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

ФАКУЛЬТЕТ _____«Информатика и система управления»_____

КАФЕДРА _____«Программное обеспечение ЭВМ и информационные технологии_____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Система для бронирования авиабилетов

Студент _____ИУ7-23М____ _____ _____**Алферова И.В.**___
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ _____**Ступников .А.А**
(Подпись, дата) (И.О.Фамилия)

2024 г.

ТУТ ТЗ

СОДЕРЖАНИЕ

Глоссарий

Термин	**Определение**
Front-end	Клиентская сторона пользовательского интерфейса к программно-аппаратной части сервиса
Back-end	Программно-аппаратная часть сервиса
Валидация	Проверка данных на соответствие заданным условиям и ограничениям
REST	архитектурный стиль взаимодействия компонентов распределённого приложения в сети, основанный на протоколе HTTP
Веб-интерфейс	Веб-страница или совокупность веб-страниц, предоставляющая пользовательский интерфейс для взаимодействия с сервисом или устройством посредством протокола HTTP и веб-браузера
Сервис-ориентированная архитектура	Модульный подход к разработке программного обеспечения, основанный на использовании распределённых, слабо связанных заменяемых компонентов, оснащённых стандартизированными интерфейсами для взаимодействия по стандартизированным протоколам
JSON	JSON (JavaScript Object Notation) - это формат обмена данными, который представляет собой легкий и удобный способ передачи структурированных данных между приложениями. JSON основан на синтаксисе языка JavaScript и использует пары "ключ-значение" для описания объектов и массивов. Он поддерживает различные типы данных, такие как строки, числа, логические значения, массивы и вложенные объекты.
Kafka	Apache **Kafka** — распределённый программный брокер сообщений с открытым исходным кодом, разрабатываемый в

	рамках фонда Apache на языках Java и Scala.
Docker	Это платформа, которая предназначена для разработки, развёртывания и запуска приложений в контейнерах.
Авиарейс	Путь пассажирского самолета, предусмотренный расписанием в одном направлении. В целях установления четкого различия между многочисленными рейсами, выполняемыми авиакомпанией, каждому самостоятельному рейсу присваивается номер, под которым он обозначается во всех справочных материалах и расписаниях.

ВВЕДЕНИЕ

Высокая конкуренция между существующими системами по заказу и бронированию авиабилетов, появление ряда сайтов-агентств в сети Интернет [1], предлагающих свои услуги по реализации авиабилетов, потребовали пересмотра в концепциях создания и развития новых систем. Это влечет за собой создание новых технологий, основанных на архитектуре «клиент-сервер», и переход от робастных к адаптивным режимам работы серверных узлов таких систем.

В основу новой концепции создания систем заказа и бронирования авиабилетов должны быть положены следующие основные принципы: дружелюбный интерфейс пользователя при работе с системой, высокая технологичность при создании таких систем, основанная на современных информационных технологиях, использование высокоскоростных выделенных каналов и эффективная работа серверных узлов таких систем.

Целью данной работы разработка распределенной системы, позволяющей производить бронирование авиабилетов. Для достижения поставленной цели необходимо решить следующие задачи:

- сформулировать основные требования к системе;
- описать требования к подсистемам;
- описать архитектуру системы, а также сценарии ее функционирования;
- выбрать и обосновать инструменты для разработки ПО;
- реализовать программный продукт.

Аналитический раздел

В данном разделе будет произведена постановка задачи, описана разрабатываемая система, сформулированы общие требования к системе, а также требования к функциональным характеристикам, приведены функциональные требования к системе с точки зрения пользователя, описаны входные и выходные данные системы, сформулированы требования к программной реализации и функциональные требования к подсистемам. Также будет приведена топология системы.

Постановка задачи

Разрабатываемая система должна предоставлять пользователям возможность бронирования авиабилетов. В зависимости от количества сделанных ранее заказов система должна рассчитывать скидку на новые согласно условиям программы лояльности. Должен быть предусмотрен просмотр статистики входящих запросов от лица администратора.

Описание системы

Разрабатываемый сервис должен представлять собой распределённую систему для бронирования авиабилетов. Если клиент хочет оформить бронь, ему необходимо зарегистрироваться, указав информацию: фамилия, имя, электронная почта. В случае, если зарегистрированному ранее пользователю нужно отменить заказ, получить информацию о его бронированиях или статусе в программе лояльности, ему нужно авторизоваться. Для неавторизованных пользователей доступен только просмотр общей информации.

Общие требования к системе

1) Разрабатываемое программное обеспечение должно обеспечивать функционирование системы в режиме 24/7/365 со среднегодовым временем

доступности не менее 99.9%. Допустимое время, в течение которого система не доступна, за год должна составлять 24*365*0.001=8.76 часа.

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

3) Вся приходящая на сервер информация должна валидироваться.

4) Каждый узел должен автоматически восстанавливаться после сбоя.

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 5 секунд.

3) Портал должен обеспечивать возможность запуска в современных браузерах: не менее 85% пользователей Интернета должны иметь возможность пользоваться порталом без какой-либо деградации функционала.

Функциональные требования к системе с точки зрения пользователя

Портал должен обеспечивать реализацию следующих функций:

1. Система должна обеспечивать регистрацию пользователей с валидацией вводимых данных.
2. Система должна обеспечивать аутентификацию пользователей.
3. Система должна обеспечивать разделение пользователей на три роли:
   - пользователь — неавторизованный пользователь;
   - клиент — авторизованный пользователь;
   - администратор.
4. Система должна предоставлять **пользователю** следующие функции:
   - просмотр информации об рейсах: аэропорт прибытия, аэропорт

отправления, даты полета, стоимость;

- регистрация;
- авторизация.

5. Система должна предоставлять **клиенту** все функции пользователя, а также следующие функции:
   - просмотр информации об рейсах: аэропорт прибытия, аэропорт отправления, даты полета, стоимость ;
   - бронирование выбранного рейса
   - отмена брони рейса;
6. Система должна предоставлять **администратору** все функции клиента, а также следующие функции:
   - получение информации об условиях программы лояльности;
   - авторизация в системе;
   - регистрация нового пользователя;

Описание входных данных

В таблице 1 представлено описание данных, поступающих системе на вход.

Таблица 1: Описание входных данных

Сущность	Входные данные
Клиент / Администратор	1. 1. фамилия, имя и отчество не более 256 символов каждое 2. поле; 3. 2. дата рождения в формате д/м/гггг; 4. 3. логин не менее 10 символов и не более 128; 5. 4. пароль не менее 8 символов и не более 128, как минимум 6. одна заглавная и одна строчная буква, только латинские буквы, без пробелов, как минимум одна цифра; 5. номер телефона;

	6. электронная почта;
Рейс	Номер рейса Дата и время рейса Аэропорт отправления Аэропорт прибытия Идентификатор аэропорта отправления Идентификатор аэропорта прибытия Стоимость билета
Билет	Идентификатор Уникальный идентификатор билета Имя пользователя Номер рейса Стоимость билета
Аэропорт	Идентификатор (в СУБД) Название аэропорта Город Страна

Выходные данные

Выходными параметрами системы являются веб-страницы. Они должны содержать следующую информацию:

- страницу регистрации и авторизации пользователя;
- информацию о рейсах ;
- окно бронирования;
- окно для администратора, позволяющее смотреть статистику.

Топология системы

Разрабатываемая система состоит одного веб-фронтэнд приложения и 5 подсистем:

- сервис аккаунтов;
- сервис рейсов;
- сервис лояльности;
- сервис билетов;
- сервис статистики.

На рисунке 1.1 приведена топология разрабатываемой системы.

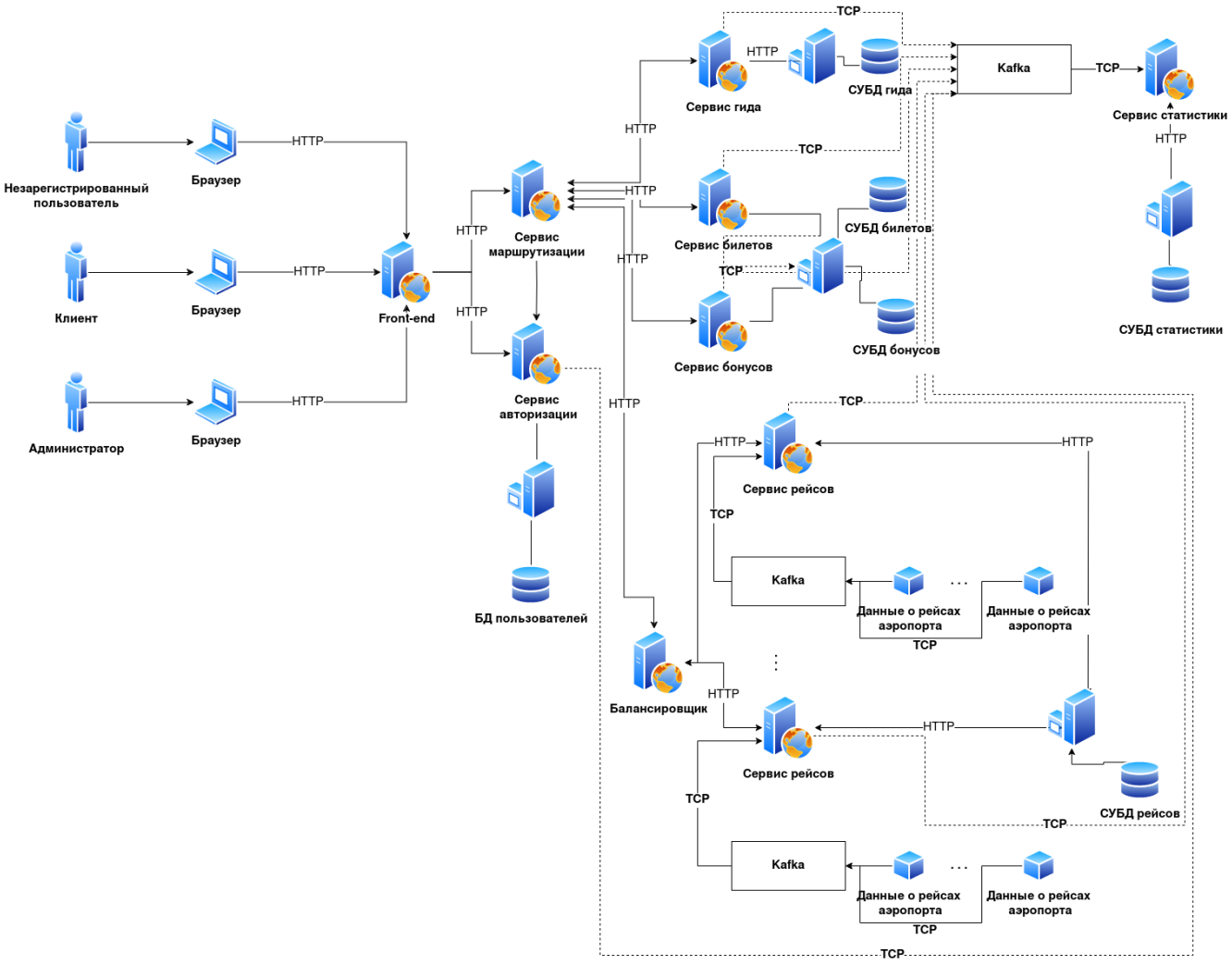

Рисунок 1.1 — Топология системы.

Фронтэнд-приложение принимает запросы от пользователей по протоколу HTTP и анализирует их. На основе проведенного анализа выполняет запросы к микросервисам бэкенда, агрегирует ответы и отсылает их пользователю.

Сервис-координатор — единая точка входа и межсервисной коммуникации.

Сервис-регистрации и авторизации отвечает за:

1. возможность регистрации нового клиента;
2. аутентификацию пользователя (клиента и администратора);
3. авторизацию пользователя;
4. выход из сессии.

Сервис билетов реализует следующие функции:

1. Просмотр сведений о конкретном билете;
2. Добавление нового билета;
3. Изменение существующего билета;
4. Удаление существующего билета;
5. Получение списка всех билетов для указанного пользователя;

Сервис рейсов реализует функции:

1. Получение значений всех параметров выбранного рейса;
2. Удаление всех значений параметров для выбранного рейса.

Сервис лояльности отвечает за ведение статистики по количеству бронирований всех клиентов, на основе которой для каждого пользователя в индивидуальном порядке предоставляется скидка на будущие заказы.

6.1. Требования к программной реализации

1. Требуется реализовать СОА (сервис-ориентированную архитектуру) [2] для реализации системы.
2. К реализации фронтэнд приложений должны быть предъявлены следующие требования:
    1. Фронтэнд должен является SPA и возвращаться пользователю с

сервера в виде одного документа.

2. Фронтэнд должен является посредником между пользователем и бекэндом. Он должен конструировать запросы, отправлять их, и обрабатывать ответы, выдавая их в виде HTML страниц.

3. К реализации бекэнд-приложений должны быть предъявлены следующие требования:
    1. Прием и возврат данных должен осуществляться в формате JSON [3] по протоколу HTTP (необходимо придерживать RESTful [4]).
    2. Доступ к базам данных должен осуществляться только из подсистем, работающих напрямую с данными ее таблиц.
    3. Выделить Gateway Service как единую точку входа и межсервисной коммуникации. В системе не должно осуществляться горизонтальных запросов.
    4. При недоступности систем портала должна осуществляться деградация функционала или выдача пользователю сообщения об ошибке.
    5. Валидацию входных данных необходимо проводить и на стороне пользователя, и на стороне фронтэнда. Микросервисы бекэнда не должны валидировать входные данные, поскольку пользователь не может к ним обращаться напрямую, они должны получать уже отфильтрованные входные данные.
4. Для запросов, выполняющих обновление данных на нескольких узлах распределенной системы, в случае недоступности одной из систем, необходимо выполнять полный откат транзакции.
5. Приложение должно поддерживать возможность горизонтального и вертикального масштабирования за счет увеличения количества функционирующих узлов и совершенствования технологий реализации компонентов и всей архитектуры системы.
6. Код необходимо хранить на Github.
7. Gateway Service должен запускаться на порту 8080, остальные сервисы

запускать на портах 8030, 8040, 8050, 8060, 8070. Фронтэнд запускается на порту 3000.

8. Каждый сервис должен быть завернут в docker.

Функциональные требования к подсистемам

В разрабатываемой системе существуют следующие подсистемы: фронтэнд, бекэнд-координатор, бекэнд регистрации и авторизации, бекэнд бронирования, бекэнд оплаты, бекэнд лояльности.

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

- он 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 [5].

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

- 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 её состояние;
- выполнять проверку существования клиента, также регистрацию и аутентификацию пользователей;

- получение списка всех рейсов;
- получение информации и обновление данных о зарегистрированном пользователе;
- оформление и отмена созданного ранее бронирования;
- получение данных о бронированиях пользователя;

Сервис-регистрации и авторизации должен реализовывать следующие функциональные возможности:

- принимать и возвращать данные в формате JSON по протоколу HTTP;
- возможность регистрации нового клиента и обновление данных уже существующего;
- проверка существования клиента;
- обеспечение авторизации пользователя через аккаунт как в системе.

Сервис лояльности должен реализовывать представленные такие функциональные возможности, как:

- получение и отправка ответов на запросы в формате JSON по протоколу HTTP;
- получение величины скидки по конкретному пользователю;
- получение детальной информации о конкретном участнике программы лояльности;
- обновление числа бронирований и статуса по программе лояльности (предусмотреть, как повышение, так и понижение в случае отмены бронирования);
- внесение изменений в размер скидки по конкретному пользователю.

Вывод

В данном разделе была произведена постановка задачи, описана разрабатываемая система, сформулированы общие требования к системе, а также требования к функциональным характеристикам, приведены функциональные требования к системе с точки зрения пользователя, описаны входные и выходные данные системы, сформулированы требования к

программной реализации и функциональные требования к подсистемам. Также была приведена топология разрабатываемой системы.

Конструкторский раздел

В данном разделе будет приведен концептуальный дизайн системы, сценарии функционирования, диаграммы прецедентов, спецификация сценариев и классов, структура базы данных.

Концептуальный дизайн

На рисунке 2.1 представлена IDEF0-диаграмма системы верхнего уровня.

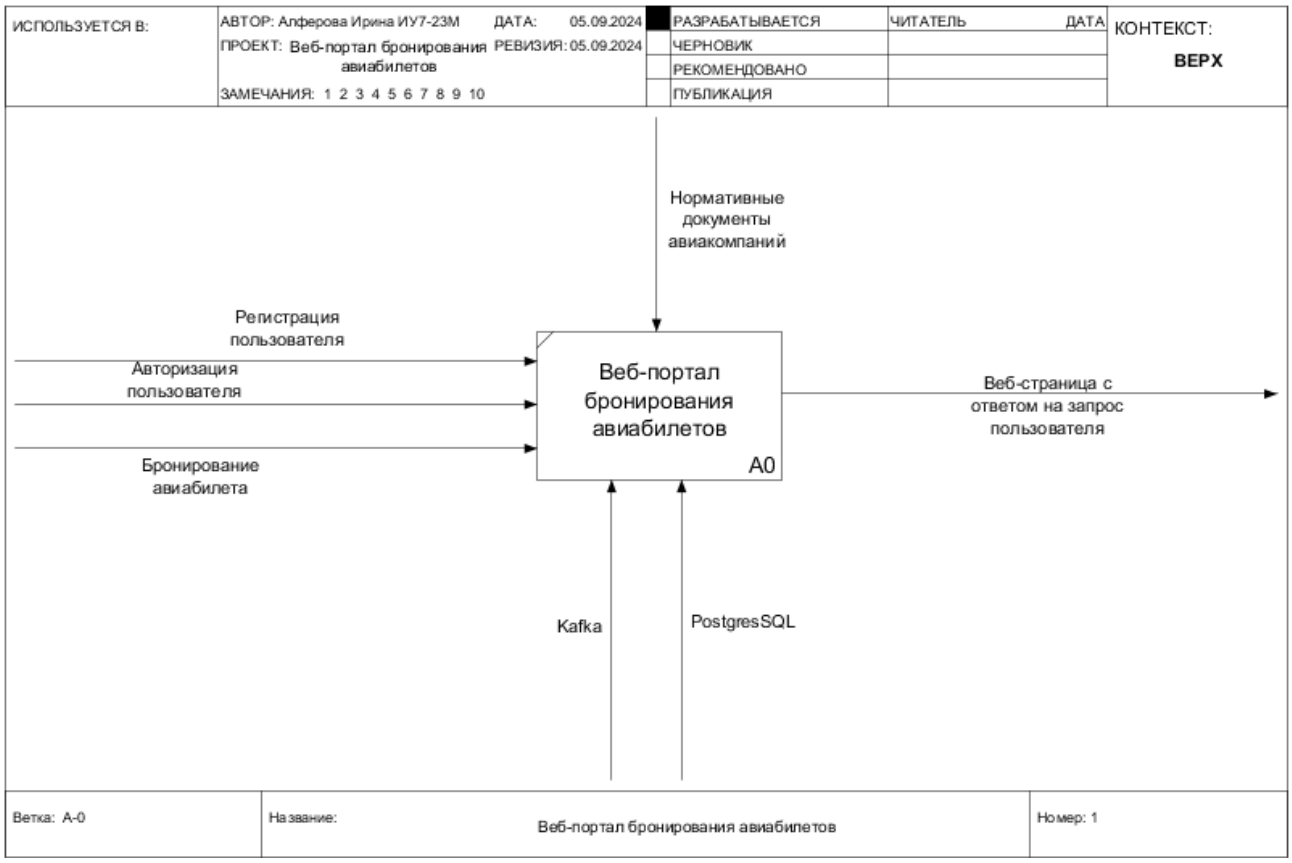

Рис. 2.1 – IDEF0-диаграмма системы для бронирования рейсов.

На рисунке 2.2 представлена IDEF0-диаграмма системы первого уровня, описывающая процесс обработки запроса от пользователя.

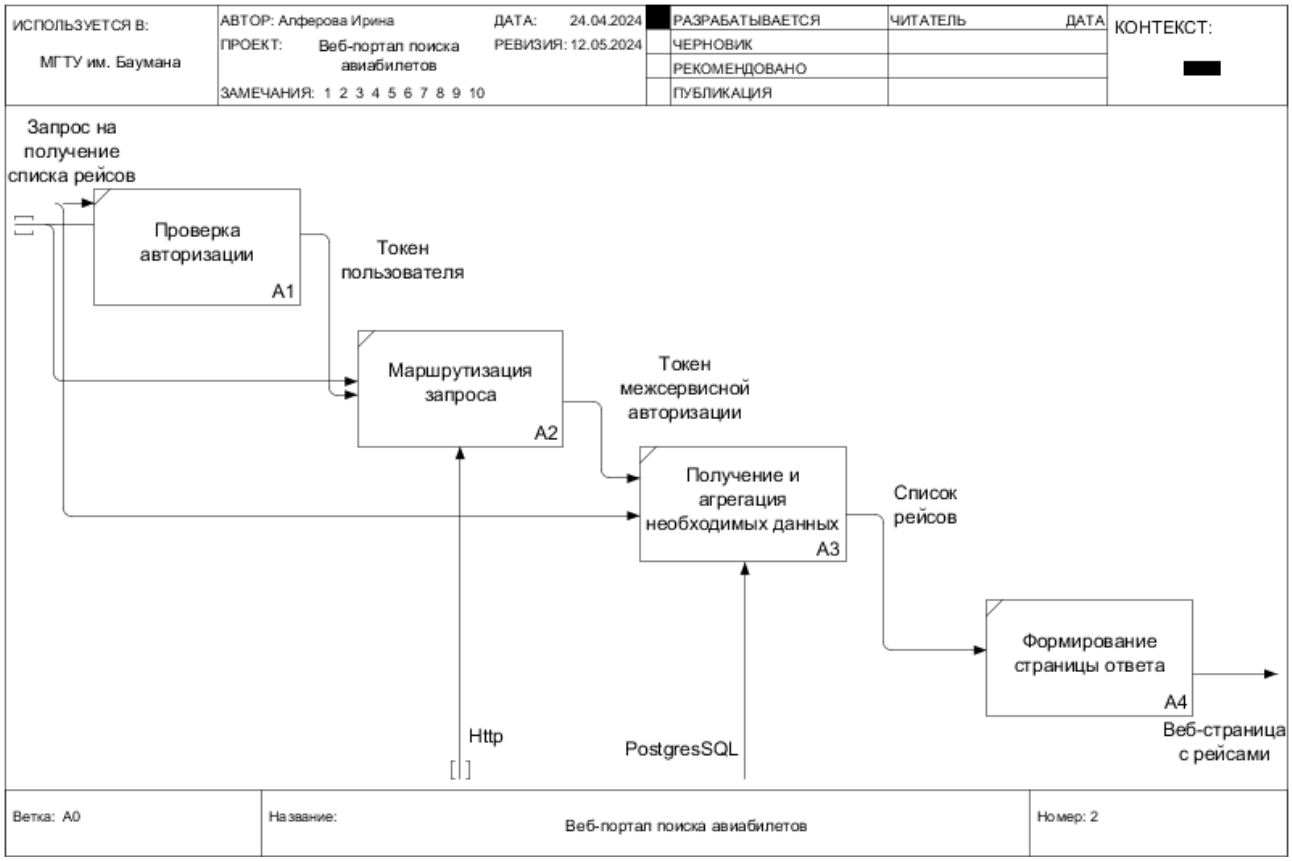

Рис. 2.2 – IDEF0-диаграмма процесса выполнения запроса в системе.

Сценарии функционирования системы

Регистрация нового пользователя

Действие может быть выполнено только пользователем с ролью «Администратор системы» и состоит из следующих шагов:

1. Ввод данных пользователя (логин, пароль, роль) в графический интерфейс; в корректности введенных данных происходит переход к следующему шагу, иначе система предлагает повторить ввод;
2. Отправка данных и токена авторизованного пользователя сервису учетных данных пользователей;
3. Валидация сервисом учетных данных токена и полученных данных пользователя; в случае успеха данные добавляются в БД пользователей, и

возвращается сообщение об успешном выполнении операции, иначе возвращается сообщение об ошибке.

Авторизация зарегистрированного пользователя

Действие может быть выполнено любым неавторизованным пользователем и включает следующие шаги:

1. Ввод логина и пароля пользователем в графический интерфейс; в случае корректности введенных данных происходит переход к следующему шагу, иначе система предлагает повторить ввод;
2. Отправка данных сервису учетных данных пользователей;
3. Валидация сервисом учетных данных полученных логина и пароля; в случае наличия в БД пользователей записи с такими данными сервис возвращает авторотационный токен, иначе возвращается сообщение об ошибке.

Удаление зарегистрированного пользователя

Действие может быть выполнено только пользователем с ролью «Администратор системы» и состоит из следующих шагов:

1. Выбор в графическом интерфейсе пользователя для удаления (невозможно удалить авторизованного в данный момент пользователя); в случае корректного выбора происходит переход к следующему шагу, иначе система предлагает повторить ввод;
2. Отправка токена авторизованного пользователя и идентификатора пользователя для удаления сервису учетных данных пользователей;
3. Валидация сервисом учетных данных полученного токена пользователя и идентификатора удаляемого пользователя; в случае успеха происходит удаление из БД пользователя с указанным идентификатором и возврат сообщения об успешном выполнении операции, иначе возвращается сообщение об ошибке.

.

Отмена купленного билета

Действие может быть выполнено только пользователем с ролью “Клиент” и состоит из следующих шагов:

1. Выбор билета для отмены в графическом интерфейсе;
2. Отправка токена авторизованного пользователя и идентификатора отменяемого билета сервису маршрутизации запросов;
3. Отправка сервисом маршрутизации сервису учетных данных пользователей запроса на валидацию пользовательского токена; в случае успеха происходит переход к следующему шагу, иначе возвращается сообщение об ошибке;
4. Отправка токена межсервисной авторизации и идентификатора отменяемого билета сервису билетов;
5. Валидация сервисом билетов токена межсервисной авторизации и идентификатора отменяемого билета; в случае успеха происходит переход к следующему шагу, иначе возвращается сообщение об ошибке;
6. Если при бронировании билета были потрачены бонусные рубли: Отправка токена межсервисной авторизации и идентификаторов владельца отменяемого билета сервису данных бонусной программы, иначе пропустить следующий пункт;
7. Валидация сервисом бонусной программы токена межсервисной авторизации и идентификатора владельца отменяемого билета; в случае успеха происходит возврат на баланс пользователя использованных при покупке бонусных рублей, иначе сервису маршрутизации возвращается сообщение об успехе, но запрос на возврат бонусных рублей помещается в очередь;
8. Возврат сервисом маршрутизации ответа от сервиса билетов.

Диаграммы прецедентов

В системе выделены три роли — пользователь (неавторизованный пользователь), клиент (авторизованный пользователь), администратор. Диаграмма прецедентов для пользователя и клиента объединены в одну. На рисунке 2.3 представлена диаграмма прецедентов системы для авторизованного пользователя.

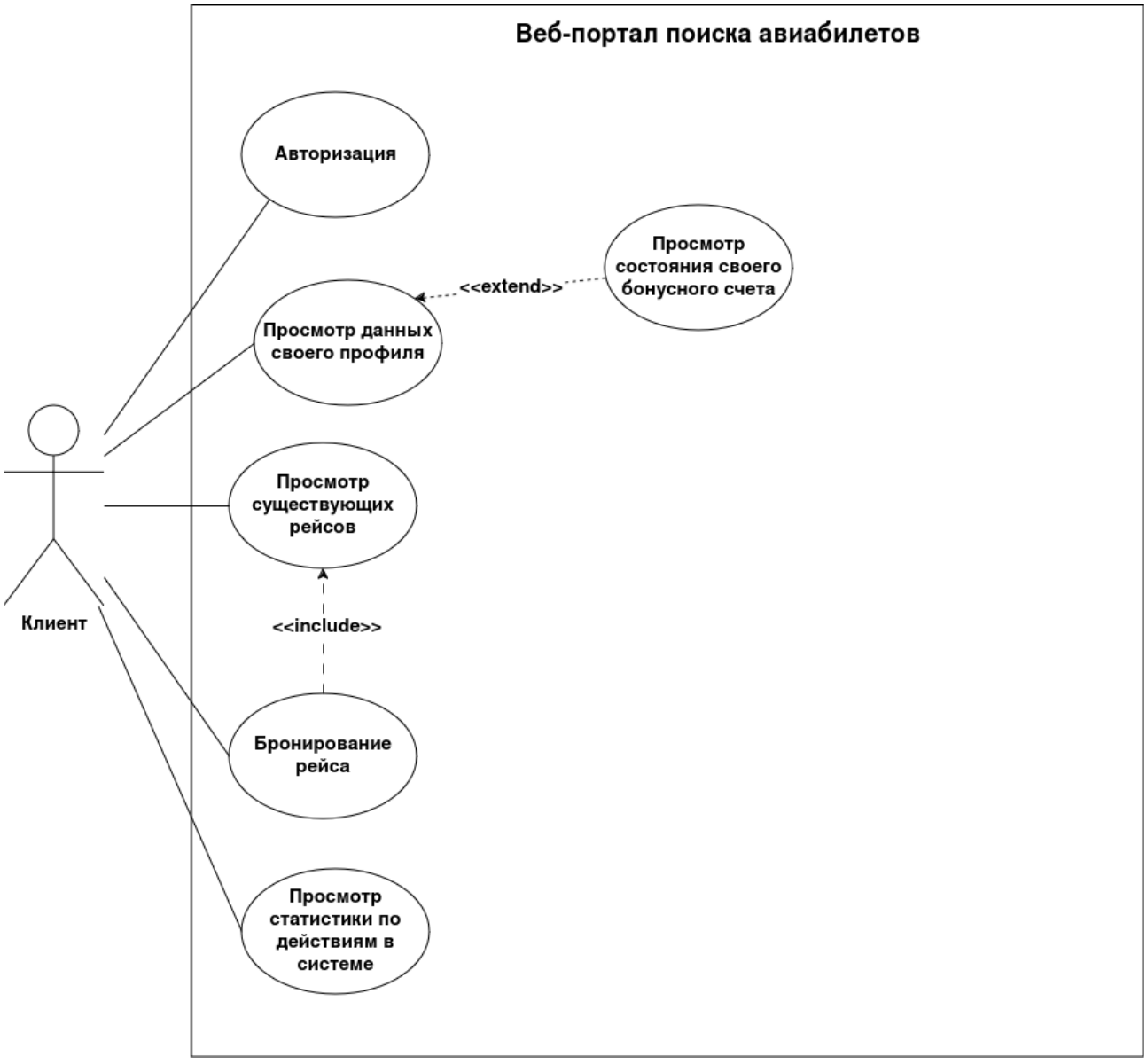

Рис. 2.3 – Диаграмма прецедентов для пользователя.

Спецификации классов

На приведенном ниже рисунке 2.4 представлена диаграмма классов .

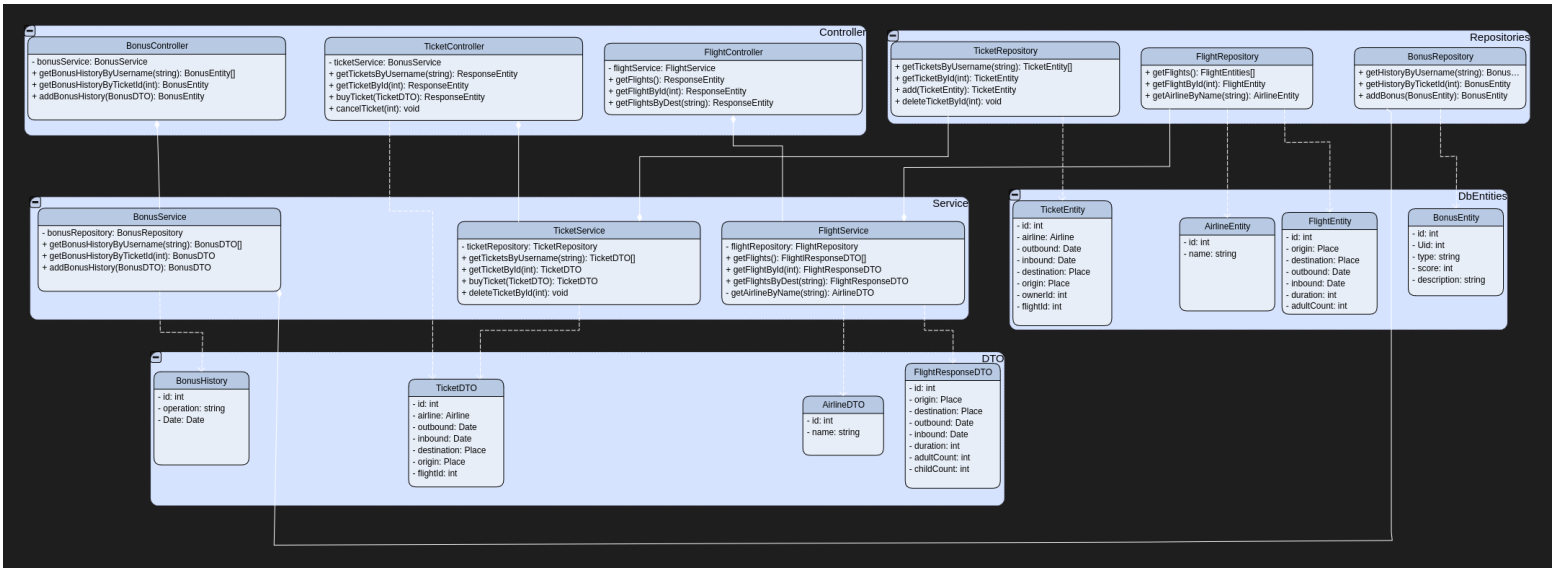

Рис. 2.4 – Диаграмма классов.

В таблицах 2-4 описаны атрибуты классов уровня сущностей БД.

Таблица 2: Описание атрибутов сущности билета

Имя	Тип	Описание
Id	integer	Идентификатор (в СУБД)
TicketUid	string	Уникальный идентификатор билета
Username	string	Имя пользователя
FlightNumber	string	Номер рейса
Price	integer	Стоимость билета
Status	string	Статус билета

Таблица 3: Описание атрибутов сущности аэропорта

Имя	Тип	Описание
Id	int	Идентификатор (в СУБД)
Name	string	Название аэропорта
City	string	Город
Country	string	Страна

Таблица 4: Описание атрибутов сущности рейса

Имя	Тип	Описание
Id	int	Идентификатор (в СУБД)
FlightNumber	string	Номер рейса
Datetime	string	Дата и время рейса
FromAirport	Airport	Аэропорт отправления
ToAirport	Airport	Аэропорт прибытия
FromAirportID	int	Идентификатор аэропорта отправления
ToAirportID	int	Идентификатор аэропорта прибытия
Price	int	Стоимость билета

Классы Flight, Airport, FlightResponse, PaginationResponse представляют объекты для передачи данных между классами бизнес-логики. Атрибутивный состав Flight, Airport, FlightResponse, PaginationResponse аналогичен составу ассоциированных с ними сущностей базы данных. Атрибуты вспомогательного класса PaginationResponse приведены ниже в таблице 5.

Таблица 5: Описание атрибутов класса PaginationResponse

Имя	Тип	Описание
Page	int	Номер страницы
PageSize	int	Размер страницы
TotalElements	int	Общее количество элементов
Items	[]FlightResponse	Список объектов FlightResponse

Repository отвечают за взаимодействие микросервиса с базой данных. Методы каждого приведены в таблицах 6-7.

Таблица 6: Описание методов класса FlightRepository

Методы	
Название	**Описание**
Create(Flight)	param: flight [Flight – in] — экземпляр рейса Добавить запись о рейсе в БД
Find(string): Flight	param: flightUid [string – in] — идентификатор рейса Найти запись о рейсе по идентификатору рейса
FindFlights(int, int): Flight[]	Param: page [int – in] — страница для пагинации, size [int – in] — количество элементов на странице Получить все записи о рейсах из БД
Update(Flight)	param: flight [Flight – in] — экземпляр рейса Обновить информацию о рейсе в БД
Delete(string)	Param: flightUid [string – in] — идентификатор рейса Удалить запись о рейсе из БД

Таблица 7: Описание методов класса TicketRepository

Методы	
Название	**Описание**
Create(Ticket)	param: ticket [Ticket – in] — экземпляр брони Добавить запись о билете в БД
Find(string, string): Ticket	param: username [string – in] — имя пользователя, ticketUid [string – in] — идентификатор билета Найти запись о конкретном билете пользователя
Find(string, int, int): Ticket[]	Param: username [string – in] — имя пользователя, page [int – in] — страница для пагинации, size [int – in] — количество элементов на странице Получить все записи билетов пользователя из БД
Delete(string)	Param: ticketUid [string – in] — идентификатор билета Отменить билет

Структура базы данных

На рисунке 2.5 представлена схема структуры БД, используемой в разрабатываемом ПО.

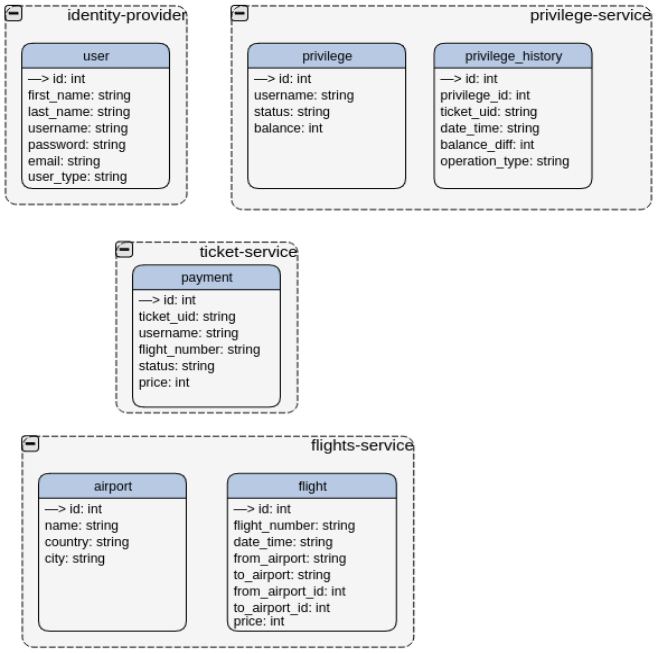

Рис. 2.5 – Схема БД.

Технологический раздел

В данном разделе будет обоснован выбор операционной системы, СУБД, языка разработки компонентов, а также приведен дизайн пользовательского интерфейса.

Выбор операционной системы

Согласно требованиям технического задания, разрабатываемый портал должен обладать высокой доступностью, работать на типичных архитектурах ЭВМ (Intel x86, Intel x64, ARM), а также 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 мирового уровня, как Google.com, Amazon.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 Ubuntu [6]. Ubuntu — 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.

Выбор СУБД

В соответствии с техническим заданием разработка бекэнда предусматривает следующие требования:

- **Безопасность хранения данных**. Несанкционированный доступ к данным клиентам должен быть невозможен.
- **Транзакционность**. Должен соблюдаться принцип «ACID» (Atomicity— Атомарность, Consistency — Согласованность, Isolation — Изолированность, Durability — Надежность). Атомарность гарантирует, что транзакция не может быть зафиксирована частично. Согласованность — что успешное завершение транзакции оставит систему в согласованном состоянии. Изолированность — что параллельно выполняемые транзакции не будут влиять друг на друга. Надежность — что успешно завершенная транзакция будет зафиксирована, а в случае сбоя, после восстановления системы, результаты транзакции не будут утеряны.
- **Масштабируемость**. Выбранная СУБД должна поддерживать репликацию, шардирование.

PostgreSQL [7] – реляционная система управления базами данных. Она является некоммерческим ПО с открытым исходным кодом. Для работы с этой СУБД существуют библиотеки для таких распространенных языков программирования, как Python, Ruby, Perl, PHP, C, C++, Java, C#, Go. Она работает под управлением многих операционных систем: Linux, MacOS, Windows, FreeBSD, Solaris и др. По сравнению с MySQL система PostgreSQL лучше работает с репликацией, так как в ней существует журнал (средство восстановления системы в случае сбоя) физической модификации страниц. PostgreSQL осуществляет асинхронную репликацию типа «ведущий — ведомый».

Выбор СУБД PostgreSQL для хранения данных разрабатываемой системы обеспечит надежность, безопасность и масштабируемость.

Выбор языка разработки компонентов

Исходя из приведенных требований к системе, можно выявить следующие требования к языку программирования.

- **Наличие разнообразных библиотек**. Использование готовых библиотек ускоряет разработку программного обеспечения. Также важно, что благодаря использованию распространенных оттестированных библиотек снижается вероятность ошибки. Это повышает надежность программного обеспечения.
- **Совместимость с выбранными ранее технологиями**. Выбранный язык должен уметь взаимодействовать с ОС Linux, СУБД PostgreSQL.
- **Высокая скорость разработки**. На ранних этапах разработки портала технические требования часто меняются. Язык программирования должен позволять как можно быстрее вносить изменения в коды программ.

В качестве языка программирования, соответствующему приведенным выше требованиям, был взят язык программирования Golang [8], разработанный компанией Google, ориентированный на создание микросервисов. Среди его плюсов можно выделить следующие:

- **Простота** – чёткие синтаксические правила и понятная семантика.
- **Объектно-ориентированный подход в формате «утиной» типизации**.
- **Безопасность**. Обойти или взломать механизмы защиты крайне сложно.
- **Производительность**. Язык программирования Golang сочетает в себе скорость языка Си с синтаксической простотой языка Python.
- **Надёжность**. Компилятор способен выявить ошибки ещё до выполнения

кода, то есть на ранних стадиях.

Дизайн пользовательского интерфейса

Пользовательский интерфейс в разрабатываемой системе представляет собой Web-интерфейс, доступ к которому осуществляется через браузер (тонкий клиент). На рисунках 3.1-3.6 приведен разработанный пользовательский интерфейс.

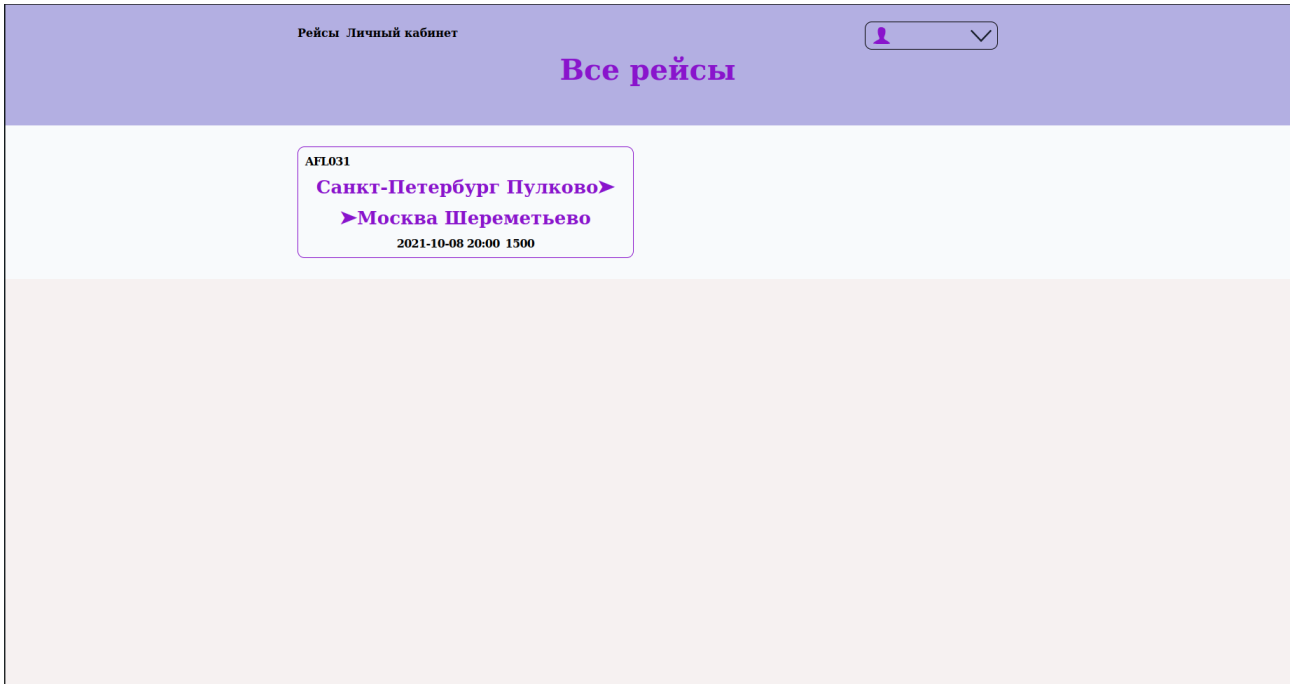

Рис. 3.1 – Главная страница.

Рис. 3.2 – Страница входа.

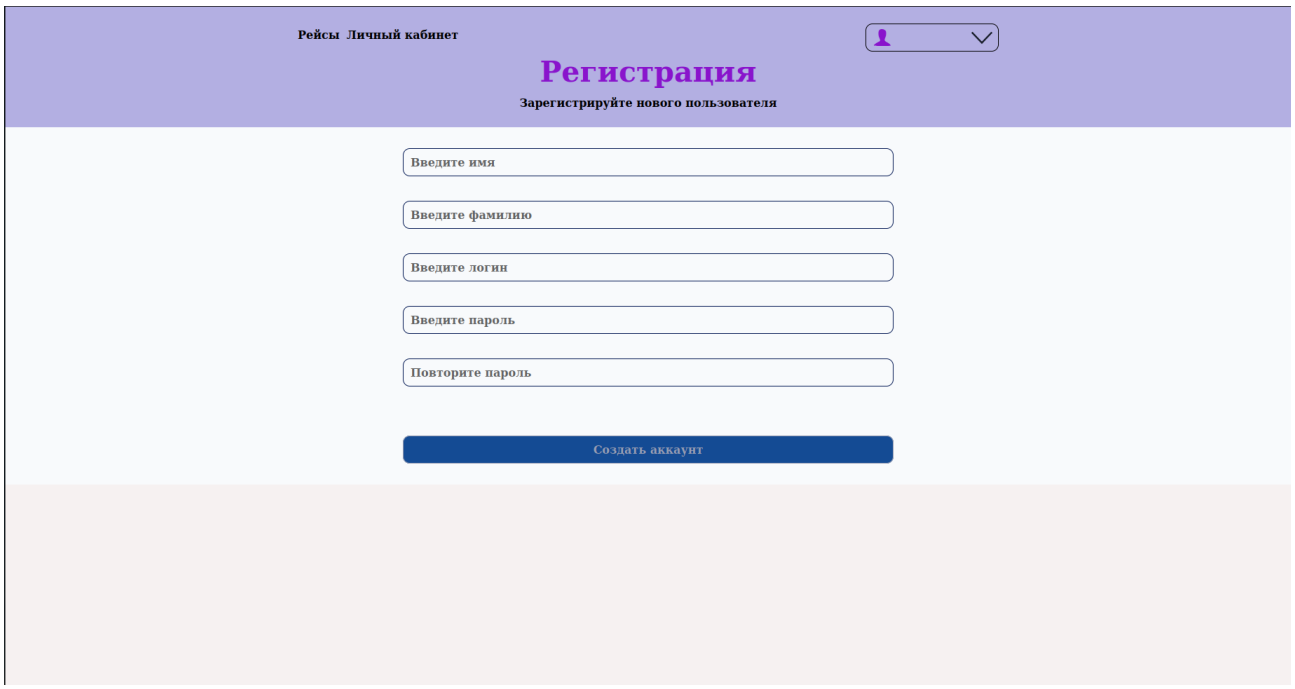

Рис. 3.3 – Страница регистрации.

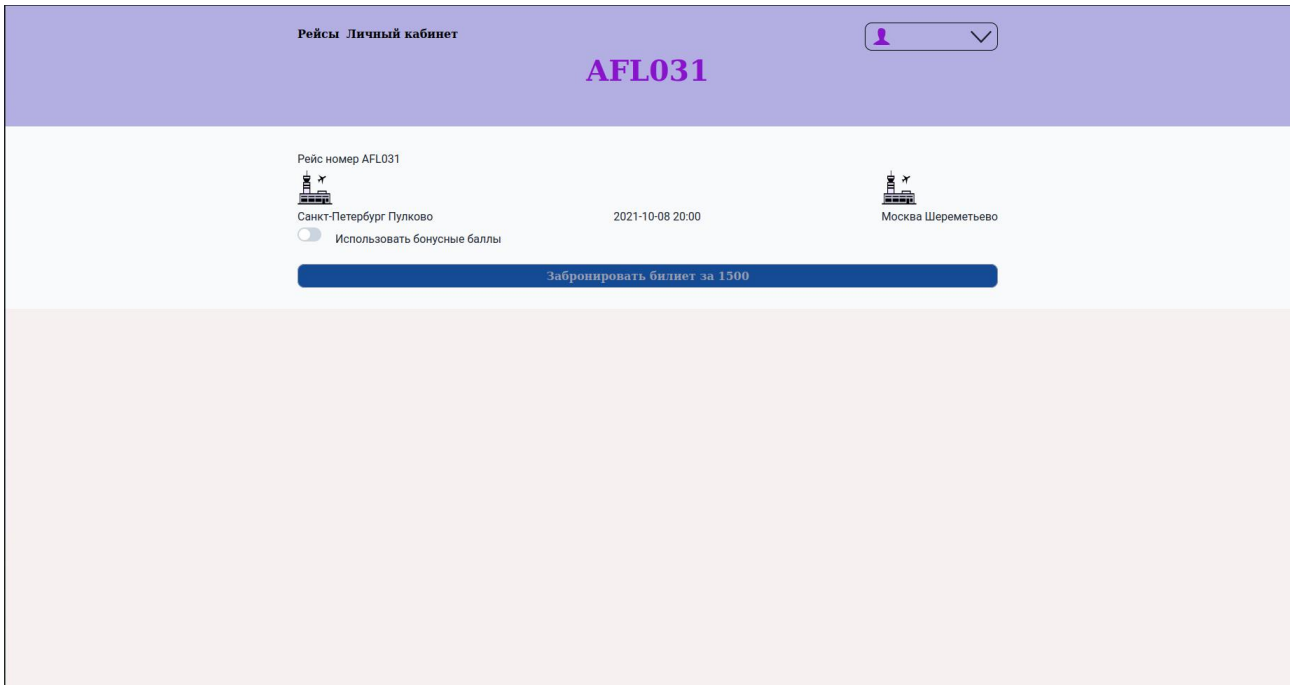

Рис. 3.4 – Окно бронирования рейса.

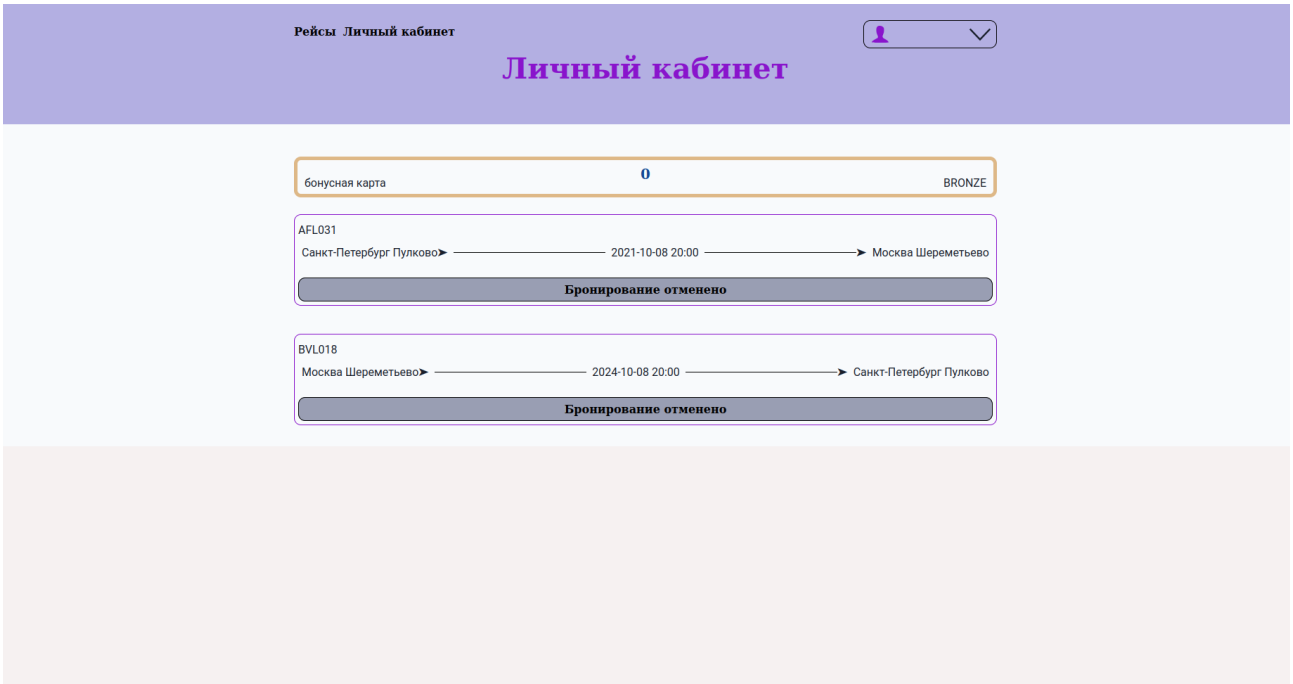

Рис. 3.5 – Страница с бонусным счётом и билетами пользователя.

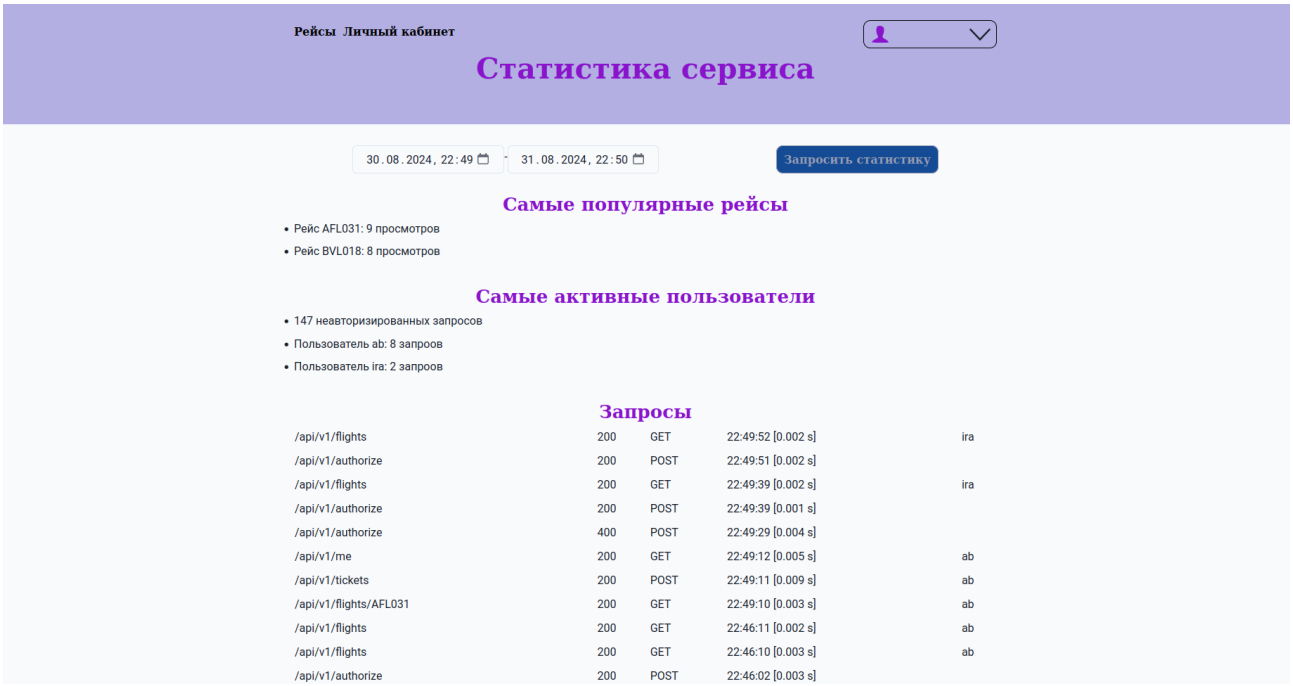

Рис. 3.6 – Страница со статистикой (доступна администратору).

ЗАКЛЮЧЕНИЕ

В рамках данной работы была разработана распределенная система, позволяющая производить бронирование ресйсов. Для достижения поставленной цели были решены следующие задачи:

- сформулированы основные требования к системе;
- описаны требования к подсистемам;
- описана архитектура системы, а также сценарии ее функционирования;
- выбраны и обоснованы инструменты для разработки ПО;
- реализован программный продукт.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Анализ трафика десяти крупных агрегаторов авиабилетов в выдаче «Яндекса» [Электронный ресурс]. – Режим доступа: https://vc.ru/seo/35717 analiz-trafika-desyati-krupnyh-agregatorov-aviabiletov-v-vydache-yandeksa (Дата обращения: 24.06.2024).

2. Erl T. SOA design patterns (paperback). — Pearson Education, 2008.

3. Bourhis P., Reutter J. L., Vrgoč D. JSON: Data model and query languages //Information Systems. – 2020. – Т. 89. – С. 101478.

4. Ahmad I. et al. Implementation of RESTful API Web Services Architecture in Takeaway Application Development //2021 1st International Conference on Electronic and Electrical Engineering and Intelligent System (ICE3IS). – IEEE, 2021. – С. 132-137.

5. Docker Documentation [Электронный ресурс]. – Режим доступа: https://docs.docker.com/ (Дата обращения: 30.06.2024).

6. Ubuntu Documentation [Электронный ресурс]. – Режим доступа: https://ubuntu.com/ (Дата обращения: 30.06.2024).

7. PostgreSQL Documentation [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/docs/ (Дата обращения: 30.06.2024).

8. Golang Documentation [Электронный ресурс]. – Режим доступа: https://go.dev/doc/ (Дата обращения: 30.06.2024).