СОДЕРЖАНИЕ

Техническое задание

Глоссарий

В данном техническом задании используются следующие обозначения:

1. Узел системы – региональный сервер, содержащий данные авторов и читателей указанного региона;
2. Валидация – проверка данных на соответствие заданным условиям и ограничениям;
3. REST – архитектурный стиль взаимодействия компонентов распределённого приложения в сети;
4. Медиана времени отклика – среднее время предоставления данных пользователю;
5. Латентность географического положения – увеличение времени отклика приложения, обуславливаемое географическим положением элементов системы или пользователя.
6. Аутентификация – процесс проверки подлинности пользователя или устройства.
7. Авторизация – это процесс проверки прав доступа.
8. OpenID Connect – это протокол аутентификации и авторизации, который строится на основе протокола OAuth 2.0.
9. Identity Provider – это сервис, который управляет аутентификацией и предоставляет информацию о пользователе в рамках системы аутентификации и авторизации;
10. JSON – это популярный формат текстовых данных, который используется для обмена данными в современных веб - и мобильных приложениях;
11. JSON Web Token (JWT) – объект, состоящий из трех частей: заголовка (header), полезной нагрузки (payload) и подписи; является открытым стандартом (RFC 7519) для создания токенов доступа, основанный на формате JSON;

12. Kubernetes – открытое программное обеспечение для оркестровки контейнеризированных приложений, автоматизации их развёртывания, масштабирования и координации в условиях кластера;

13. Паттерн «репозиторий» – это шаблон проектирования, представляющий собой абстрактный механизм хранения для коллекций сущностей;

14. Паттерн «пул объектов» – это шаблон проектирования, представляющий собой набор инициализированных и готовых к использованию объектов: когда системе требуется объект, он не создаётся, а берётся из пула; когда объектб ольше не нужен, он не уничтожается, а возвращается в пул;

15. Паттерн «фабрика объектов» – это порождающий шаблон проектирования для создания объектов, относящихся к одной предметной области, связанных логически.

Введение

В современном мире авиаперевозки становятся все более популярным способом путешествия, что требует эффективной системы бронирования авиабилетов. Разработка системы бронирования авиарейсов имеет высокую актуальность в связи с растущим спросом на авиаперевозки и необходимостью обеспечения удобства и оперативности процесса бронирования.

Данный проект представляет собой техническое задание на разработку портала бронирования авиабилетов, функциональность которого включает в себя возможность поиска и выбора оптимальных авиарейсов по различным критериям (цена, время вылета), онлайн-бронирование билетов, управление бронированиями, а также использование системы лояльности.

В данном техническом задании будут описаны основные функциональные требования к порталу, архитектура системы и основные интеграции с другими сервисами.

Краткое описание предметной области

Система бронирования авиарейсов охватывает все аспекты планирования, организации и осуществления пассажирских авиаперевозок. Она включает в себя такие процессы, как поиск и сравнение авиабилетов, бронирование мест, управление тарифами и скидками.

Существующие аналоги

Среди аналогов разрабатываемого проекта можно выделить следующие:

— Skyscanner: Это один из самых известных и широко используемых инструментов для поиска и бронирования авиабилетов. Система предоставляет информацию о рейсах более чем 1200 авиакомпаний и предлагает широкий выбор фильтров для поиска. Однако, Skyscanner не имеет прямого доступа к продаже авиабилетов, и клиенты должны перейти на сайт авиакомпании для бронирования.

— Google Flights: Еще одна популярная система бронирования авиабилетов, которая предлагает широкий спектр возможностей для поиска и сравнения цен. Google Flights использует алгоритмы машинного обучения для анализа данных и предоставления наиболее подходящих вариантов перелета. Однако, как и Skyscanner, Google Flights является только поисковым инструментом и не предоставляет возможности бронирования напрямую.

— Momondo: Эта система бронирования авиабилетов также предлагает большой выбор рейсов и направлений, а также позволяет сравнивать цены и выбирать наиболее подходящий вариант. Momondo имеет прямой доступ к продаже авиабилетов и предлагает различные дополнительные услуги, такие как страховка и помощь в оформлении документов.

Разрабатываемый проект должен обладать следующими преимуществами:

1. собственная система лояльности: все приведённые выше сервисы являются лишь агрегаторами авиабилетов, но не предлагают систему лояльности;

2. просмотр активных бронирований: приведённые сервисы не имеют личного кабинета пользователя, а потому не предоставляют возможности просмотра купленных билетов;

3. возврат забронированных билетов: приведённые сервисы не имеют личного кабинета пользователя, а потому не предоставляют возможности возврата купленных билетов.

Описание системы

Разрабатываемый сервис должен представлять собой распределенную систему для управления бронированиями билетов на авиарейсы.

Пользователь может выступать в качестве:

— клиента, который может посмотреть активные бронирования, забронировать новый билет или вернуть приобретённый ранее, а также посмотреть остаток средств на счёте программы лояльности;

— администратора, который может просматривать отчеты о работе системы бронирования.

Основания для разработки

Разработка ведется в рамках выполнения лабораторных работ по курсу «Методология программной инженерии» на кафедре «Программное обеспечение ЭВМ и информационные технологии» факультета «Информатика и системы управления» МГТУ им. Н. Э. Баумана.

Назначение разработки

Главное назначение разрабатываемого портала – возможность пользоваться бонусными баллами вне зависимости от авиакомпании, у которой приобретаются билеты, а также управлять текущими бронированиями (просмотривать или удалять).

Требования к системе

1. Разрабатываемое программное обеспечение должно обеспечивать функционирование системы в режиме 24/7/365 со среднегодовым временем доступности не менее 99.9%.

2. Необходимо поддерживать возможность добавления нового узла во время работы системы без перезапуска.

3. Каждый узел должен автоматически восстанавливаться после сбоя в течение 15 минут.

4. Должна быть реализована деградация функциональности в случае отказа некритических источников.

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 без учета латентности географического расположения узла;

2. По результатам работы модуля сбора статистики медиана времени отклика системы на запросы, добавляющие или изменяющие информацию на портале не должна превышать 5 секунд без учета латентности географического расположения узла.

Функциональные требования к порталу с точки зрения пользователя

Портал должен обеспечивать реализацию следующих функций:

1. Регистрация и авторизация пользователей с валидацией вводимых данных как через интерфейс приложения, так и через популярные социальные сети.

2. Аутентификация пользователей.

3. Ролевая модель пользователей. Выделяются следующие роли:

   — покупатель;

   — администратор.

4. покупатель имеет следующий набор функций:

   — поиск билетов;

   — покупка билета, в том числе с использованием бонусных баллов;

   — просмотр купленных билетов;

   — возврат билета;

— просмотр истории начисления и списания бонусных баллов;

— просмотр остатка на счёте бонусных баллов.

5. Администратор имеет функцию неограниченного полномочия по изменению контента на портале.

Требования к надежности и к документации

Система должна работать в соответствии с данным техническим заданием без перезапуска. Необходимо использовать «зеркалируемые серверы» для всех подсистем, которые будут держать нагрузку в случае сбоя до тех пор, пока основной сервер не восстановится.

Исполнитель должен подготовить и передать Заказчику следующие документы:

— руководство по развертыванию Системы;

— руководство администратора Системы;

— руководство для покупателя по использованию Системы.

Топология системы

Топология системы представлена на рисунке 1.

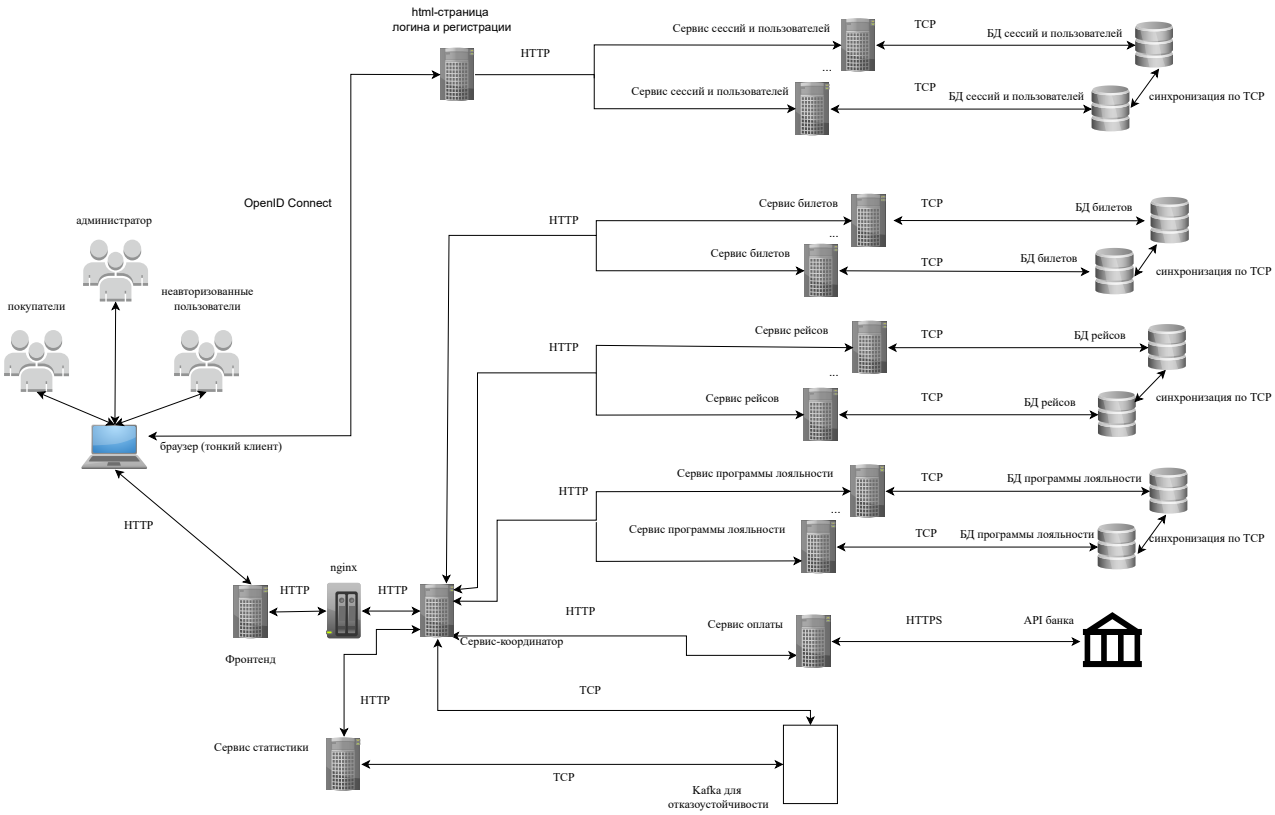

Рисунок 1 – Топология системы

Система должна состоять из 2 подсистем:

— подсистема авторизации;

— подсистема бронирования рейсов.

Подсистема бронирования рейсов должна состоять из фронтенда и шести сервисов, что наиболее целесообразно для реализации ее основного назначения.

Подсистема авторизации должна состоять из одного сервиса и html-страницы с полями для ввода логина и пароля, а также регистрации.

Взаимодействие подсистемы бронирования авиарейсов с подсистемой авторизации должно осуществляться по протоколу OpenID Connect.

Все сервисы подсистемы бронирования рейсов должны взаимодействовать друг с другом через сервис-координатор, запросы с фронтенда в том числе сначала должны приходить на сервис-коорлинатор, а затем перенаправляться на нужный сервис.

Сервисы, используемые пользователями часто (сервис сессий и пользователей, сервис-координгнатор, сервис рейсов, сервис билетов, сервис программы лояльности) должны работать в нескольких экземплярах в разных географических позициях для обеспечения одинаковой задержки при загрузке страниц

из разных географических позиций и, соответственно, меньшей нагрузки на сеть. Для реализации этого должна использоваться технология CDN, работающая в соответствии алгоритмом GeoDNS (алгоритм выбора сервера, который основан на географическом расположении пользователей).

В рамках одной географической позиции также могут быть использованы несколько работающих экземпляров сервиса. Нагрузка между сервисами может быть распределена с помощью алгоритма RoundRobin.

При этом некоторые из дублируемых сервисов обращаются к своей локальной копии базы данных. Отсюда возникает необходимость синхронизации баз данных. Для этого могут быть использованы инструменты для работы с распределёнными транзакциями, встроенные в большинство популярных СУБД.

Фронтенд должен принимать запросы от пользователя по протоколу HTTP и возвращать ответ в виде HTML страниц, файлов стилей и java script.

Перечислим сервисы и их ответственность в системе.

1. Сервис-координатор отвечает за координацию запросов внутри системы. Для реализации балансировки запросов используется инфраструктура Kubernetes.

2. Сервис сессий и пользователей отвечает за сессию пользователей портала и реализует следующие функции:

   — регистрация пользователя (покупателя);

   — авторизация пользователя (вход, или «логин»);

   — выход из сессии («логаут»).

   — получение информации, изменение, удаление покупателя.

   Сервис использует в своей работе базу данных. О каждом покупателе хранится следующая информация:

   — логин (уникальное в рамках системы текстовое поле, максимум 80 символов);

   — имя и фамилия (каждое является текстовым полем, максимум 80 символов);

   — дата рождения.

3. Сервис программы лояльности реализует следующие функции:

   — узнать баланс на счёте лояльности пользователя и историю изменения счёта лояльности с указанием для каждого события истории: суммы изменения, даты изменения, типа изменения и идентификатора билета, связанного с изменением;

   — произвести начисление бонусных баллов при покупке билета;

   — произвести списание бонусных баллов при возврате билета (при этом остаток бонусного счёта не может быть меньше 0).

   Сервис программы лояльности использует в своей работе базу данных для хранения текущего баланса и истории изменений бонусного счёта каждого покупателя.

4. Сервис билетов реализует следующие функции:

   — получить информацию о всех купленных билетах пользователя;

   — получить информацию о билете пользователя с заданным идентификатором;

   — отметить билет купленным или возвращённым.

   Сервис билетов использует в своей работе базу данных. О каждом билете хранится следующая информация:

   — уникальный а рамках системы идентификатор билета;

   — логин пользователя, купившего билет;

   — идентифкатор рейса;

   — цена билета;

   — статус билета (оплачен / возвращён).

5. Сервис рейсов реализует следующие функции:

   — получить информацию о всех рейсах;

   — получить информацию о рейсе с заданным идентификатором.

   Сервис рейсов использует в своей работе базу данных. О каждом рейсе хранится следующая информация:

— уникальный а рамках системы идентификатор рейса;

— дата полёта;

— аэропорт отправления;

— аэропорт назначения;

— цена рейса.

6. Сервис оплаты отвечает взаимодейтсвие с внешними сервисами банка в части оплаты билетов и возврата билетов.

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

Кроме того, nginx отвечает за балансировку трафика между несколькими серверами сервера-координатора, а также за возврат ответов на статические запросы – изображения, кеширование.

Требования по реализации со стороны заказчика

1. Требуется использовать сервис-ориентированную архитектуру для реализации системы.

2. Требуется использование очереди сообщений Kafka для повышения отказоустойчивости системы.

3. Система состоит из микросервисов. Каждый микросервис отвечает за свою область логики работы приложения.

4. Взаимодействие между сервисами осуществляется посредством HTTP запросов.

5. Данные сервисов должны храниться в базе данных. Каждый сервис взаимодействует только со своей схемой данных. Взаимодействие сервисов происходит по технологии REST.

6. При недоступности систем портала должна осуществляться деградация функциональности или выдача пользователю сообщения об ошибке.

7. Необходимо предусмотреть авторизацию пользователей через интерфейс приложения.

8. Для авторизации использовать протокол OpenID Connect.

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

10. Приложение должно поддерживать возможность горизонтального и вертикального масштабирования за счет увеличения количества функционирующих узлов и совершенствования технологий реализации компонентов и всей архитектуры системы.

Функциональные требования по подсистемам

1. Фронтенд – это серверное приложение при разработке которого необходимо учитывать следующие факторы:

   — фронтенд должен принимать запросы по протоколу HTTP и формировать ответы пользователям портала в формате HTML;

   — в зависимости от типа запроса фронтенд должен отправлять последовательные запросы в соответствующие бекенды;

   — запросы к бекендам необходимо осуществляет по протоколу HTTP; данные необходимо передавать в формате JSON;

   — требуется использование веб-сервиса Nginx для более быстрого возврата пользователям статического содержимого (изображения, таблиц стилей, JavaScript файлов), а также для балансировки запросов к нескольким экземплярам сервиса-координатора.

2. Сервис пользователей, сервис сессий, сервис программы лояльности, сервис рейсов и сервис билетов – 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.

3. 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);

— существовать в нескольких экземплярах, чтобы координация запросов не была узким местом приложения;

— уведомлять сервис статистики о событиях в системем.

4. Сервис статистики и сервис оплаты – 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.

Сценарий взаимодействия фронтенда и бекендов

Рассмотрим, как взаимодействуют фронтенд и бекенды на примере выполнения запроса от пользователя на получение списка рейсов.

1. На фронтенд приходит данный запрос пользователя.

2. Фронтенд анализирует запрос пользователя и, если пользователь был авторизован ранее, получает из него JWT-токен. Далее выполняется

запрос к ближайшему географически наиболее свободному сервису-координатору. Координатор проверяет корректность полученных данных (того, что данные поступили в ожидаемом формате, то есть в формате, соответствующем протоколу взаимодействия). Координатор отправляет запрос на сервис рейсов. В сервисе рейсов осуществляется проверка корректности полученных данных (того, что данные поступили в ожидаемом формате, то есть в формате, соответствующем протоколу взаимодействия) и проверка JWT-токен-а (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 рейсов сервису-координатору, который в свою очередь возвращает результат на фронтенд.

3. Если ошибки нигде не произошло, то производится генерация HTML содержимого страницы ответа пользователю с использованием данных, полученных от сервиса рейсов. В ином случае генерируется страница с описанием ошибки.

Пользовательский интерфейс

Для реализации пользовательского интерфейса должен быть использован подход MVP (Model-View-Presenter). Этот подход к проектированию интерфейса является популярным шаблоном проектирования, который помогает разделить логику приложения на три основных компонента: Модель (Model), Представление (View) и Презентер (Presenter). Этот подход позволяет улучшить структуру приложения, облегчить его тестирование и управление.

Пользовательский интерфейс в разрабатываемой системе должен обладать следующими характеристиками:

— Адаптивность к размеру экрана устройства пользователя – пользовательский интерфейс «подстраивается» под всевозможные размеры экранов устройств: мобильных телефонов, планшетов, ноутбуков и т.д.

— 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.

— Интуитивно понятный интерфейс – все кнопки имеют подписи при наведении на них, многие содержат иконки, облегчающие восприятие пользователем.

Проектирование системы

Концептуальный дизайн

Концептуальная модель системы в нотации IDEF0

Для создания функциональной модели портала, отражающей его основные функции и потоки информации наиболее наглядно использовать нотацию IDEF0. На рисунке 1 приведена концептуальная модель системы. На рисунке 2 представлена декомпозиция функциональной модели системы.

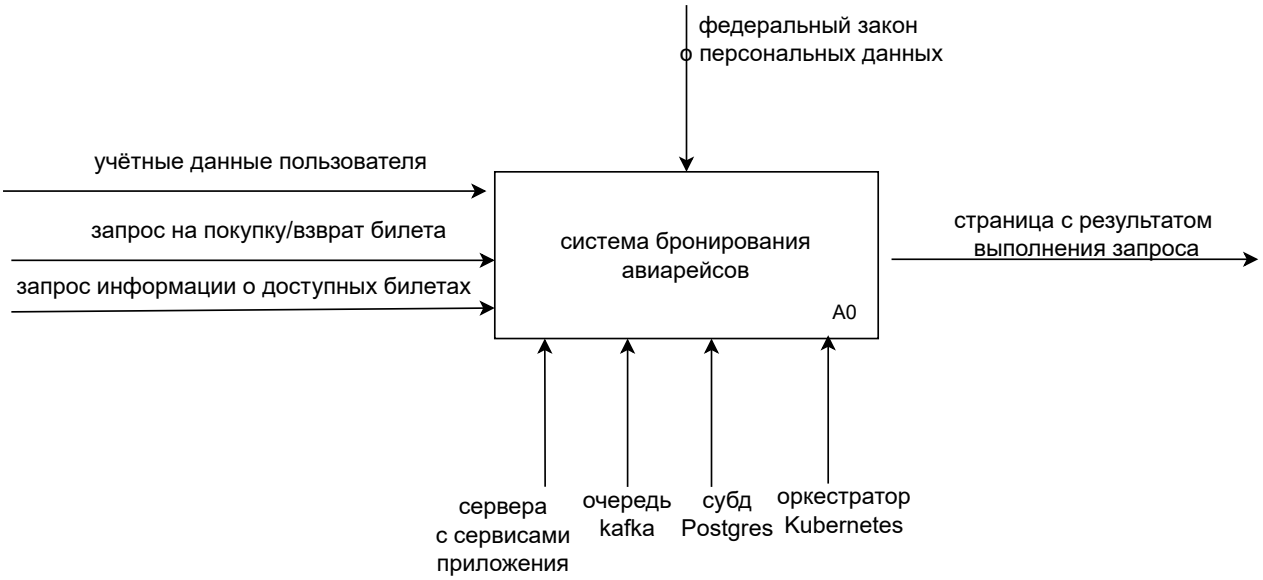

Рисунок 1 – Концептуальная модель в нотации IDEF0

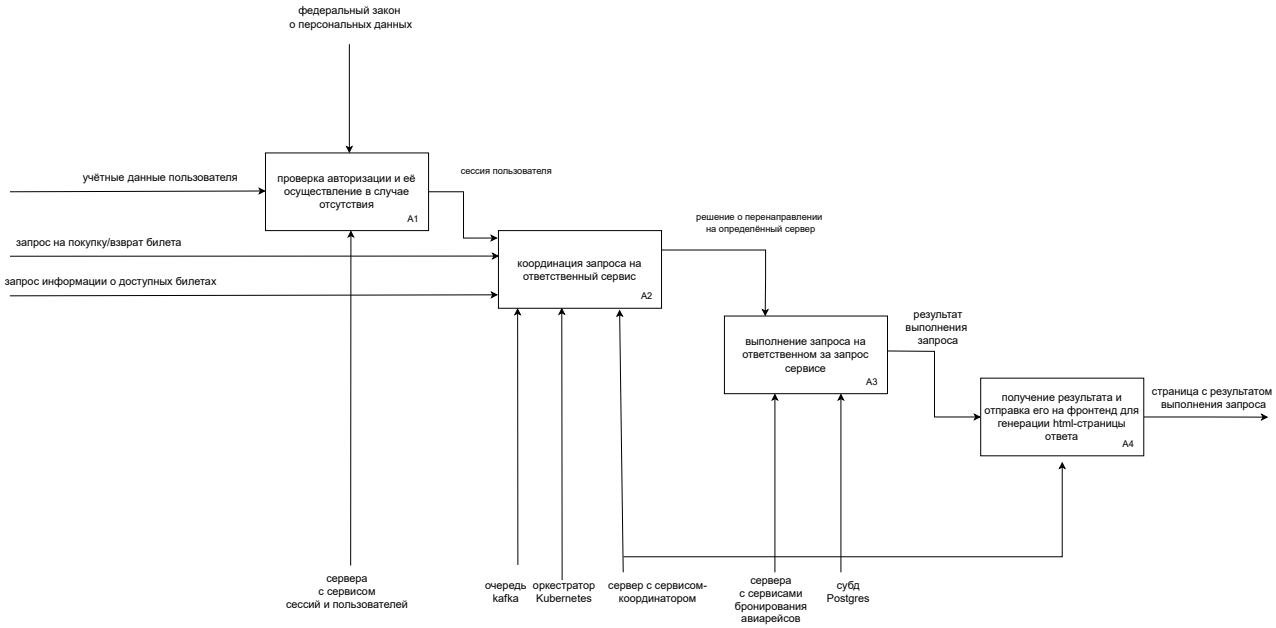

Рисунок 2 – Детализированная концептуальная модель в нотации IDEF0

Сценарии функционирования системы

Для детальной разработки портала используется унифицированный язык моделирования UML. В системе выделены 2 роли: администратор и пользователь (авторизованный и неавторизованный). На рисунках 4-3 представлены диаграммы прецедентов для выделенных ролей и описаны сценарии функционирования наиболее значимых прецедентов.

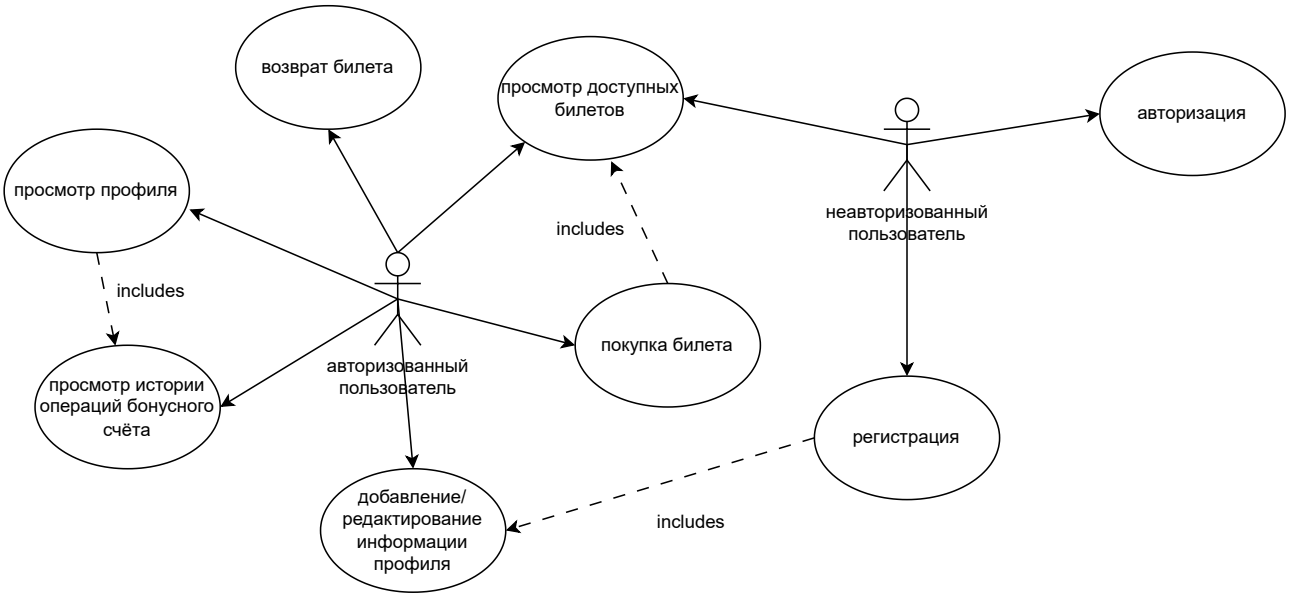

Рисунок 3 – Диаграмма прецедентов для роли «пользователь»

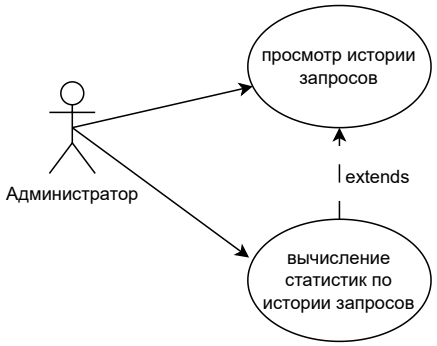

Рисунок 4 – Диаграмма прецедентов для роли «администратор»

Регистрация пользователя

1. Пользователь переходит на страницу входа с помощью кнопки «войти», либо автоматически перенаправляется на соответствующую страницу при попытке совершения действий, которые невозможно совершить без регистрации (например, покупка или возврат билетов).

2. Пользователь нажимает на кнопку «зарегистрироваться» и перенаправляется на страницу регистрации, где вводит различные наборы полей. Валидация входных данных осуществляется «на лету» на стороне пользователя. При отправке данных на фронтенд, он тоже производит валидацию.

3. Пользователь нажимает кнопку «Регистрация» и перенаправляется на главную страницу портала.

Авторизация на портале клиента или мастера

1. Пользователь переходит на страницу входа с помощью кнопки «войти», либо автоматически перенаправляется на соответствующую страницу при попытке совершения действий, которые невозможно совершить без регистрации (например, покупка или возврат билетов).

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 покупки билетов

Сценарий доступен как для авторизованного, так и для неавторизованного пользователя.

1. Пользователь задаёт параметры поиска авиабилета (город отправления, город назначения, дату поездки) и нажимает кнопку «найти».
2. На экране появляется список доступных билетов с детальной информацией о билете (цена, время вылета).

Покупка билета

Сценарий доступен только для авторизованного пользователя.

1. Пользователь задаёт параметры поиска авиабилета (город отправления, город назначения, дату поездки) и нажимает кнопку «найти».
2. На экране появляется список доступных билетов с детальной информацией о билете (цена, время вылета).
3. Пользователь выбирает билет, переходит на страницу с детальной информацией о билете и нажимает кнопку «купить».
4. Пользователю предлагается выбрать: начислить баллы за покупаемый билет или списать баллы с бонусного счёта для уменьшения цены билета.
5. С бонусным счётом производится операция в соответствии с выбранном на предыдущем шаге действием пользователя.

Возврат билета

Сценарий доступен только для авторизованного пользователя.

1. Пользователь выбирает кнопку «личный кабинет» на главной странице портала.
2. В разделе «купленные билеты» пользователь выбирает билет, который хочет вернуть, и нажимает соответствующую кнопку.
3. С бонусным счётом производятся дейтсвия, обратные к совершённым при покупке билета. При этом остаток на бонусном счёт не может быть отрицательным.

Просмотр операций с бонусным счётом

Сценарий доступен только для авторизованного пользователя.

1. Пользователь выбирает кнопку «личный кабинет» на главной странице портала.
2. Пользователь открывает раздел «история операций с бонусным счётом» и перенаправляется на страницу с историей.

Получение статистики

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

Спецификация сценария покупки билета

Нормальный ход сценария.

Таблица 1 – Спецификация покупки билета

Действия актера	Отклик системы
выбор билета из списка доступных	открытие страницы с подробной информацией о билете
запрос покупки билета	успешная проверка авторизации и запрос подтверждения операции
подтверждение операции	осуществление покупки

Альтернативный ход сценария.

Таблица 2 – Спецификация покупки билета

Действия актера	Отклик системы
выбор билета из списка доступных	открытие страницы с подробной информацией о билете
запрос покупки билета	успешная проверка авторизации и запрос подтверждения операции
отклонение операции	покупка не осуществляется

Альтернативный ход сценария.

Таблица 3 – Спецификация покупки билета

Действия актера	Отклик системы
выбор билета из списка доступных	открытие страницы с подробной информацией о билете
запрос покупки билета	неуспешная проверка авторизации, перенаправление на страницу ввода учётных данных
ввод учётных данных	успешная проверка учётных данных, запрос подтверждения операции
подтверждение операции	осуществление покупки

Логический дизайн

Диаграммы классов

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

— сервиса рейсов – на рисунке 5;

— сервиса билетов – на рисунке 6;

— сервиса сессий и пользователей – на рисунке 7;

— сервиса программы лояльности – на рисунке 8;

— сервиса-координатора – на рисунке 9;

— сервиса оплаты – на рисунке 10;

— сервиса статистики – на рисунке 11.

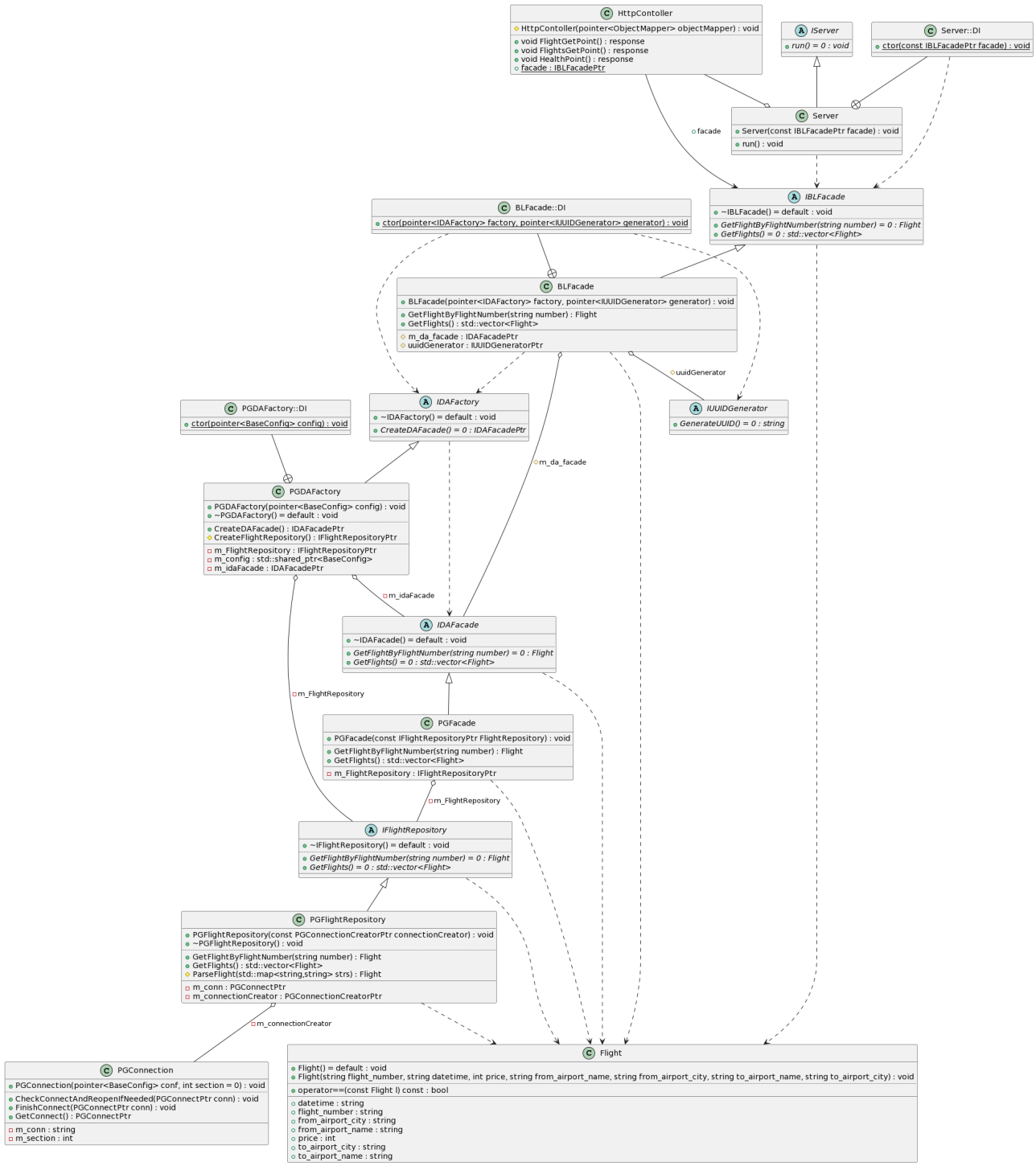

Рисунок 5 – Диаграмма классов сервиса рейсов

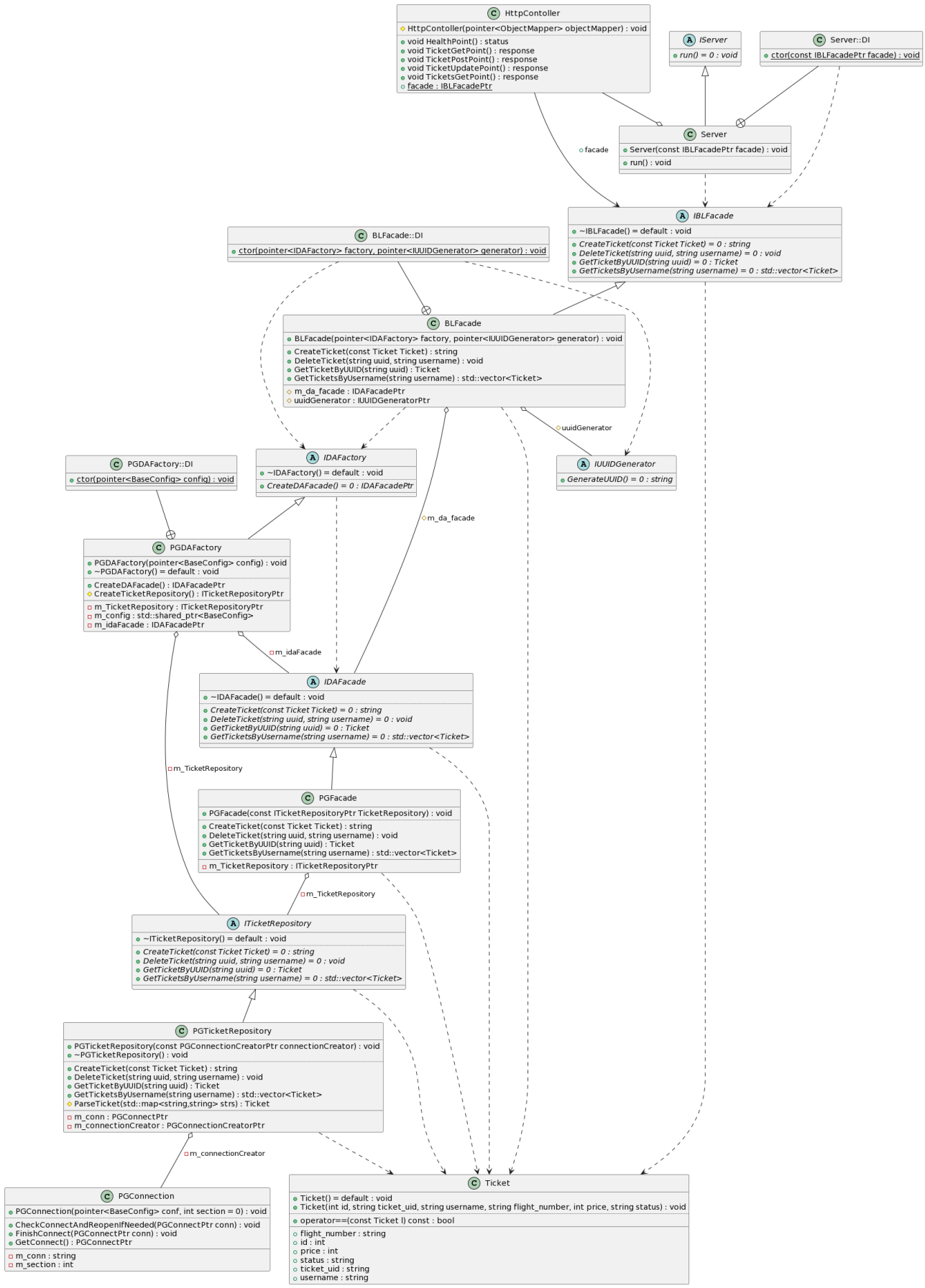

Рисунок 6 – Диаграмма классов сервиса билетов

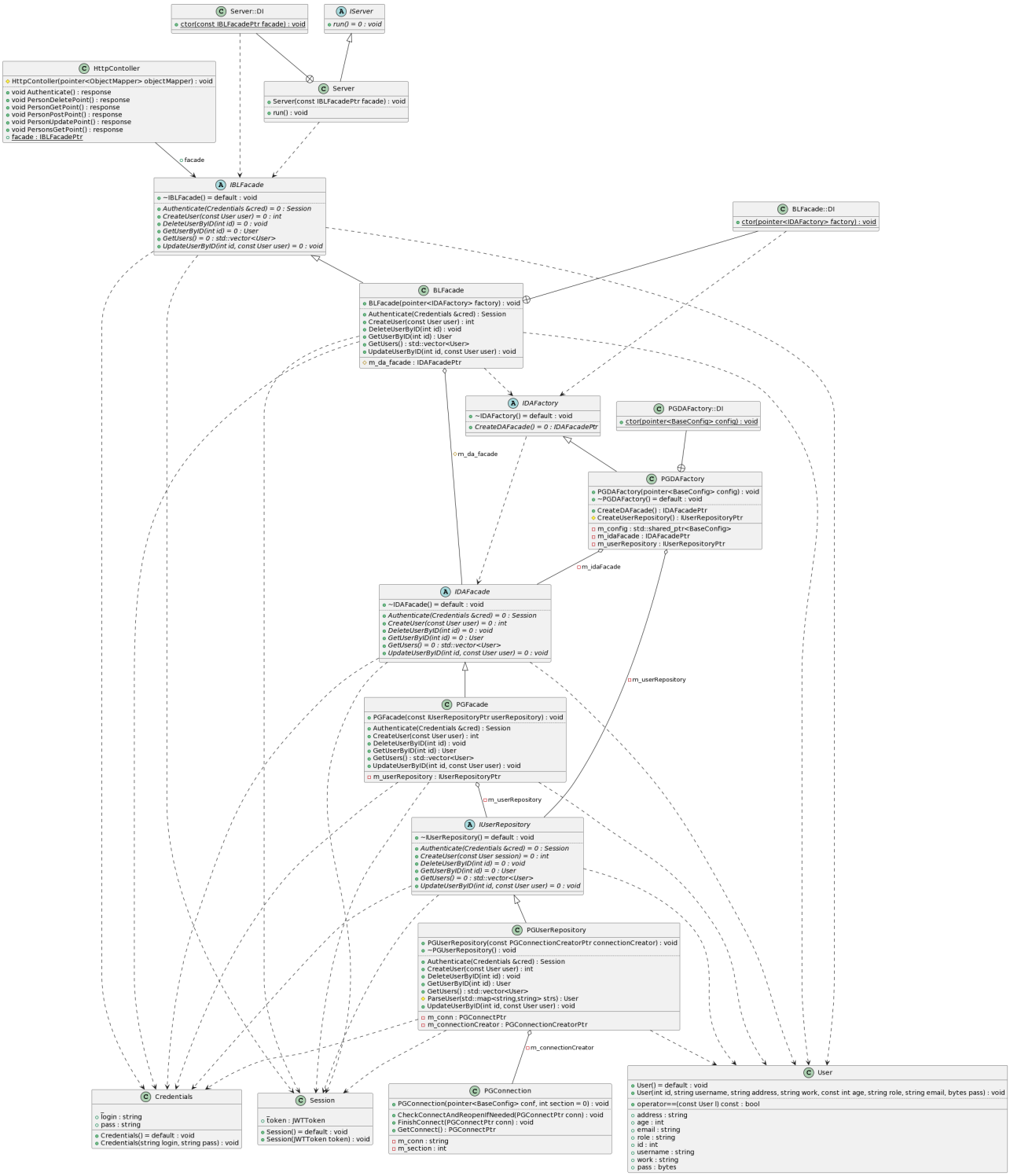

Рисунок 7 – Диаграмма классов сервиса сессий и пользователей

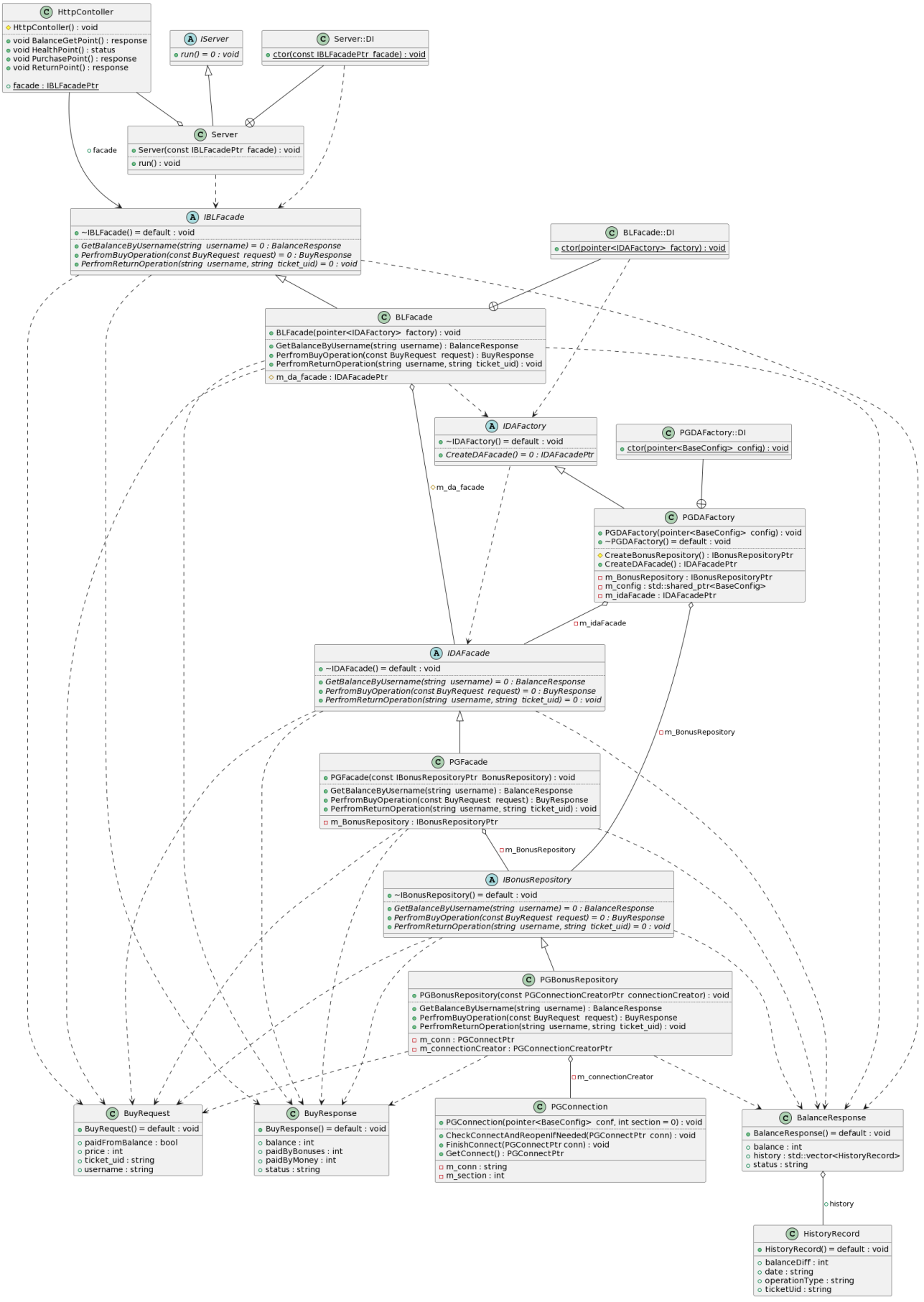

Рисунок 8 – Диаграмма классов сервиса программы лояльности

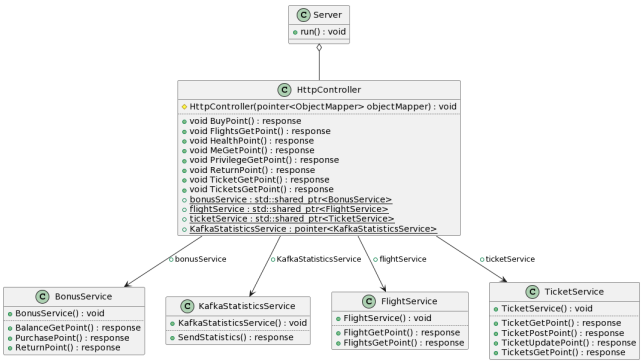

Рисунок 9 – Диаграмма классов сервиса-координатора

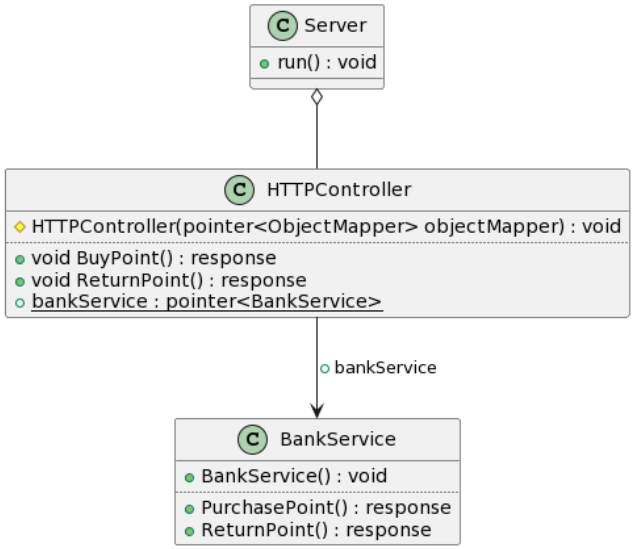

Рисунок 10 – Диаграмма классов сервиса оплаты

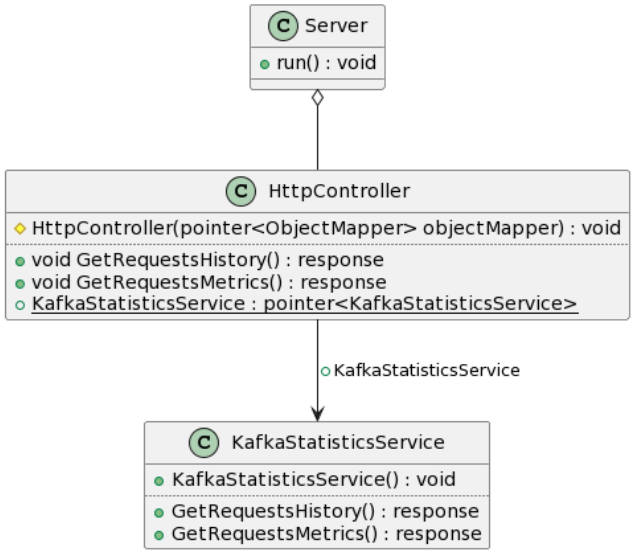

Рисунок 11 – Диаграмма классов сервиса статистики

Описание классов сервисов

Сервисы рейсов, билетов, программы лояльности и сессий и пользователей спроектированы похожим образом. Они имеют:

— слой доступа к данным, реализованный с помощью паттерна «Репозиторий» для абстракции хранения, а также паттерна «пул объектов» для эффективного использования активных подключений к базе данных;

— слой логики работы сервиса;

— слой интерфейса – слой связи с другими сервисами с помощью http-запросов.

Так как сервисы спроектированы одинаково, часть классов совпадают. Опишем их:

— IDAFacade – абстрактный интерфейс слоя доступа к данным; PGDAFacade – реализация этого абстрактного интерфейса для работы с данными, хранящимися под управлением СУБД Postgres;

— IDAFactory – абстрактный интерфейс фабрики объектов, относящихся к библиотеке работы с данными (интерфейс спроектирован в соответствии с паттерном «фабрика объектов»); PGDAFactory – реализация этого абстрактного интерфейса для работы с данными, хранящимися под управлением СУБД Postgres;

— PGConnection – пул активных подлключений к базе данных, спроектированный в соответствии с паттерном «пул объектов»;

— IBLFacade – абстрактный интерфейс слоя логики; BLFacade – реализация этого абстрактного интерфейса;

— IServer – абстрактный интерфейс серверного приложения, предоставляющего интерфейс, описанный HTTPController; Server – реализация этого интерфейса.

— HTTPController – класс, описывающий 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 занимается подготовкой результирующих данных к отправке по сети после выполнения запроса.

Кроме того, каждый сервис имеет собственные классы. Опишем их.

Описание классов сервиса рейсов

— Flight – класс, описывающий рейс, имеет поля:

  – дата и время рейса;
  – цена рейса;
  – номер рейса;
  – аэропорты вылета и прилёта;
  – города вылета и прилёта.

— IFlightRepository – абстрактный интерфейс для работы с данными рейсов (интерфейс спроектирован в соответствии с паттерном «репозиторий»). PGFlightRepository – реализация этого абстрактного интерфейса для работы с данными, хранящимися под управлением СУБД Postgres.

Описание классов сервиса билетов

— Ticket – класс, описывающий билет, имеет поля:

  – номер билета;
  – цена билета;
  – номер рейса;
  – статус (оплачен или отменён);
  – идентификатор билета;
  – владелец билета.

— ITicketRepository – абстрактный интерфейс для работы с данными билетов (интерфейс спроектирован в соответствии с паттерном «репозиторий»). PGTicketRepository – реализация этого абстрактного интерфейса для работы с данными, хранящимися под управлением СУБД Postgres.

Описание классов сервиса сессий и пользователей

— User – класс, описывающий пользователя, имеет поля:

  – идентификатор пользователя;
  – логин пользователя;
  – роль пользователя;
  – возраст пользователя;
  – пароль пользователя (в виде хэша).

— Session – класс, описывающий сессию пользователя, является JWT-token-ом;

— Credentials – класс, который хранит логин и пароль пользователя.

— IUserRepository – абстрактный интерфейс для работы с данными пользователей и сессий (интерфейс спроектирован в соответствии с паттерном «репозиторий»). PGUserRepository – реализация этого абстрактного интерфейса для работы с данными, хранящимися под управлением СУБД Postgres.

Описание классов сервиса программы лояльности

— BuyRequest – класс, описывающий запрос на покупку билета, имеет поля:

  – решение пользователя о списании или о начислении бонусов;
  – цена билета;
  – номер билета;
  – имя пользователя.

— BuyResponse – класс, описывающий ответ на покупку билета, имеет поля:

  – баланс счёта программы лояльности после покупки билета;
  – количество денег, списанное за счёт бонусов;

  - количество денег, списанное за счёт средств пользователя;
  - статус счёта лояльности (золотой, серебряный, бронзовый) после покупки.

— BalanceResponse – класс, описывающий ответ на запрос информации о бонусном счёта и истории операций бонусного счёта, имеет поля:

  - баланс счёта программы лояльности;
  - статус счёта лояльности (золотой, серебряный, бронзовый);
  - история операций (каждая запись содержит: абсолютное значение изменения баланса, дату операции, типа операции – списание или начисление, идентификатор билета).

— IBonusRepository – абстрактный интерфейс для работы с данными программы лояльности (интерфейс спроектирован в соответствии с паттерном «репозиторий»). PGBonusRepository – реализация этого абстрактного интерфейса для работы с данными, хранящимися под управлением СУБД Postgres.

Описание классов сервиса-координатора

— HTTPController – класс, описывающий набор HTTP-методов, которые доступны на сервисе. Фактически предасвляет собой весь программный интерфейс системы. В своей работе для обслуживания приходящих запросов сервис использует интерфейсы сервисов программы лояльности (BonusService), рейсов (FlightService) и билетов (TicketService). Кроме того, сервис перенаправляет статистику запросов в очередь Kafka с помощью интерфейса KafkaStatistcsService.

— Server – реализация серверного приложения, предоставляющего интерфейс, описанный HTTPController.

Описание классов сервиса оплаты

— HTTPController – класс, описывающий набор HTTP-методов, которые доступны на сервисе (покупка и возврат билетов). В своей работе для обслуживания приходящих запросов сервис использует сервисы банка для выполнения транзакций со счётом пользователя.

— Server – реализация серверного приложения, предоставляющего интерфейс, описанный HTTPController.

Описание классов сервиса статистики

— HTTPController – класс, описывающий набор HTTP-методов, которые доступны на сервисе (просмотр истории запросов к системе и метрики, рассчитанные на основе этой истории). В своей работе для обслуживания приходящих запросов сервис использует очередь Kafka.

— Server – реализация серверного приложения, предоставляющего интерфейс, описанный HTTPController.

Диаграмма деятельности

На рисунке 12 изображена диаграмма деятельности при покупке билета.

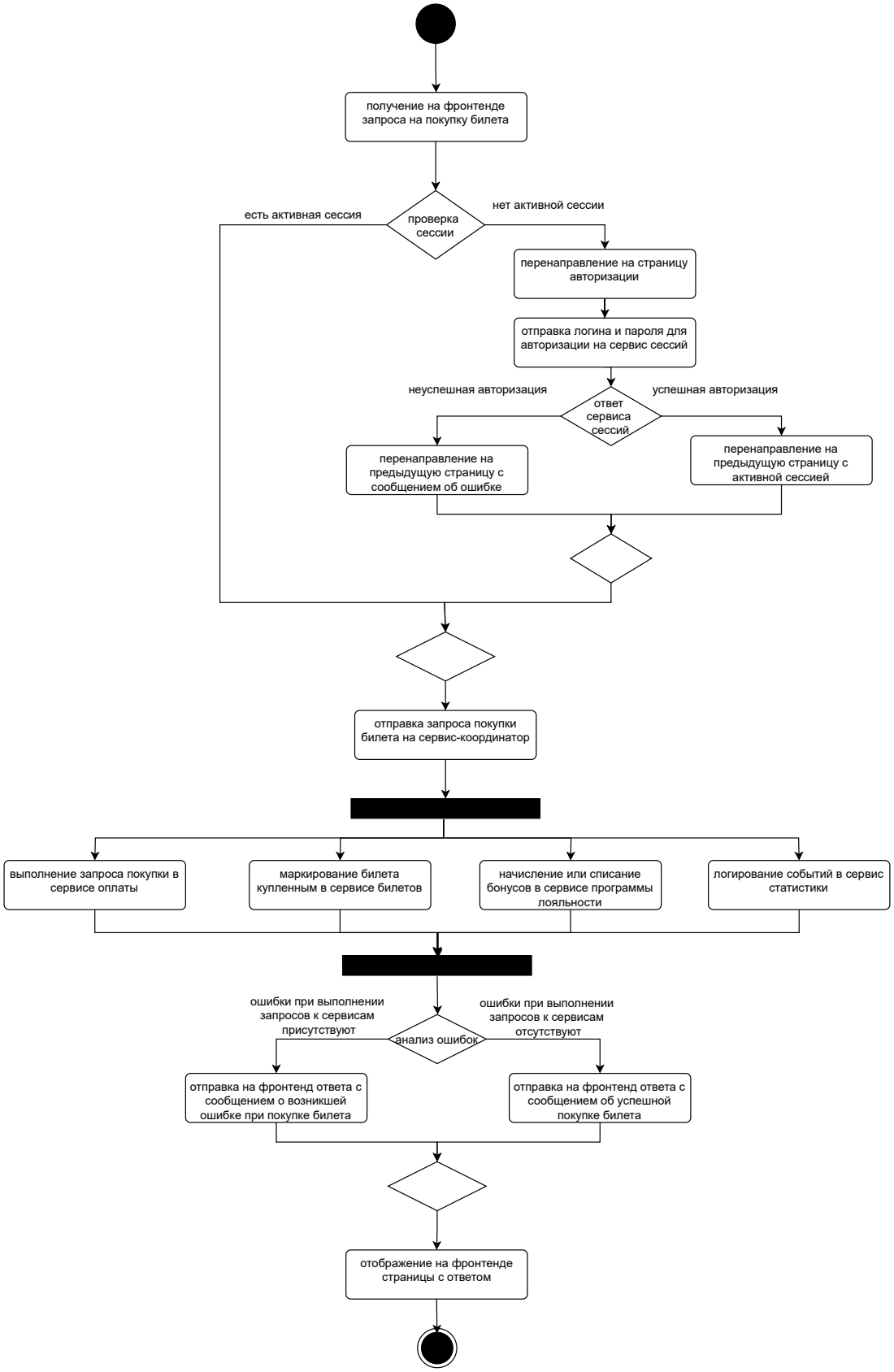

Рисунок 12 – Диаграмма деятельности при покупке билета

Высокоуровневый дизайн пользовательского интерфейса

Пользовательский интерфейс в разрабатываемой системе представляет собой Web-интерфейс, доступ к которому осуществляется через браузер (тонкий клиент).

Страница портала состоит из «шапки» (верхней части страницы, в которой находится логотип и верхнее меню со ссылками на основные разделы портала), основной части и «футера» (нижней части страницы, в которой обычно размещают ссылки на редко посещаемые, но необходимые, страницы, например, страницы с пользовательским соглашением).

Обобщенно структуру страниц портала можно представить следующим образом:

— страница с обучением для пользователя;

— главная страница с поиском авиабилетов;

— страница подробной информацией о билете и кнопкой покупки;

— личный кабинет пользователя (информация о пользователе с возможностью ее изменить);

— страница с купленными билетами (с возможностью вернуть еще не использованные билеты);

— страница программы лояльности (история операций бонусного счета, остаток на бонусном счёте);

— страница входа;

— страница регистрации;

— страница со статистикой запросов в приложении (доступно администраторам);

— о нас (гарантия, способы оплаты, контакты).

Ниже приведены основные формы портала.

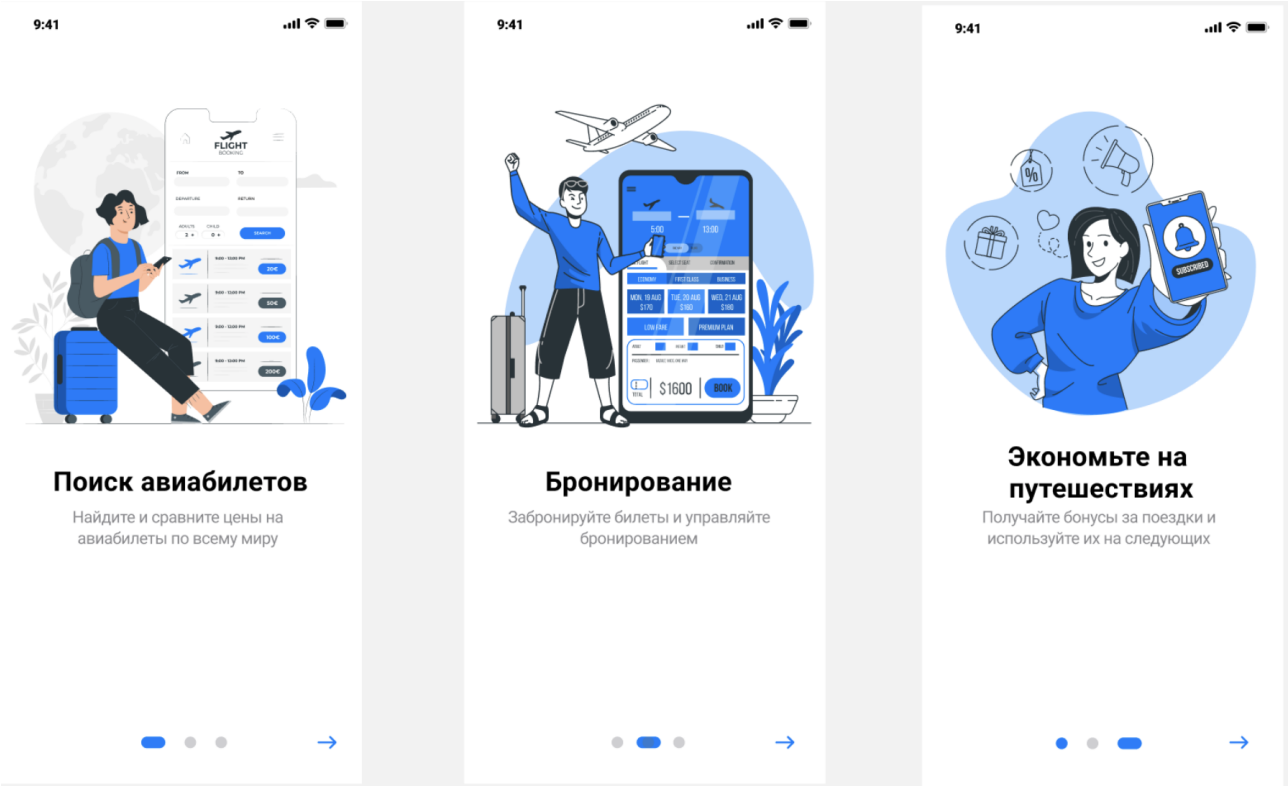

Рисунок 13 – Форма обучения пользователя

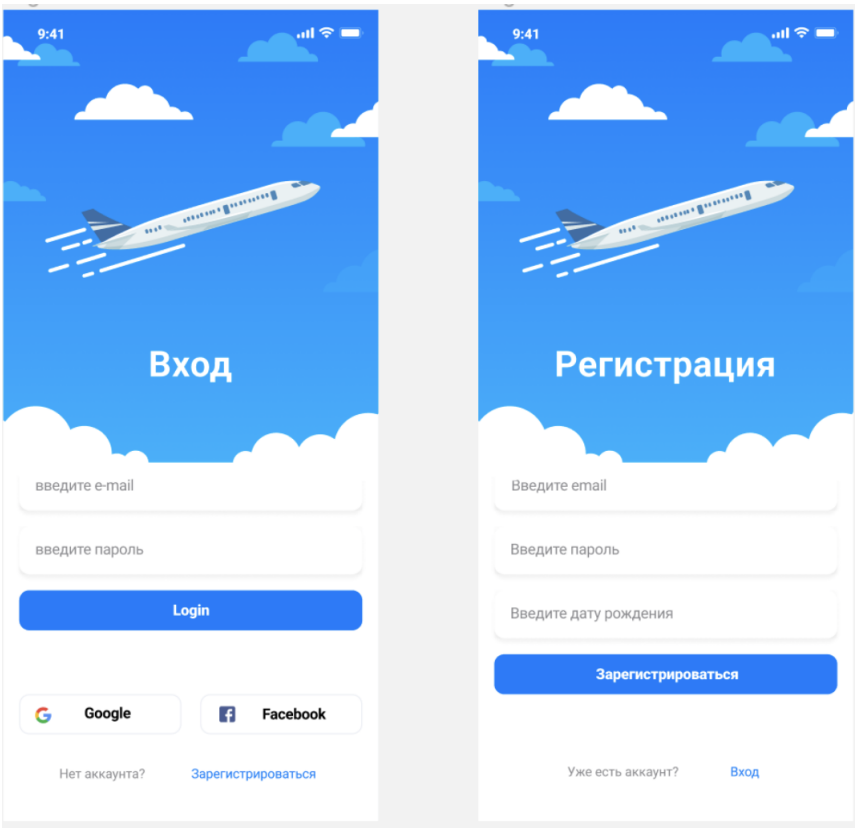

Рисунок 14 – Форма входа и регистрации пользователя

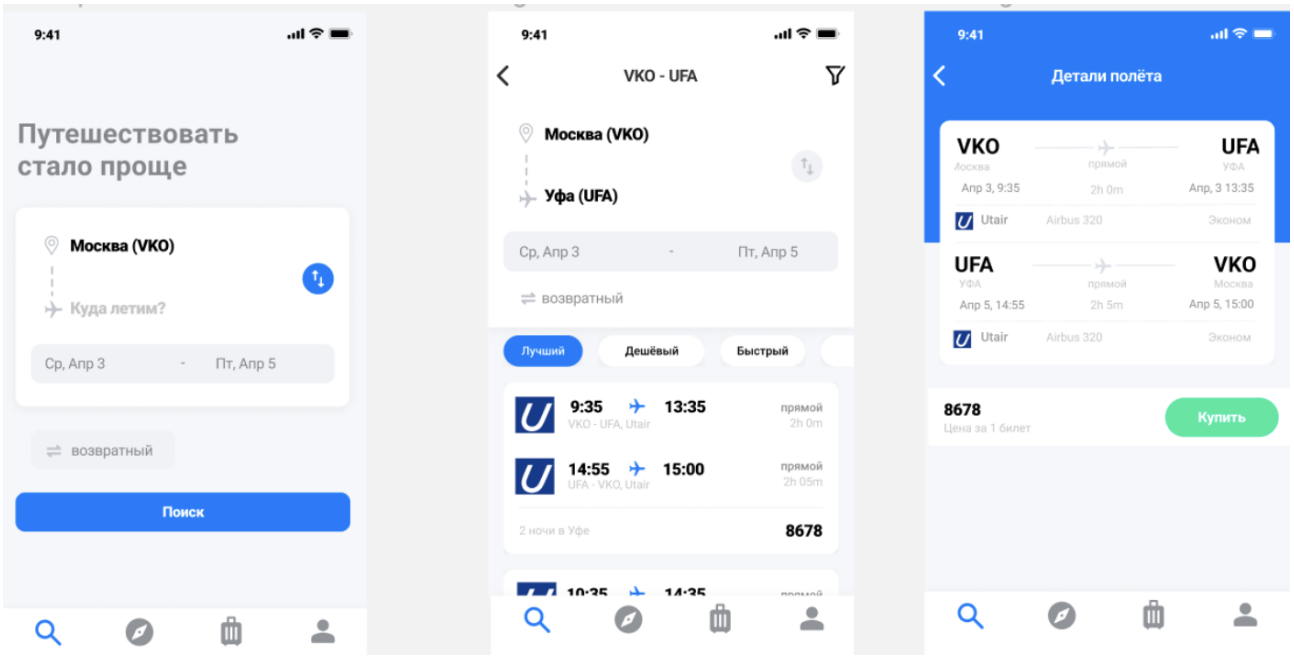

Рисунок 15 – Формы с поиском авиабилетов и покупкой билета

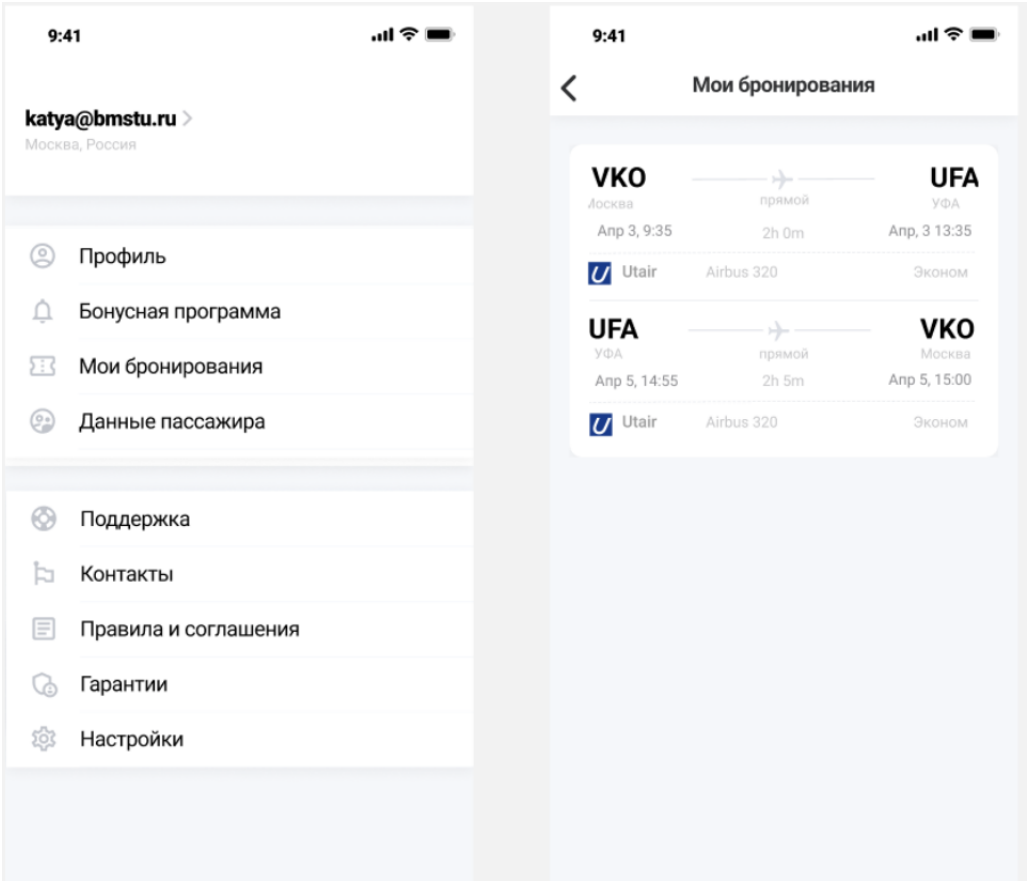

Рисунок 16 – Формы с аккаунтом пользователя и бронированиями

Физический дизайн

Выбор системы развертывания компонентов распределенной системы

Согласно требованиям технического задания, разрабатываемая система должна быть распределенной. Характерной особенностью распределенных систем является высокое многообразие используемых технологий. Особенно непростая ситуация возникает, когда разные компоненты системы используют разные версии одной и той же библиотеки. Для того чтобы компоненты не конфликтовали друг с другом, необходимо ввести требование изолированности. В этом случае приходится использовать отдельные серверы, что может быть экономически нецелесообразно, либо использовать контейнеризацию.

Контейнеризация – это методология разработки и управления приложениями, которая позволяет упаковывать приложение и все его зависимости в изолированный контейнер (образ изолируемой части системы, содержащий приложение со всеми его зависимостям). Контейнеры обеспечивают среду выполнения для приложения, которая полностью отделена от других контейнеров и основной системы. Это облегчает развертывание, масштабирование и управление приложениями, а также обеспечивает надежность и безопасность при работе в различных средах.

В качестве платформы для автоматизации развертывания, масштабирования и управления контейнеризированными приложениями на серверах было решено использовать Kubernetes. Такое решение было принято в результате следующего сравнительного анализа.

Существует несколько аналогов Kubernetes, которые также предоставляют возможности для управления контейнеризированными приложениями:

1. Docker Swarm – это оркестратор контейнеров, разработанный Docker, который позволяет управлять кластером Docker-хостов и запускать контейнеры в них. Он более прост в использовании по сравнению с Kubernetes, но может быть менее мощным в некоторых аспектах.

2. Apache Mesos – это распределенная система управления ресурсами, которая также поддерживает запуск контейнеров. Он предоставляет более общий подход к управлению ресурсами и приложениями, чем

Kubernetes.

3. Amazon ECS (Elastic Container Service) – это управляемый сервис от Amazon Web Services для запуска и управления контейнерами на инфраструктуре AWS. Он предлагает простой способ запуска и масштабирования контейнеров без необходимости управления инфраструктурой.

Достоинства Kubernetes по сравнению с аналогами:

— Мощные возможности оркестрации: Kubernetes предоставляет широкий спектр возможностей для управления и автоматизации развертывания приложений в контейнерах.

— Большое сообщество и экосистема: Kubernetes имеет активное сообщество разработчиков и широкий выбор инструментов и плагинов для расширения его функциональности.

— Поддержка различных облачных и локальных сред: Kubernetes поддерживает различные облачные провайдеры и может быть развернут как локально, так и в облаке.

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.

— 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 недолжен ограничивать разработчиков в выборе программного обеспечения, библиотек.

— Цена.

Под данные требования лучше всего подходит ОС Ubuntu. Ubuntu – это дистрибутив, использующий ядро Linux. Как и все дистрибутивы Linux, Ubuntu является ОС с открытым исходным кодом, бесплатным для использования.

Выбор СУБД

В качестве СУБД была выбрана PostgreSQL, 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 мира.

— 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.

Выбор языка разработки и фреймворков компонент портала

Проанализируем техническое задание на разработку портала. Исходя из приведенных требований к системе, можно выявить требования к языку программирования:

— Совместимость с выбранными ранее технологиями. Выбранный язык должен уметь взаимодействовать с ОС Linux, СУБД PostgreSQL.

— Производительность: C++ является компилируемым языком программирования, что позволяет создавать быстродействующие приложения. Он обладает низким уровнем абстракции, что позволяет разработчику более тонко управлять ресурсами и оптимизировать производительность программы.

— Расширяемость: C++ обладает возможностью использовать объектно-ориентированный подход к программированию, что делает его удобным для создания сложных и масштабируемых систем.

Выбор oatpp в качестве фреймворка для разработки программного обеспечения также имеет свои преимущества и может быть обоснован следующими аспектами:

— Высокая производительность: oatpp является легковесным и быстрым фреймворком, который спроектирован для обеспечения высокой производительности приложений. Он оптимизирован для работы с сетевыми запросами и обработки данных, что позволяет создавать эффективные веб-сервисы.

— Поддержка многопоточности: oatpp предоставляет удобные инструменты для работы с многопоточностью, что позволяет создавать параллельные и распределенные приложения. Это особенно важно для систем, где требуется обработка большого количества запросов одновременно.

— RESTful API: oatpp поддерживает разработку RESTful API, что делает его удобным выбором для создания веб-сервисов и API. Он предоставляет инструменты для удобной маршрутизации запросов, валидации данных и других задач, связанных с разработкой веб-приложений.

— Модульность и расширяемость: oatpp построен на основе модульной архитектуры, что позволяет легко добавлять новый функционал и расширять возможности фреймворка. Это делает его гибким инструментом для разработки различных типов приложений.

Обеспечение надежности портала

В данной системе для обеспечения надежности функционирования СУБД будет применяться репликация и шардинг. Для обеспечения надежности данных СУБД необходимо разработать скрипт для автоматического создания резервной копий базы данных по расписанию.

Для фронтенда и бекендов целесообразно применить зеркалирование. Это обеспечит отказоустойчивость системы: в случае сбоя любого из ее узлов запросы на чтение данных будут выполняться.