



Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

«Система бронирования авиабилетов»

Студент ИУ7И-22М
(Группа)

(Подпись, дата)

Динь Вьет Ань
(И.О.Фамилия)

Руководитель

(Подпись, дата)

А. А. Ступников
(И.О.Фамилия)

2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1 Аналитическая часть	6
1.1. Описание системы	6
1.2. Функциональные требования к системе с точки зрения пользователя	6
1.3. Входные данные	7
1.4. Выходные параметры	9
1.5. Состав системы	11
1.5.1. Фронтенд	11
1.5.2. Сервис-координатор	11
1.5.3. Сервис регистрации и авторизации	12
1.5.4. Сервис полетов	13
1.5.5. Сервис билетов	14
1.5.6. Сервис бонусов	15
1.5.7. Сервис статистики	16
1.5.8. Сервис kafka	16
1.5.9. Сервис consumer	16
1.5.10. Сервис zookeeper	17
1.6. Требования к программной реализации	18
1.7. Функциональные требования к подсистемам	18
1.8. Пользовательский интерфейс	20
1.9. Сценарий взаимодействия с приложением	20
2 Конструкторская часть	22
2.1. Концептуальный дизайн	22
2.2. Сценарии функционирования системы	24
2.3. Диаграммы прецедентов	25
3 Технологическая часть	28
3.1. Выбор операционной системы	28
3.2. Выбор СУБД	28

3.3. Выбор языка разработки и фреймворков компонент портала	29
3.4. Выбор фреймворка фронтенд разработки	31
3.5. Высокоуровневый дизайн пользовательского интерфейса	32
ЗАКЛЮЧЕНИЕ	36
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	37

ВВЕДЕНИЕ

В современном мире, где скорость и удобство имеют первостепенное значение, система бронирования авиабилетов стала неотъемлемой частью путешествий. Появление интернета и развитие технологий позволили перевести процесс бронирования в онлайн-среду, сделав его доступным и удобным для миллионов людей по всему миру.

С каждым годом количество пассажиров, пользующихся авиаперевозками, увеличивается. Это обусловлено увеличением доступности авиаперелетов, развитием инфраструктуры и увеличением спроса на туризм. Создание современных систем бронирования позволяет удовлетворить возрастающий спрос на авиаперевозки и обеспечить удобство пользователей.

В сфере авиаперевозок увеличивается конкуренция между авиакомпаниями. Создание современной системы бронирования с удобным интерфейсом, широким выбором рейсов и эффективной системой оплаты позволяет привлечь больше клиентов и укрепить конкурентные позиции.

Целью курсовой работы является разработка системы бронирования авиабилетов. Для ее достижения необходимо выполнить следующие задачи:

- описать разрабатываемую систему;
- сформулировать требования к системе бронирования авиабилетов;
- спроектировать архитектуру распределенной системы;
- произвести выбор стека технологий для реализации системы;
- реализовать распределенную систему для бронирования авиабилетов.

1 Аналитическая часть

1.1. Описание системы

Разрабатываемый портал должен представлять собой систему для покупки авиабилетов.

Если пользователь хочет купить билет, то ему нужно пройти регистрацию, указав следующую информацию: фамилия, имя, номер телефона, адрес электронной почты, пароль. Для неавторизованных пользователей доступен только просмотр общей информации сайта: списка рейсов с указанием даты и времени отлета, аэропортов отправления и прибытия, цены билета.

1.2. Функциональные требования к системе с точки зрения пользователя

Портал должен обеспечивать реализацию следующих функций.

1. Система должна обеспечивать регистрацию и авторизацию пользователей с валидацией вводимых данных.
2. Аутентификация пользователей.
3. Разделение всех пользователей на 3 роли:
 - неавторизованный пользователь (гость);
 - авторизованный пользователь (пользователь);
 - администратор.
4. Предоставление возможностей **гостю, пользователю, администратору** представленных в таблице 1.1.

Таблица 1.1 – Функции пользователей

Гость	<ul style="list-style-type: none"> 1. Просмотр списка рейсов (включая фильтрацию и сортировку по всем полям); 2. Регистрация в системе; 3. Авторизация в системе.
Пользователь	<ul style="list-style-type: none"> 1. Авторизация в системе; 2. Просмотр списка рейсов (включая фильтрацию и сортировку по всем полям); 4. Получение информации о данных текущего аккаунта; 5. Просмотр истории покупок билетов; 6. Получение детальной информации по билету; 7. Просмотр списка купленных и сданных билетов; 8. Покупка билета на выбранный рейс; 9. Возврат билета.
Администратор	<ul style="list-style-type: none"> 1. Функции пользователя; 2. Просмотр статистики по сайту.

1.3. Входные данные

Входные параметры системы представлены в таблице 1.2.

Таблица 1.2 – Входные данные

Сущность	Входные данные
Регистрация пользователя	<ul style="list-style-type: none"> 1. <i>фамилия</i> не более 256 символов; 2. <i>имя</i> не более 256 символов; 3. <i>логин</i> не более 256 символов; 4. <i>пароль</i> не более 128 символов; 5. <i>номер телефона</i> в формате (+7XXXXXXXXXX); 6. <i>роль</i> администратор или пользователь; 7. <i>электронная почта</i> в формате (*@*.*).

Продолжение на следующей странице

Сущность	Входные данные
Аутентификация пользователя	1. <i>логин</i> не более 256 символов; 2. <i>пароль</i> не более 128 символов.
Покупка билета	1. <i>номер рейса</i> ; 2. <i>цена билета</i> ; 3. <i>флаг, отвечающий за списывание или зачисление бонусов</i> .
Возврат билета	1. <i>идентификатор билета</i> .
Получение детальной информации о билете	1. <i>идентификатор билета</i> .
Фильтр и пагинация полетов	1. <i>номер полета</i> не более 20 символов; 2. <i>минимальная цена</i> не менее 1; 3. <i>максимальная цена</i> не менее 1; 4. <i>минимальное время вылета</i> ; 5. <i>максимальное время вылета</i> ; 6. <i>аэропорт отправления</i> ; 7. <i>аэропорт прибытия</i> ; 8. <i>номер страницы</i> не менее 1; 9. <i>размер страницы</i> не менее 1; 10. <i>объектов на странице</i> не менее 1.

1.4. Выходные параметры

Выходными параметрами системы являются web-страницы. В зависимости от запроса и текущей роли пользователя они содержат следующую информацию (таблица 1.3).

Таблица 1.3 – Выходные параметры

Гость	1. Список рейсов: <ul style="list-style-type: none">• номер полета;• аэропорт отправления;• аэропорт прибытия;• дата и время отлета;• цена билета.
	2. Окно авторизации.
	3. Окно регистрации.
Пользователь	1. Список рейсов: <ul style="list-style-type: none">• номер полета;• аэропорт отправления;• аэропорт прибытия;• дата и время отлета;• цена билета.
	2. Список купленных и сданных билетов: <ul style="list-style-type: none">• количество бонусов на счету данного пользователя;• номер полета;• аэропорт отправления;• аэропорт прибытия;• дата и время отлета;• итоговая цена билета с учетом бонусов;• статус билета (куплен или сдан).

Администратор	<p>3. Детальная информация о пользователе, вошедшем в систему;</p> <ul style="list-style-type: none"> • фамилия; • имя; • логин; • роль; • электронная почта; • номер телефона; • количество бонусов на счету данного пользователя.
	<p>4. История покупок билетов:</p> <ul style="list-style-type: none"> • дата и время покупки/сдачи билета; • количество начисленных/списанных бонусов.
	<p>1. Список рейсов:</p> <ul style="list-style-type: none"> • номер полета; • аэропорт отправления; • аэропорт прибытия; • дата и время отлета; • цена билета.
	<p>2. Список купленных и сданных билетов:</p> <ul style="list-style-type: none"> • количество бонусов на счету данного пользователя; • номер полета; • аэропорт отправления; • аэропорт прибытия; • дата и время отлета; • итоговая цена билета с учетом бонусов; • статус билета (куплен или сдан).
	<p>3. Детальная информация о пользователе, вошедшем в систему;</p> <ul style="list-style-type: none"> • фамилия; • имя; • логин; • роль; • электронная почта; • номер телефона; • количество бонусов на счету данного пользователя.

<p>4. История покупок билетов:</p> <ul style="list-style-type: none"> • <i>дата и время покупки/сдачи билета;</i> • <i>количество начисленных/списанных бонусов.</i>
<p>5. Статистика по portalу, собранная через сервис статистики:</p> <ul style="list-style-type: none"> • <i>метод запроса;</i> • <i>url запроса;</i> • <i>числовой статус выполнения запроса;</i> • <i>время выполнения запроса.</i>

1.5. Состав системы

Система будет состоять из фронтенда и 9 подсистем:

- сервис-координатор;
- сервис регистрации и авторизации;
- сервис полетов;
- сервис билетов;
- сервис бонусов;
- сервис статистики;
- сервис kafka;
- сервис consumer;
- сервис zookeeper.

1.5.1. Фронтенд

Фронтенд – принимает запросы от пользователя по протоколу HTTP и возвращает ответ в виде HTML страниц, файлов стилей и TypeScript.

1.5.2. Сервис-координатор

Сервис-координатор – сервис, который отвечает за координацию запросов внутри системы. Все сервисы портала (кроме сервиса регистрации и авторизации) должны взаимодействовать друг с другом через сервис-координатор,

запросы с фронтенда в том числе сначала должны приходить на сервис-координатор, а затем перенаправляться на нужный сервис. При этом сервис-координатор отвечает за следующие действия.

1. получения списка рейсов с пагинацией, фильтрацией и сортировкой от сервиса полетов;
2. получения аэропортов с фильтрацией от сервиса полетов;
3. получения списка билетов разных пользователей из сервиса билетов;
4. получения списка истории покупок билетов пользователя из сервиса бонусов;
5. получения информации о состоянии бонусного счета пользователя из сервиса бонусов;
6. оформление покупки билета через сервисы полетов и билетов с учетом бонусного счета пользователя из сервиса бонусов;
7. возврат билета через сервисы полетов и билетов и с изменением данных в сервисе бонусов (списание ранее начисленных бонусов или начисление ранее списанных).

1.5.3. Сервис регистрации и авторизации

Сервис регистрации и авторизации отвечает за следующие действия.

1. Регистрацию нового пользователя;
2. Аутентификацию пользователя;
3. Авторизацию пользователя;
4. Получение данных пользователей;
5. Изменение данных о пользователе;
6. Удаление пользователя.

Взаимодействие сервиса регистрации и авторизации с остальными сервисами должно осуществляться по протоколу OpenID Connect. Сам сервис представляет из себя Identity Provider. Сервис регистрации и авторизации в своей работе используют базу данных, которая хранит следующую информацию:

- Пользователь:
 - *уникальный идентификатор;*
 - *логин;*
 - *имя;*
 - *фамилия;*
 - *захешированный пароль;*
 - *номер телефона;*
 - *электронная почта;*
 - *роль.*

1.5.4. Сервис полетов

Сервис полетов реализует следующие функции.

1. Получение списка всех рейсов с фильтрацией, сортировкой и пагинацией;
2. Получение информации о конкретном рейсе;
3. Создание полета;
4. Удаление полета;
5. Получение списка всех аэропортов с пагинацией;
6. Получение информации о конкретном аэропорте;
7. Создание аэропорта;
8. Удаление аэропорта.

Сервис использует в своей работе базу данных:

- Полет:

- *уникальный идентификатор;*
 - *номер полета;*
 - *цена билета;*
 - *дата и время отправления;*
 - *идентификатор аэропорта отправления;*
 - *идентификатор аэропорта прибытия.*
- Аэропорт:
- *уникальный идентификатор;*
 - *название;*
 - *город;*
 - *страна.*

1.5.5. Сервис билетов

Сервис билетов реализует следующие функции.

1. Получение списка билетов всех пользователей с фильтрацией, сортировкой и пагинацией;
2. Получение информации о конкретном билете;
3. Создание билета;
4. Изменение билета;
5. Удаление билета.

Сервис использует в своей работе базу данных:

- Билет:
- *уникальный идентификатор;*
 - *uuid билета;*
 - *логин пользователя;*
 - *номер полета;*
 - *цена;*
 - *статус (PAID/CANCELED).*

1.5.6. Сервис бонусов

Сервис бонусов реализует следующие функции.

1. Получение списка бонусных счетов всех пользователей с фильтрацией, сортировкой и пагинацией;
2. Получение информации о конкретном бонусном счете;
3. Создание бонусного счета;
4. Изменение бонусного счета;
5. Удаление бонусного счета;
6. Получение списка историй начисления и списания бонусов всех пользователей с фильтрацией и сортировкой;
7. Получение информации о конкретной истории;
8. Создание истории;
9. Изменение истории;
10. Удаление истории;

Сервис использует в своей работе базу данных:

- Бонусный счет:
 - *уникальный идентификатор;*
 - *логин пользователя;*
 - *статус клиента (BRONZE/SILVER/GOLD);*
 - *баланс.*
- История изменения бонусного счета:
 - *уникальный идентификатор;*
 - *идентификатор бонусного счета;*
 - *время покупки/возврата билета;*
 - *количество списанных/начисленных бонусов;*
 - *тип операции (бонусы начислены или списаны).*

1.5.7. Сервис статистики

Сервис статистики – сервис, который отвечает за запись событий сервиса координатора в базу данных для осуществления возможности быстрого обнаружения, локализации и воспроизведения ошибки в случае её возникновения. Дает возможность получить статистику с пагинацией.

Сервис использует в своей работе базу данных:

- Статистика:
 - *уникальный идентификатор;*
 - *метод запроса GET/POST/PATCH/DELETE/OPTIONS;*
 - *url запроса;*
 - *числовой статус выполнения запроса;*
 - *время выполнения запроса.*

1.5.8. Сервис kafka

Сервис kafka [1] – сервис, который необходим для сервиса статистики для сбора и обработки данных в реальном времени, что позволяет анализировать пользовательскую активность. Kafka поддерживает высокие объёмы данных и легко масштабируется, обеспечивая надёжную работу даже при значительных нагрузках. Благодаря встроенной отказоустойчивости и гарантии доставки сообщений, система статистики не потеряет важные данные при сбоях.

1.5.9. Сервис consumer

Сервис consumer – сервис, который нужен kafka для получения, обработки и анализа данных, поступающих от producer в реальном времени. Kafka действует как посредник, обеспечивая доставку сообщений между различными сервисами, что позволяет consumer-серверам асинхронно получать данные и обрабатывать их по мере поступления. Это важно для поддержания высокой производительности и отказоустойчивости, так как Kafka распределяет нагрузку между несколькими consumer-серверами, помогая избежать перегрузки. Также Kafka гарантирует надёжную доставку сообщений, что позволяет consumer корректно обрабатывать каждое сообщение без риска потери данных. Наконец,

она обеспечивает возможность параллельной обработки данных, что ускоряет анализ больших объёмов информации.

1.5.10. Сервис zookeeper

Сервис zookeeper [2] – сервис, который нужен kafka для управления и координации различных компонентов в своей распределённой системе. Вот ключевые задачи, которые решает Zookeeper в Kafka.

1. Координация кластеров: Zookeeper помогает координировать работу брокеров (серверов Kafka) внутри кластера, отслеживая их состояние. Он сообщает Kafka о том, какие узлы доступны и активно работают, обеспечивая бесперебойное взаимодействие между ними.
2. Управление метаданными: Zookeeper хранит важную информацию о топиках, партициях и распределении лидеров партиций среди брокеров. Это нужно для того, чтобы потребители (consumers) и производители (producers) могли эффективно взаимодействовать с нужными данными в кластере.
3. Обнаружение лидера: Zookeeper определяет лидера для каждой партиции Kafka, который отвечает за запись и чтение данных. В случае сбоя одного из брокеров Zookeeper автоматически выбирает нового лидера для партиции, чтобы поддерживать непрерывную работу.
4. Отказоустойчивость: Zookeeper обеспечивает высокую доступность и надёжность кластера Kafka, помогая восстанавливать компоненты после сбоев и поддерживать согласованное состояние всех узлов системы. Управление доступом: Zookeeper управляет доступом клиентов к Kafka и координирует изменения конфигурации, обеспечивая стабильность и безопасность работы кластера.
5. Таким образом, Zookeeper является критически важным компонентом для обеспечения координации, отказоустойчивости и управления Kafka-кластером.

1.6. Требования к программной реализации

1. Требуется использовать СОА (сервис-ориентированную архитектуру) для реализации системы.
2. Система состоит из микросервисов. Каждый микросервис отвечает за свою область логики работы приложения и должны быть запущены изолированно друг от друга.
3. При необходимости, каждый сервис имеет своё собственное хранилище, запросы между базами запрещены.
4. При разработке базы данных необходимо учитывать, что доступ к ней должен осуществляться по протоколу TCP.
5. Необходимо реализовать один web-интерфейс для фронтенда. Интерфейс должен быть доступен через тонкий клиент (браузер).
6. Для межсервисного взаимодействия использовать HTTP (придерживаться RESTful).
7. Выделить Gateway Service как единую точку входа и межсервисной коммуникации. В системе не должно осуществляться горизонтальных запросов.
8. Необходимо предусмотреть авторизацию пользователей через интерфейс приложения.
9. Код хранить на Github, для сборки использовать Github Actions.
10. Каждый сервис должен быть завернут в docker.

1.7. 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;
- целесообразно использовать Tailwind для упрощения написания стилей.

Сервис-координатор – 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;
- использовать очередь для отложенной обработки запросов (например, при временном отказе одного из сервисов);
- осуществлять деградацию функциональности в случае отказа некритического сервиса (зависит от семантики запроса);
- уведомлять сервис статистики о событиях в системе.

Сервис регистрации и авторизации, сервис библиотек, сервис рейтинга, сервис аренды, сервис статистики – 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 kafka, сервис consumer, сервис zookeeper – 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.

1.8. Пользовательский интерфейс

Для реализации пользовательского интерфейса должен быть использован подход MVC (Model-View-Controller). Этот подход к проектированию интерфейса является популярным шаблоном проектирования, который помогает разделить логику приложения на три основных компонента: Модель (Model), Представление (View) и Контроллер (Controller). Этот подход позволяет улучшить структуру приложения, облегчить его тестирование и управление, а также разработка фронтенда и бекенда могут быть полностью разделены между собой, то есть можно вести независимую разработку.

Пользовательский интерфейс в разрабатываемой системе должен обладать следующими характеристиками:

- Кроссбраузерность – способность интерфейса работать практически в любом браузере любой версии.
- «Плоский» дизайн – дизайн, в основе которого лежит идея отказа от объемных элементов (теней элементов, объемных кнопок и т.д.) и замены их плоскими аналогами.
- Расширяемость – возможность легко расширять и модифицировать пользовательский интерфейс.

1.9. Сценарий взаимодействия с приложением

Приведем пример работы портала на примере выполнения запроса от пользователя на получение списка купленных и сданных билетов.

1. На фронтенд приходит запрос пользователя.
2. Если пользователь был авторизован, то происходит получение токена аутентификации. Затем выполняется запрос к сервису-координатору. Если данные корректны (данные поступили в ожидаемом формате) и проверка JWT-токена (проверка того, что токен был подписан известным серверу ключом и того, что срок действия токена ещё не истёк) (если он истёк или оказался некорректным, пользователю возвращается ошибка), то отправляются запросы на сервисы билетов и бонусов.

3. Выполняются запросы к соответствующим эндпоинтам сервисов билетов для получения данных о купленных и сданных билетах пользователя, осуществляется проверка корректности полученных данных (данные поступили в ожидаемом формате) и проверка JWT-токена (проверка того, что токен был подписан известным серверу ключом и того, что срок действия токена ещё не истёк). Если он истёк или оказался некорректным, пользователю возвращается ошибка. При успешной проверке токена сервис возвращает список билетов сервису-координатору, который агрегирует полученные данные с балансом бонусного счета и происходит возврат результата на фронтенд.
4. Если ошибки нигде не произошло, то производится генерация HTML содержимого страницы ответа пользователю с использованием данных, полученных от сервиса-координатора. В ином случае генерируется страница с описанием ошибки.

2 Конструкторская часть

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 детализированная концептуальная модель системы в нотации IDEF0.



Рисунок 2.1 – Концептуальная модель системы в нотации IDEF0

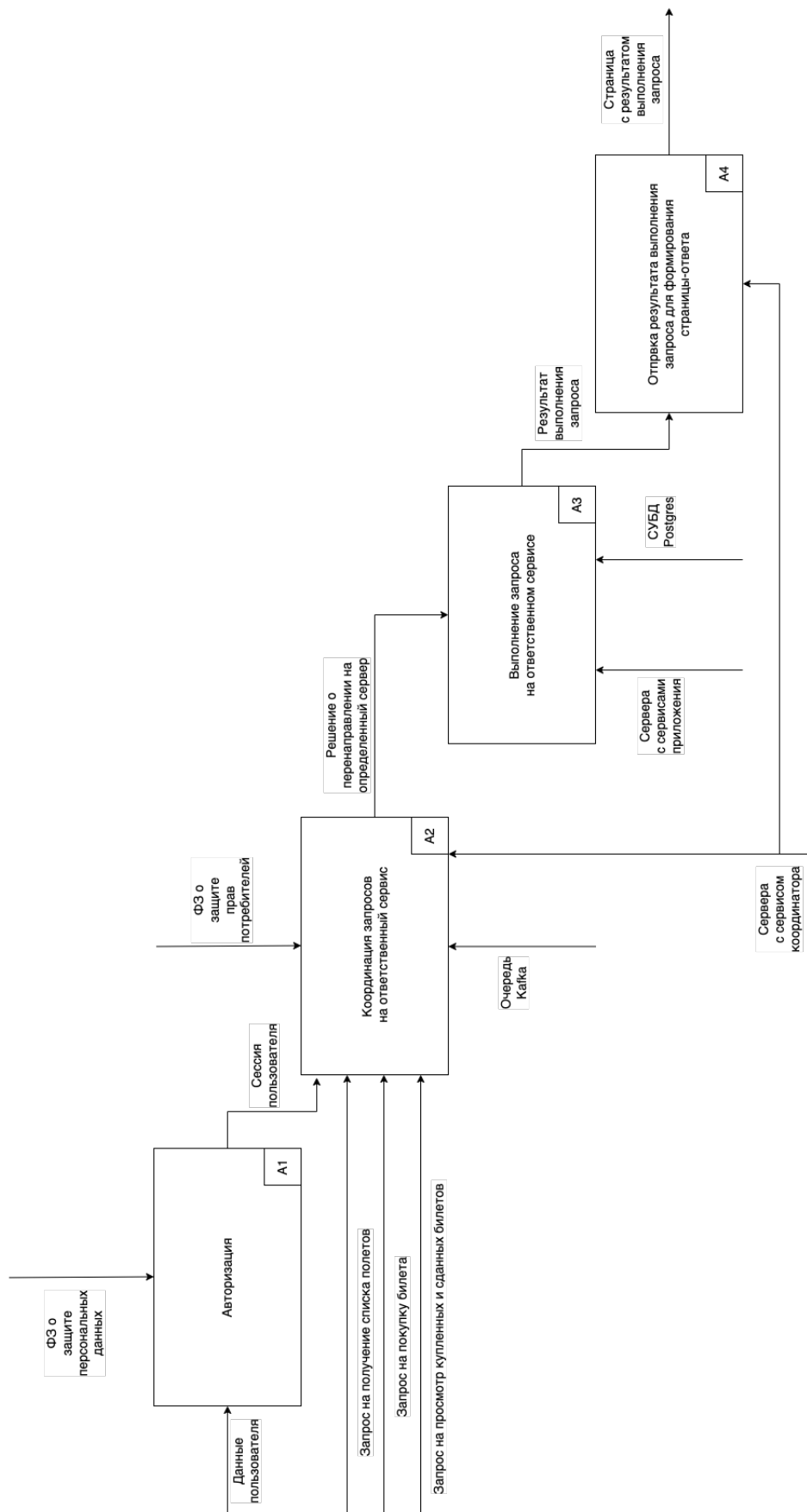


Рисунок 2.2 – Детализированная концептуальная модель системы в нотации IDEF0

2.2. Сценарии функционирования системы

Регистрация пользователя

1. Пользователь нажимает на кнопку «Зарегистрироваться» в интерфейсе.
2. Пользователь перенаправляется на страницу, которая содержит поля для заполнения его данных.
3. Пользователь вводит данные в форму и для завершения регистрации нажимает на кнопку «Зарегистрироваться», тем самым подтверждая верность своих данных, а также согласие на их обработку и хранение.
4. Если пользователь с введенным для регистрации логином, почтой или номером телефона уже существует, то клиент получает сообщение об ошибке. При успешной регистрации клиент попадает на страницу со списком полетов.

Авторизация клиента

1. Пользователь нажимает на кнопку «Авторизоваться» в интерфейсе.
2. Пользователь перенаправляется на страницу авторизации, которая содержит поля для заполнения логина и пароля.
3. Пользователь завершает работу с формой авторизации нажатием кнопки «Войти».
4. При обнаружении ошибки в данных, пользователь получает сообщение об ошибке; при совпадении данных с записью в базе данных аккаунтов пользователь получает доступ к системе и перенаправляется на страницу со списком полетов.

Покупка билета

1. Пользователь на главной странице видит таблицу со списком полетов. При желании он может отфильтровать и отсортировать их по каждому полю.
2. Пользователь нажимает на кнопку «Купить билет» у выбранного рейса.

3. Пользователь выбирает, что делать с бонусами (списать или копить) и нажимает кнопку «Забронировать».
4. В появившемся модальном окне подтверждения с подробной информацией о выбранном рейсе пользователь нажимает на кнопку «Подтвердить».
5. Пользователь видит модальное окно с информацией об успешности совершенной покупки и нажимает кнопку «Окей».

Возврат купленного билета

1. Для просмотра купленных билетов пользователь переходит на страницу «Билеты», нажимая соответствующую кнопку в выезжающем боковом меню.
2. Пользователь попадает на страницу, на которой видит все свои купленные и сданные билеты.
3. У каждого купленного билета есть кнопка «Сдать билет», на которую нажимает пользователь.
4. В появившемся модальном окне подтверждения с подробной информацией о выбранном билете пользователь нажимает на кнопку «Подтвердить».
5. Пользователь видит измененный статус билета на «Билет сдан» и обновленный баланс бонусного счета.

2.3. Диаграммы прецедентов

В системе выделены 3 роли: Неавторизованный пользователь, Авторизованный пользователь, Администратор. На рисунках 2.3-2.5 представлены диаграммы прецедентов для каждой из ролей.

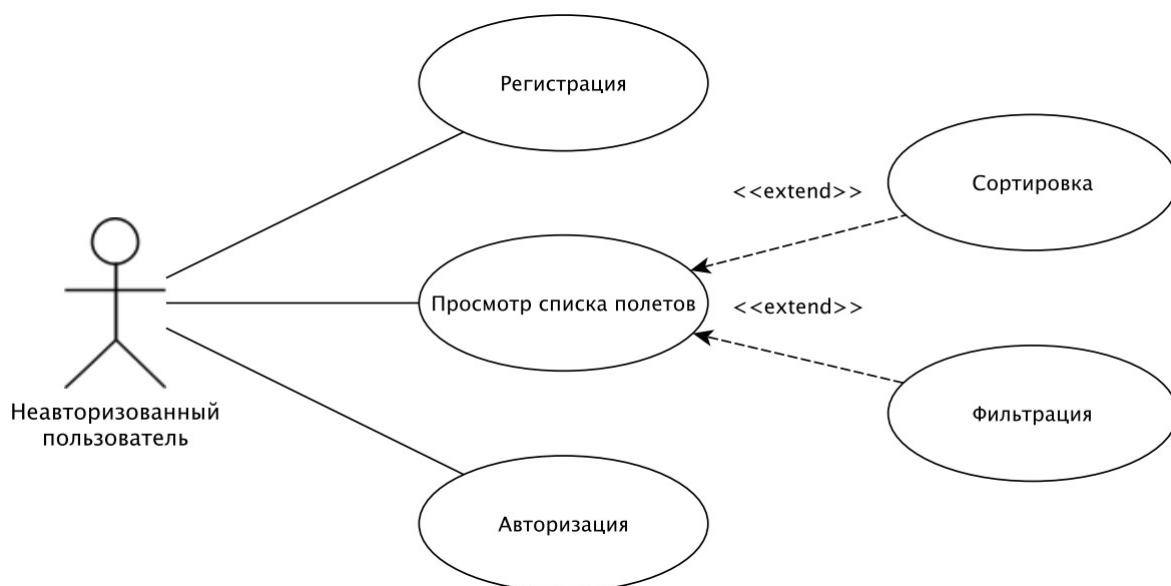


Рисунок 2.3 – Диаграмма прецедентов с точки зрения неавторизованного пользователя



Рисунок 2.4 – Диаграмма прецедентов с точки зрения авторизованного пользователя

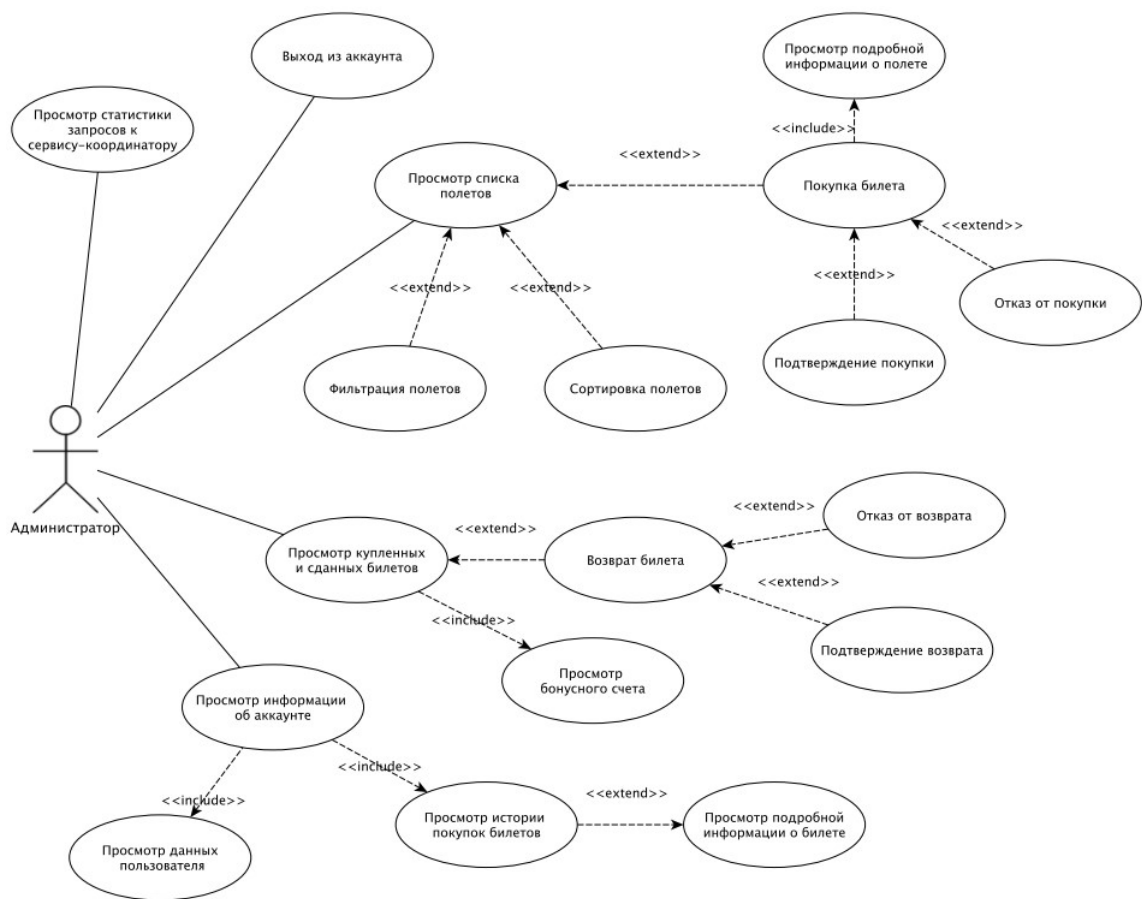


Рисунок 2.5 – Диаграмма прецедентов с точки зрения администратора

3 Технологическая часть

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, требования к ОС следующие.

- **Распространенность.** На рынке труда должно быть много специалистов, способных администрировать распределенную систему, работающую под управлением выбранной операционной системы.
- **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.
- **Наличие требуемого программного обеспечения.** Выбор операционной системы не должен ограничивать разработчиков в выборе программного обеспечения, библиотек.
- **Цена.**

Под данные требования лучше всего подходит ОС Linux, дистрибутив Ubuntu. **Ubuntu** [3] — это дистрибутив, использующий ядро Linux. Как и все дистрибутивы Linux, Ubuntu является ОС с открытым исходным кодом, бесплатным для использования. Поставляется как в клиентской (с графическим интерфейсом), так и в серверной (без графического интерфейса) версиями. Ubuntu поставляется с современными версиями ПО. Преимуществом Ubuntu являются низкие требования к квалификации системных администраторов. Однако Ubuntu менее стабильна в работе.

3.2. Выбор СУБД

В качестве СУБД была выбрана **PostgreSQL** [4], 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

Для разработки бэкенда существует множество языков программирования, каждый из которых имеет свои сильные и слабые стороны, для данного приложения был выбран **Python** [5] по следующим причинам:

1. Простота и скорость разработки: Python позволяет разработчикам писать меньше кода и быстрее реализовывать функциональность, благодаря удобному синтаксису и мощным фреймворкам (Django, Flask, FastAPI). Это особенно важно для стартапов и небольших проектов, где критично быстрое создание прототипов и внедрение изменений.
2. Широкая экосистема: В Python существует множество готовых библиотек для работы с базами данных, аутентификацией, кэшированием, оче-

редями, обработкой данных и другими задачами бэкенда. Это ускоряет разработку и снижает количество ручной работы.

3. Поддержка современных технологий: Python активно используется для задач, связанных с машинным обучением, обработкой данных и научными вычислениями. Это делает его выбором номер один для компаний, работающих с большими данными или развивающих искусственный интеллект.
4. Сообщество и поддержка: Python имеет одно из крупнейших сообществ разработчиков, что упрощает решение проблем, обновление знаний и получение поддержки.
5. Универсальность: Python может использоваться не только для разработки веб-приложений, но и для автоматизации, обработки данных и других областей, что делает его многофункциональным инструментом.

В качестве фреймворка для создания веб-приложений на Python был выбран **FastAPI** [6] по следующим причинам.

1. Производительность: FastAPI предлагает одну из самых высоких производительностей среди Python-фреймворков. Это особенно важно для приложений, требующих быстрого отклика при большом количестве запросов, таких как API и микросервисы.
2. Поддержка асинхронности: В отличие от Django, который частично поддерживает асинхронные операции, FastAPI изначально построен с учетом асинхронного программирования. Это делает его идеальным для приложений, где необходимо эффективно обрабатывать большое количество параллельных запросов (например, в реальном времени или при работе с внешними API).
3. Простота и автоматическая валидация: FastAPI использует аннотации типов Python для автоматической валидации данных и генерации документации, что значительно упрощает работу с API. Это улучшает качество кода и сокращает количество ошибок.

4. Генерация документации «из коробки»: FastAPI автоматически создает документацию API с использованием стандартов OpenAPI и Swagger. Это экономит время разработчиков и упрощает работу с клиентами и другими командами.
5. Гибкость и современность: FastAPI предлагает более гибкий и легкий подход к разработке по сравнению с Django, сохраняя простоту и читабельность кода, как в Flask. Он идеально подходит для создания быстрых, масштабируемых и легких приложений, особенно в микросервисной архитектуре.

3.4. Выбор фреймворка фронтенд разработки

Для разработки фронтенда был выбран фреймворк **React** [7] 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 «жестким» 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.

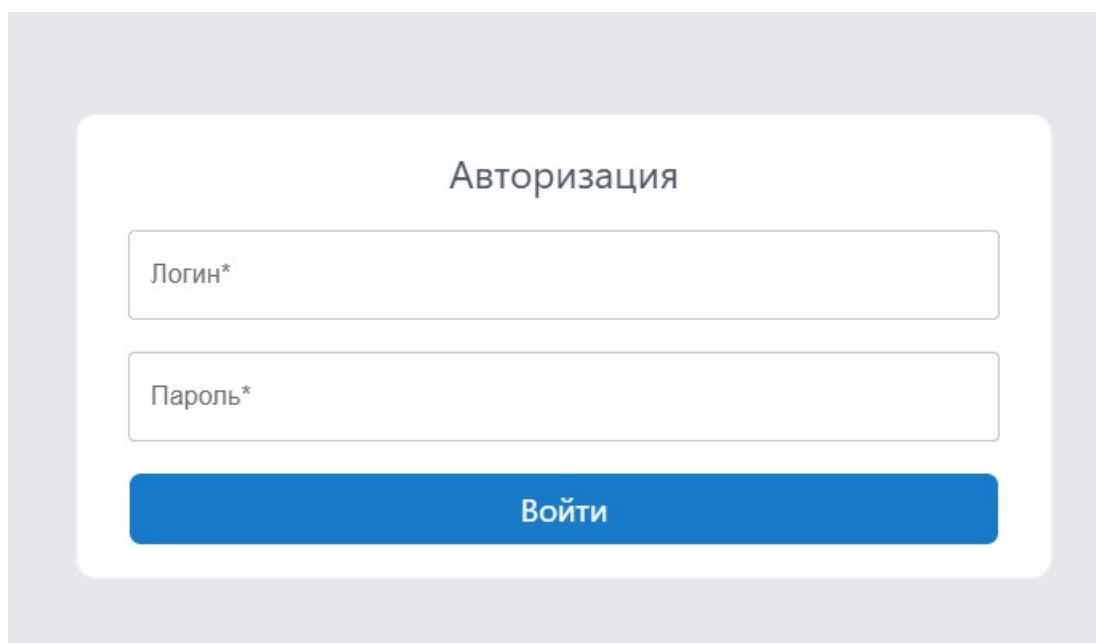
3.5. Высокоуровневый дизайн пользовательского интерфейса

Пользовательский интерфейс в разрабатываемой системе представляет собой web-интерфейс, доступ к которому осуществляется через браузер (тонкий клиент). Страница системы состоит из «шапки», бокового меню и основной части.

Обобщенно структуру страниц системы можно представить следующим образом:

- страница авторизации;
- страница регистрации;
- страница со списком полетов;
- страница со списком купленных и сданных билетах;
- страница с информацией о пользователе и историей покупок билетов;
- страница со статистикой сервиса-координатора.

Данные страницы представленные на рисунках 3.1-3.7 приведены примеры описанных страниц.

The image shows a web form titled "Авторизация" (Authorization) centered on a light gray background. The form itself is a white rounded rectangle. It contains two input fields: the first is labeled "Логин*" (Login*) and the second is labeled "Пароль*" (Password*). Below these fields is a solid blue button with the white text "Войти" (Login).

Авторизация

Логин*

Пароль*

Войти

Рисунок 3.1 – Страница авторизации

Регистрация

Рисунок 3.2 – Страница регистрации

Список полетов

Строк на странице: 10 1-10 из 25 < > >|

Рисунок 3.3 – Страница с полетами (пример сортировки и фильтрации)

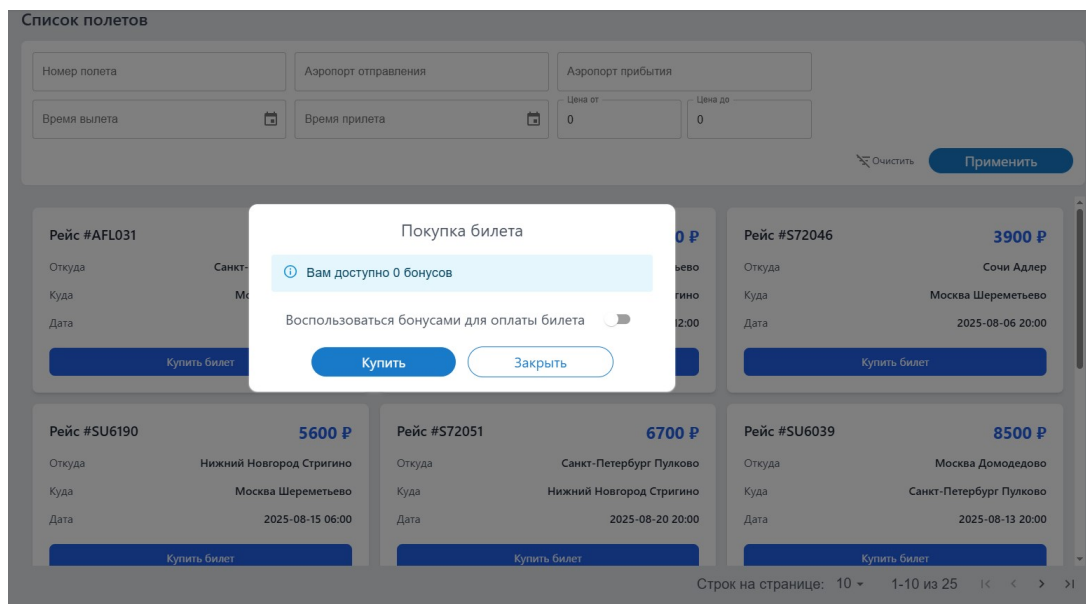


Рисунок 3.4 – Страница с полетами (пример покупки билета)

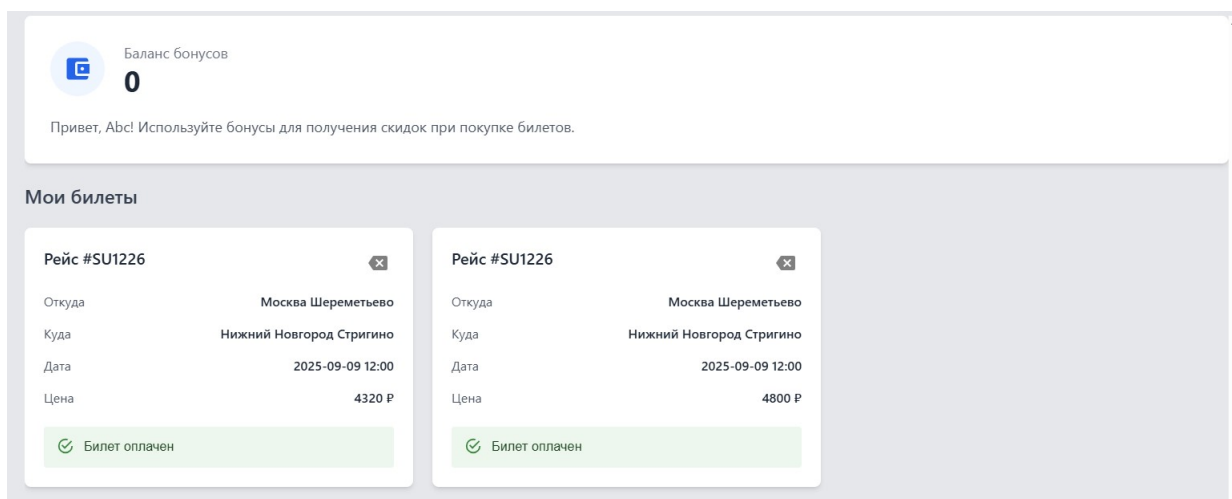


Рисунок 3.5 – Страница с купленными и сданными билетами

Информация о пользователе и история покупок билетов

Информация о пользователе

Логин

abc

Имя

Abc

Фамилия

Abc

Роль

USER

Почта

abc@gmail.com

Телефон

+79877655654

На Вашем счету 0 бонусов

Дата и время сдачи билета:

2025-06-02 18:30:41

Списано бонусов:

480

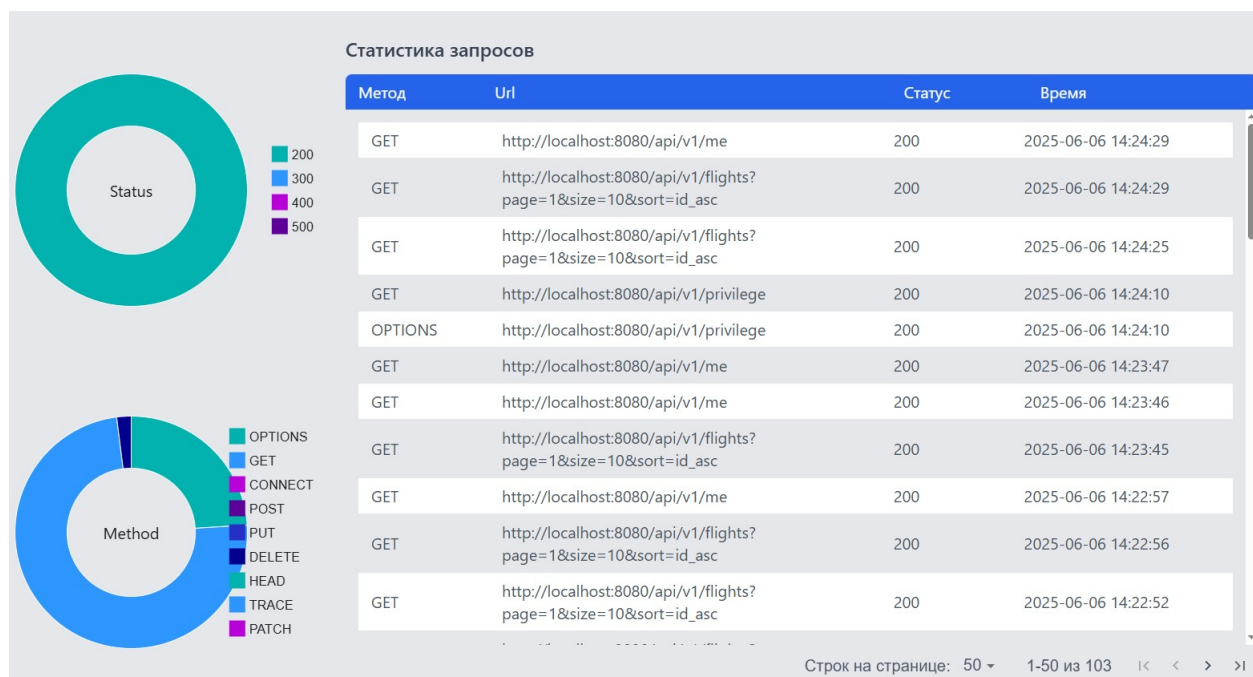
Дата и время покупки билета:

2025-06-02 18:30:34

Начислено бонусов:

480

Рисунок 3.6 – Страница с информацией о пользователе и историей покупок



ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы была разработана система для бронирования авиабилетов. В качестве фреймворка для бэкенда использовался FastAPI, для фронтенда был выбран React. В ходе выполнения работы была решена задача интеграции бэкенда и фронтенда через REST API, что обеспечило взаимодействие между клиентом и сервером. Для сбора статистики результатов запросов к сервису-координатору был использован брокер сообщений Kafka. Также была реализована авторизация и аутентификации пользователей с использованием JWT (JSON Web Token), что обеспечило защиту данных и доступ к сервису только авторизованным пользователям.

Таким образом, были решены следующие задачи, а цель достигнута.

- Описана разрабатываемая система.
- Сформулированы требования к системе бронирования авиабилетов.
- Спроектирована архитектура распределенной системы.
- Произведен выбор стека технологий для реализации системы.
- Реализована распределенная система бронирования авиабилетов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Kafka [Электронный ресурс] [Электронный ресурс]. — Режим доступа URL: <https://kafka.apache.org> (Дата обращения: 09.04.2024).
2. Zookeeper [Электронный ресурс] [Электронный ресурс]. — Режим доступа URL: <https://zookeeper.apache.org> (Дата обращения: 09.04.2024).
3. Ubuntu [Электронный ресурс] [Электронный ресурс]. — Режим доступа URL: <https://ubuntu.com> (Дата обращения: 09.04.2024).
4. PostgreSQL [Электронный ресурс] [Электронный ресурс]. — Режим доступа URL: <https://www.postgresql.org> (Дата обращения: 09.04.2024).
5. Python [Электронный ресурс] [Электронный ресурс]. — Режим доступа URL: <https://www.python.org> (Дата обращения: 09.04.2024).
6. FastAPI [Электронный ресурс] [Электронный ресурс]. — Режим доступа URL: <https://fastapi.tiangolo.com> (Дата обращения: 09.04.2024).
7. React [Электронный ресурс] [Электронный ресурс]. — Режим доступа URL: <https://react.dev> (Дата обращения: 09.04.2024).