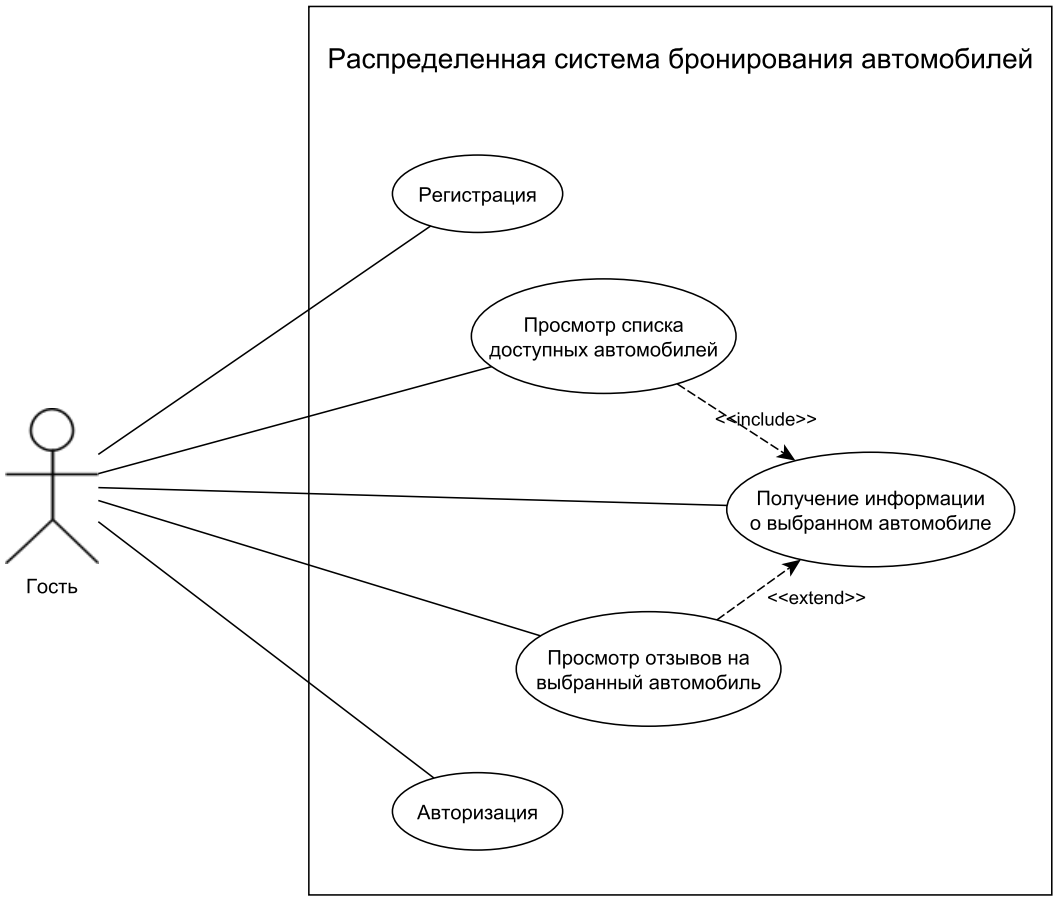

Рисунок 2.3 – Диаграмма с точки зрения Гостя.

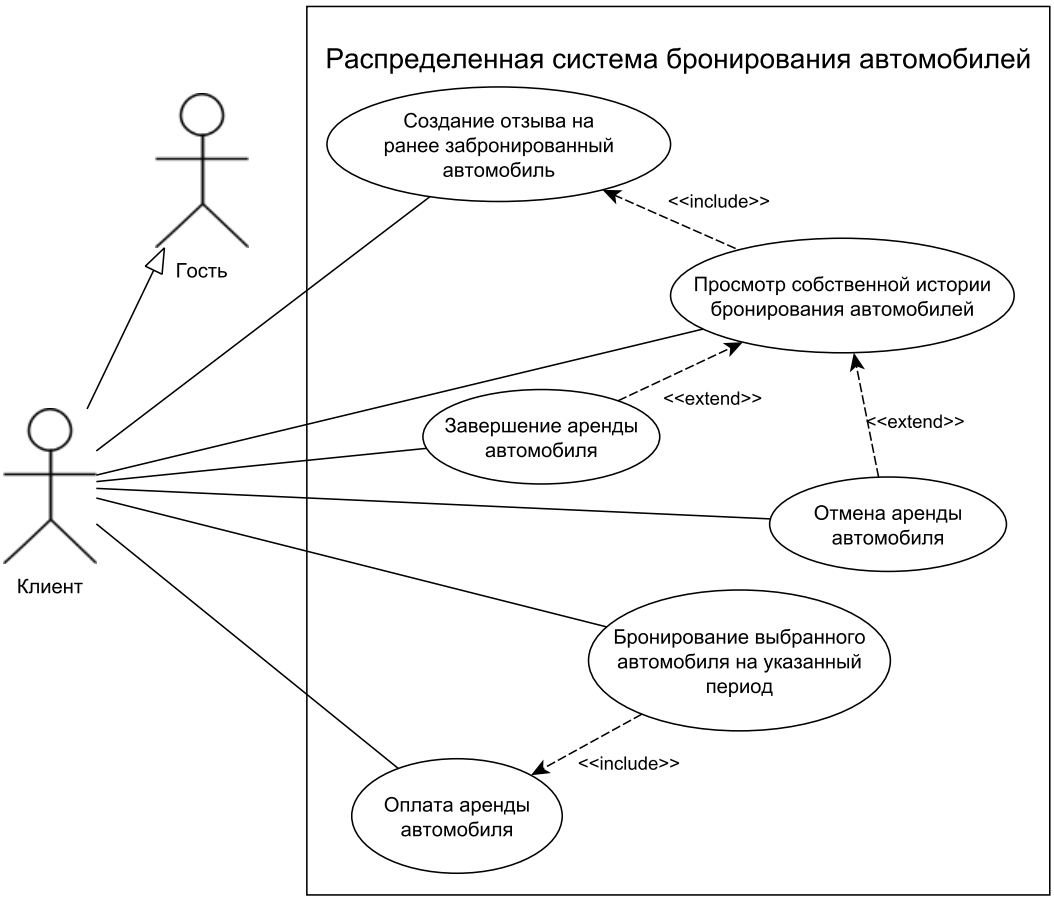

Рисунок 2.4 – Диаграмма с точки зрения Клиента.

Рисунок 2.5 – Диаграмма с точки зрения Администратора.

2.4 Спецификация классов

Иерархии классов для разработки серверных приложений представлены в виде диаграммы классов:

- сервис автомобилей (Рисунок 2.6);
- сервис аренд (Рисунок 2.7);
- сервис оплат (Рисунок 2.8);
- сервис-координатор (Рисунок 2.9).

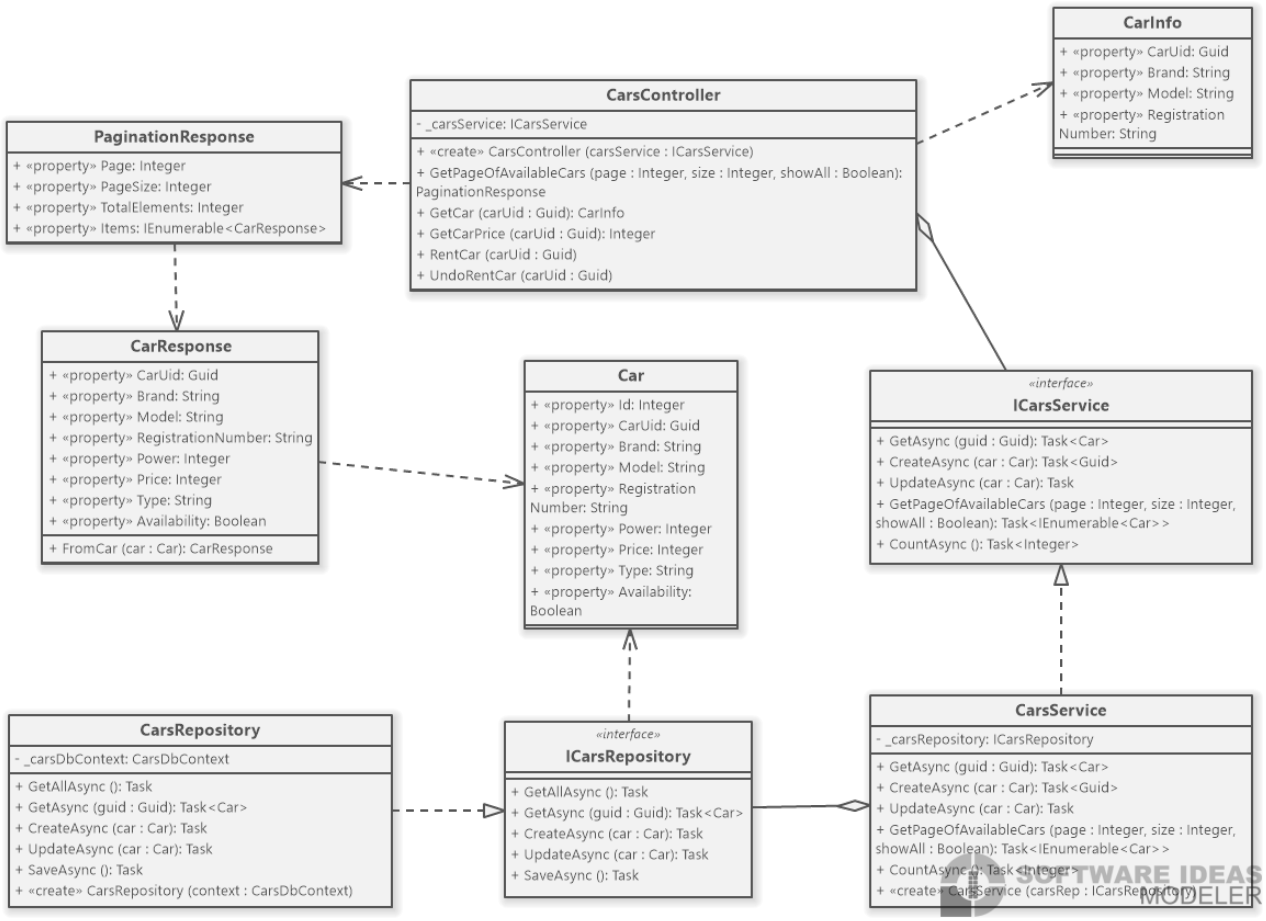

Рисунок 2.6 – Диаграмма классов сервиса автомобилей

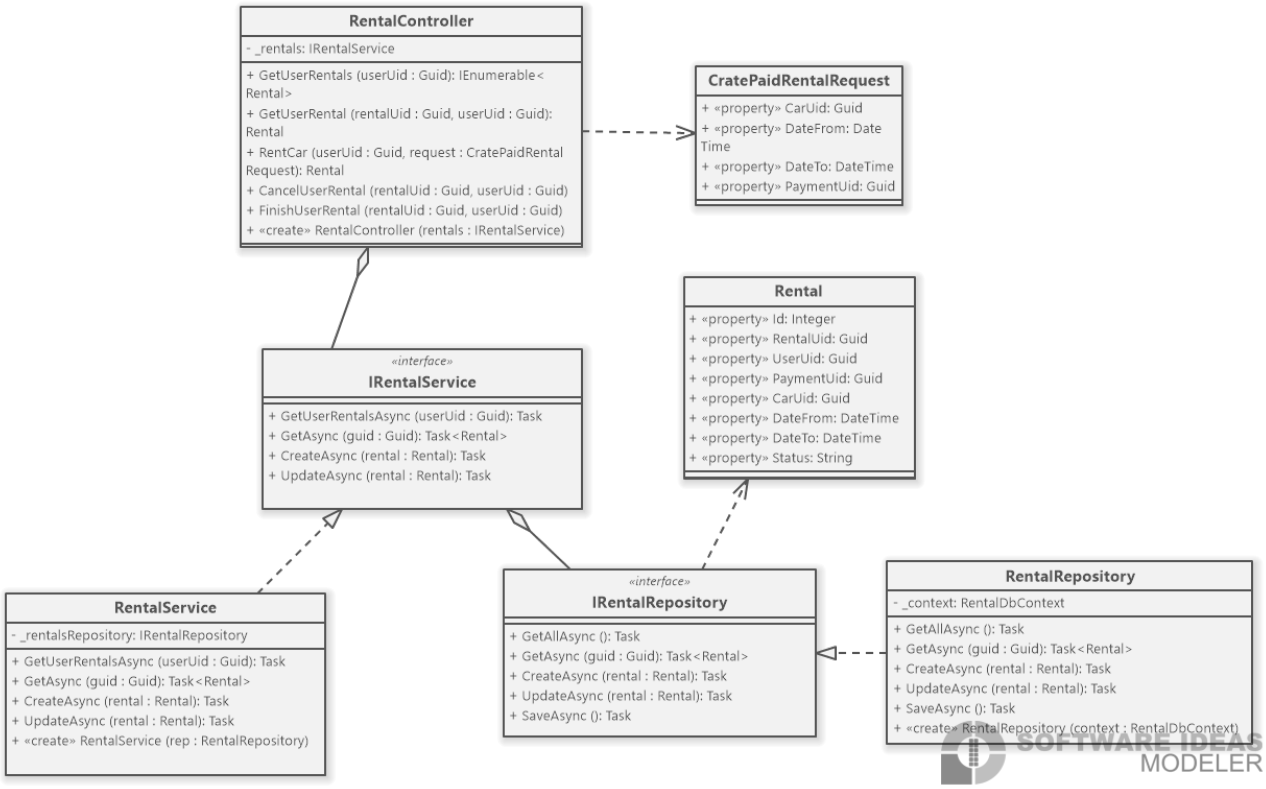

Рисунок 2.7 – Диаграмма классов сервиса аренд

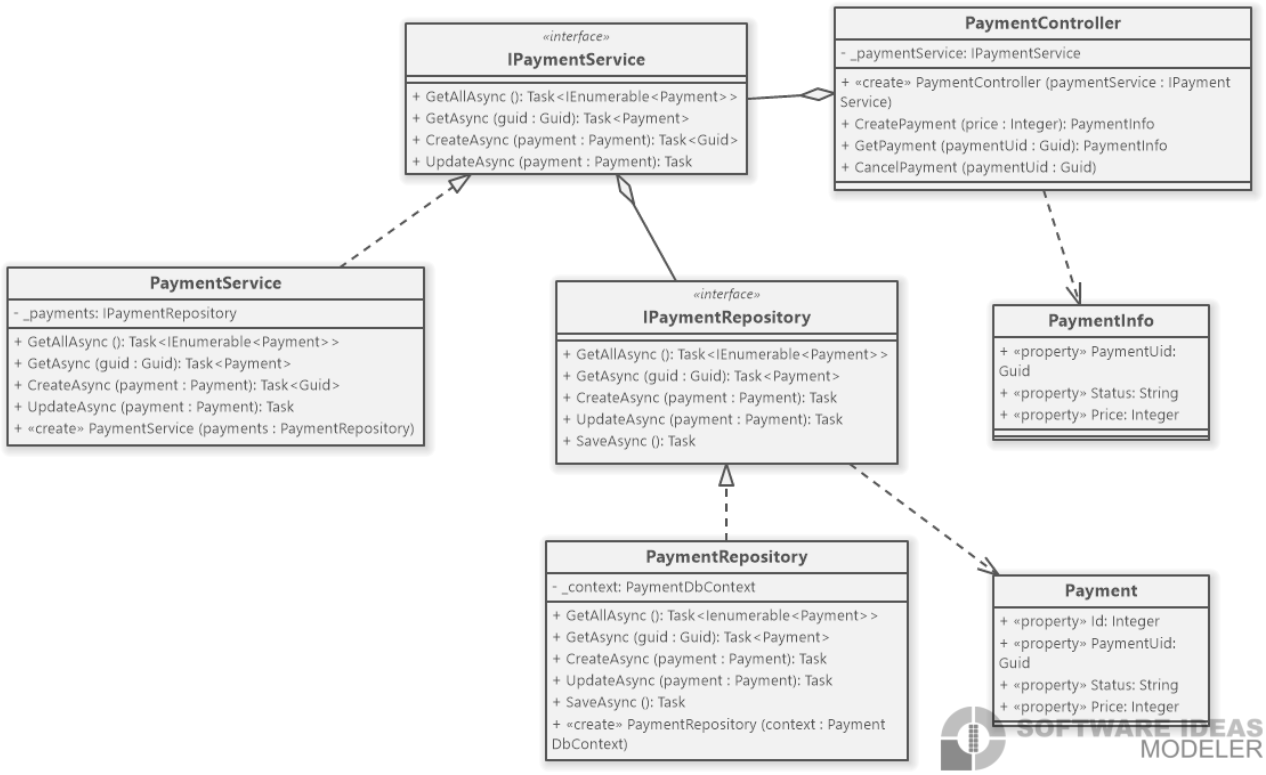

Рисунок 2.8 – Диаграмма классов сервиса оплат

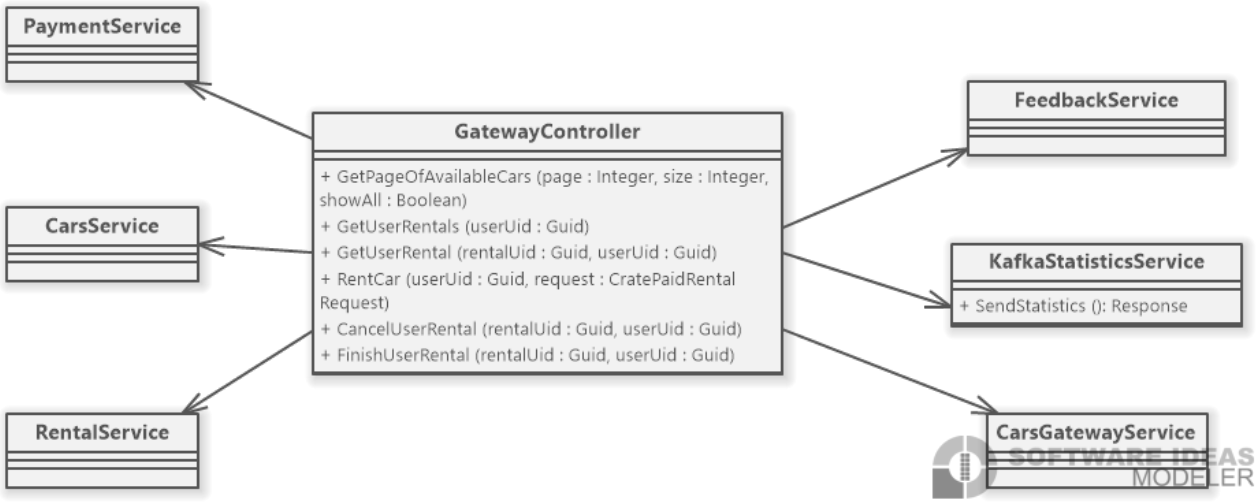

Рисунок 2.9 – Диаграмма классов сервиса-координатора

Описание основных классов сервисов

Сервисы автомобилей, аренд, оплат и отзывов спроектированы похожим образом. Они имеют:

- слой доступа к данным, реализованный с помощью паттерна «Репозиторий» для абстракции хранения и для эффективного использования активных подключений к базе данных;
- слой логики работы сервиса;
- слой интерфейса (контроллер) – слой связи с другими сервисами с помощью http-запросов.

Классы контроллеров описывают набор HTTP-методов, которые доступны на сервисе. Фактически занимается распаковкой данных, пришедших по сети, заполнением необходимых структур этими данными и вызовом функций, реализующих логику работы методов. Также занимаются подготовкой результирующих данных к отправке по сети после выполнения запроса.

Описание классов сервиса автомобилей

Car – класс, описываюий автомобиль, имеет поля:

- Идентификатор автомобиля;
- Бренд;

- Модель;
- Регистрационный номер;
- Мощность;
- Цена за суточную аренду;
- Тип автомобиля;
- Доступность для аренды;

CarInfo – класс, содержащий краткую информацию об автомобиле в виде, пригодном для пользователя.

CarResponse – класс, содержащий полную информацию об автомобиле в виде, пригодном для пользователя.

PaginationResponse – класс для пагинации автомобилей.

CarsController – контроллер, описывающий набор HTTP-методов, который доступен на сервисе автомобилей.

ICarsRepository – абстрактный интерфейс для работы с данными автомобилей (интерфейс спроектирован в соответствии с паттерном «репозиторий»). CarsRepository – реализация этого абстрактного интерфейса для работы с данными, хранящимися под управлением СУБД Postgres.

ICarsService – абстрактный интерфейс, содержащий слой логики работы с данными. CarsService – реализация этого абстрактного интерфейса.

Описание классов сервиса аренд

Rental – класс, описывающий запись об аренде автомобиля, имеет поля:

- Идентификатор аренды;
- Идентификатор пользователя;
- Идентификатор оплаты;
- Идентификатор автомобиля;
- Дата начала аренды;
- Дата конца аренды;

- Статус аренды;

CreatePaidRentalRequest – класс, описывающий запрос на создание записи об аренде после её оплаты.

RentalController – контроллер, описывающий набор HTTP-методов, который доступен на сервисе аренд.

IRentalRepository – абстрактный интерфейс для работы с данными аренд (интерфейс спроектирован в соответствии с паттерном «репозиторий»). RentalRepository – реализация этого абстрактного интерфейса для работы с данными, хранящимися под управлением СУБД Postgres.

IRentalService – абстрактный интерфейс, содержащий слой логики работы с данными. RentalService – реализация этого абстрактного интерфейса.

Описание классов сервиса оплат

Payment – класс, описывающий информацию о платеже, имеет поля:

- Идентификатор оплаты;
- Статус платежа;
- Сумма оплаты;

PaymentInfo – класс, содержащий информацию о платеже в виде, пригодном для пользователя.

PaymentController – контроллер, описывающий набор HTTP-методов, который доступен на сервисе платежей.

IPaymentRepository – абстрактный интерфейс для работы с данными платежей (интерфейс спроектирован в соответствии с паттерном «репозиторий»). PaymentRepository – реализация этого абстрактного интерфейса для работы с данными, хранящимися под управлением СУБД Postgres.

IPaymentService – абстрактный интерфейс, содержащий слой логики работы с данными. PaymentService – реализация этого абстрактного интерфейса.

Описание классов сервиса-координатора

GatewayController – класс, описывающий набор HTTP-методов, которые доступны на сервисе. Фактически представляет собой весь программный

интерфейс системы. В своей работе для обслуживания приходящих запросов сервис использует все вышеописанные интерфейсы сервисов, а также сервис перенаправляет статистику запросов в очередь Kafka [3] с помощью интерфейса KafkaStatistcsService.

3 Технологический раздел

3.1 Выбор операционной системы

Согласно требованиям технического задания, разрабатываемый портал должен обладать высокой доступностью, работать на типичных архитектурах ЭВМ (Intel x86, Intel x64), а так же быть экономически недорогим для сопровождения. Таким образом, можно сформулировать следующие требования к операционной системе:

- Распространенность. На рынке труда должно быть много специалистов, способных администрировать распределенную систему, работающую под управлением выбранной операционной системы. Windows, будучи самой распространенной операционной системой для настольных и серверных компьютеров, полностью отвечает этому требованию. Найти квалифицированных системных администраторов с опытом работы на Windows Server не составит труда.

- Надежность. Операционная система должна широко использоваться в стабильных проектах, таких как Mail.Ru, Vk.com, Google.com. Эти компании обеспечивают высокую работоспособность своих сервисов, и на их опыт можно положиться. Windows Server зарекомендовала себя как надежная и стабильная платформа, используемая множеством крупных компаний, включая Amazon, Microsoft и другие, обеспечивающие высокую доступность своих сервисов.

- Наличие требуемого программного обеспечения. Выбор операционной системы не должен ограничивать разработчиков в выборе программного обеспечения, библиотек. Windows обладает обширной экосистемой программного обеспечения, среди которого большинство популярных языков программирования, фреймворков и баз данных имеют версии для Windows.

- Цена. Windows Server является коммерческим продуктом и требует приобретения лицензии. Однако, Microsoft предлагает различные варианты лицензирования, позволяющие подобрать оптимальное соотношение цены и функционала в зависимости от масштаба проекта.

Windows Server [4] может стать подходящим выбором для реализации портала, соответствуя требованиям высокой доступности, распространенности, наличию необходимых инструментов разработки и оптимальной стоимости владения.

3.2 Выбор СУБД

В качестве СУБД была выбрана PostgreSQL [5], так как она наилучшим образом подходит под требования разрабатываемой системы:

- Масштабируемость: PostgreSQL поддерживает горизонтальное масштабирование, что позволяет распределить данные и запросы между несколькими узлами базы данных. Это особенно полезно в географически распределенных системах, где данные и пользователи могут быть разбросаны по разным регионам.

- Географическая репликация: PostgreSQL предоставляет возможность настройки репликации данных между различными узлами базы данных, расположенными в разных географических зонах. Это позволяет обеспечить отказоустойчивость и более быстрый доступ к данным для пользователей из разных частей нашей страны.

- Гибкость и функциональность: PostgreSQL обладает широким набором функций и возможностей, что делает его подходящим для различных типов приложений и использования в распределенной среде. Он поддерживает сложные запросы, транзакции, хранимые процедуры и многое другое.

- Надежность и отказоустойчивость: PostgreSQL известен своей надежностью и стабильностью работы. В распределенной географической системе это особенно важно, поскольку он способен обеспечить сохранность данных и доступность даже при сбоях в отдельных узлах.

3.3 Выбор языка разработки и фреймворков компонент портала

Исходя из требований к системе, можно обосновать выбор языка программирования C# [6] и фреймворка ASP.NET Core [7] для разработки портала

следующими аргументами:

1. Язык программирования C#:

   - Кроссплатформенность: C# – современный объектно-ориентированный язык программирования, работающий на платформе .NET. Благодаря .NET Core, приложения на C# могут запускаться на различных операционных системах, включая Linux, что обеспечивает совместимость с выбранной ОС.
   - Интеграция с PostgreSQL: C# и .NET предоставляют богатый набор инструментов для работы с базами данных, включая PostgreSQL. Библиотеки, такие как Npgsql, обеспечивают удобное и эффективное взаимодействие с PostgreSQL, что соответствует техническому заданию.
   - Производительность: C# компилируется в промежуточный язык (IL), который затем исполняется виртуальной машиной .NET. Это обеспечивает высокую производительность, сравнимую с компилируемыми языками, такими как C++.
   - Расширяемость: C# поддерживает объектно-ориентированный подход, что упрощает разработку сложных и масштабируемых систем. Богатая экосистема библиотек .NET предоставляет готовые решения для множества задач, сокращая время разработки и повышая надежность проекта.

2. Фреймворк ASP.NET Core:

   - Высокая производительность: ASP.NET Core - это быстрый и легковесный фреймворк, оптимизированный для обработки веб-запросов. Он обеспечивает высокую пропускную способность и быстрый отклик, что важно для производительности портала.
   - Встроенная поддержка многопоточности: ASP.NET Core эффективно использует многопоточность для обработки большого количества запросов одновременно. Это обеспечивает масштабируемость и отзывчивость портала, даже при высокой нагрузке.

- Создание RESTful API: ASP.NET Core предоставляет мощные инструменты для разработки RESTful API, что упрощает интеграцию с другими системами и создание мобильных приложений.
- Модульность и расширяемость: ASP.NET Core построен на основе модульной архитектуры, позволяющей легко расширять функциональность приложения. Благодаря широкому выбору библиотек и компонентов .NET, ASP.NET Core обеспечивает гибкость и масштабируемость для реализации различных требований проекта.

Таким образом, выбор C# и ASP.NET Core для разработки портала обоснован высокой производительностью, совместимостью с выбранными технологиями, расширяемостью, удобством разработки и богатой экосистемой инструментов.

3.4 Выбор фреймворка фронтенд разработки

Для разработки клиентской части веб-приложения был выбран фреймворк React [8] по следующим причинам.

1. Популярность и поддержка сообщества: React является одним из самых популярных инструментов для фронтенд-разработки. Благодаря этому для React доступно множество библиотек, инструментов и ресурсов. Это значительно упрощает разработку, особенно при необходимости интеграции с другими технологиями.

2. Гибкость: В отличие от Angular, который является "жёстким"фреймворком с предустановленными решениями, React предлагает больше свободы. Разработчики могут выбирать любые библиотеки для маршрутизации, управления состоянием и работы с сервером, адаптируя проект под конкретные требования.

3. Производительность через Virtual DOM: React использует Virtual DOM для минимизации реальных изменений в DOM, что делает его быстрым даже для сложных пользовательских интерфейсов. Это особенно важно для крупных приложений с динамически изменяющимися данными.

4. Поддержка мобильной разработки: React Native предоставляет возможность использовать знания и компоненты React для разработки

мобильных приложений под iOS и Android. Это позволяет создавать кроссплатформенные приложения с минимальными усилиями.

ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы была проведена разработка вебприложения для аренды автомобилей, включающая как клиентскую, так и серверную части. На основании проведённого анализа современных технологий были выбраны оптимальные инструменты для создания высокопроизводительного и масштабируемого решения. В качестве бэкенд-фреймворка был использован ASP.NET языка программирования C#, обеспечивающий высокую производительность, поддержку асинхронного программирования и простоту интеграции с современными инструментами. Для фронтенда был выбран React, отличающийся гибкостью, большим сообществом и поддержкой современных подходов к созданию интерактивных пользовательских интерфейсов.

Разработанное приложение предоставляет удобный интерфейс для поиска и аренды автомобилей, а также обеспечивает автоматизированный сбор и обработку данных о пользователях и аренде. В процессе реализации были решены задачи по организации архитектуры приложения, обеспечению безопасности данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Identity Provider [Электронный ресурс]. — Режим доступа URL: https://www.okta.com/identity-101/why-your-company-needs-an-identity-provider/ (Дата обращения: 09.04.2024).
2. Docker [Электронный ресурс]. — Режим доступа URL: https://www.docker.com/ (Дата обращения: 09.04.2024).
3. Kafka [Электронный ресурс]. — Режим доступа URL: https://kafka.apache.org (Дата обращения: 09.04.2024).
4. Microsoft Windows Server [Электронный ресурс]. — Режим доступа URL: https://www.microsoft.com/windows-server (Дата обращения: 09.04.2024).
5. PostgreSQL [Электронный ресурс]. — Режим доступа URL: https://www.postgresql.org (Дата обращения: 09.04.2024).
6. C# [Электронный ресурс]. — Режим доступа URL: https://learn.microsoft.com/dotnet/csharp/ (Дата обращения: 09.04.2024).
7. ASP.NET Core [Электронный ресурс]. — Режим доступа URL: https://dotnet.microsoft.com/apps/aspnet (Дата обращения: 09.04.2024).
8. React [Электронный ресурс]. — Режим доступа URL: https://react.dev (Дата обращения: 09.04.2024).