**Software Architecture Document**

**(Описание Архитектуры)**

#### **1.** **Introduction (Введение)**

##### **1.1** **Purpose**

Целью данного документа является описание архитектуры разрабатываемой автоматизированной информационной системы контроля эксперимента “Платформа”, как набора точек зрения на неё - use case view, logical view, process view, deployment view, implementation view. Взаимодействие элементов разных точек зрения представлено в виде UML-диаграмм.

## 1.2 Scope (Область применения)

Этот документ относится к автоматизированной информационной системе контроля эксперимента “Платформа”, которое будет разработано инициативной группой “SaDiVa”. Разрабатываемая информационная система предоставит возможность сотрудникам эксперимента “Платформа” контролировать чистоту проведения эксперимента, а также влиять на события в эксперименте. Формат разрабатываемой системы будет представлять собой клиент-серверное приложение. Для работы с системой будет предусмотрено несколько сценариев использования в зависимости от прав пользователя.

Данный документ будет использоваться при разработке продукта программистами, программными архитекторами и системными аналитиками.

## 1.3 Definitions, Acronyms and Abbreviations (Определения и аббревиатуры)

см. Glossary

[Glossary.docx](https://docs.google.com/document/d/1Ou6epUB6-zqZc1wfUPYO2CbKB8MVWuJW/edit?usp=drive_link&ouid=103372480510558009952&rtpof=true&sd=true)

## 1.4 References (Ссылки)

[Vision.docx](https://docs.google.com/document/d/1E4bIXlkFc2ZZ4bEkHAf8uSEvKtMdK4Rw/edit?usp=drive_link&ouid=103372480510558009952&rtpof=true&sd=true)

* SRS: [SRS.docx](https://docs.google.com/document/d/1FPEVQYix8eWcn4zHgGHRQIlBTKPg9gIR/edit?usp=drive_link&ouid=103372480510558009952&rtpof=true&sd=true)
* UC:

[Usecase.docx](https://docs.google.com/document/d/1WsK6MgHvOmjKYwUbCVXd2ZMJJeIUCtnk/edit?usp=drive_link&ouid=103372480510558009952&rtpof=true&sd=true)

* RL:

[RiskList.docx](https://docs.google.com/document/d/1wFBYTW8Hgl7Q1R_LKRvdZGVd30RoQ7UR/edit?usp=drive_link&ouid=103372480510558009952&rtpof=true&sd=true)

## 1.5 Overview (Обзор документа)

В разделах данного документа содержится информация о:

1. Представлении архитектуры
2. Целях и ограничениях архитектуры
3. Ряде различных архитектурных представлений для изображения аспектов системы
4. Производительности и качестве

#### **2.** **Architectural Representation (Представление архитектуры)**

| Diagram\View | Use Case View | Logical View | Implementation view | Process view\*\* | Deployment View |
| --- | --- | --- | --- | --- | --- |
| Use Case Diagram | + | - | - |  | - |
| Class Diagram | + (Взаимодействие сущностей) | +  (Описание основных классов и интерфейсов их взаимодействия) | +  (Полное описание классов с указанием их методов/полей, указать типы связей между классами) |  | - |
| Activity Diagram | +  (Абстрактное описание) | +  (Более подробное описание, уровни взаимодействия должны совпадать с диаграммой пакетов) | +  (Полное описание прецедента с указанием вызываемых методов, используемых классов и объектов). |  | - |
| State Machine Diagram | +  (Абстрактное описание) | +  (Более подробное описание, уровни взаимодействия должны совпадать с диаграммой пакетов) | +  (Полное описание прецедента с указанием вызываемых методов, используемых классов и объектов). |  | - |
| Sequence Diagram | +  (Абстрактное описание) | +  (Более подробное описание, уровни взаимодействия должны совпадать с диаграммой пакетов) | +  (Полное описание прецедента с указанием вызываемых методов, используемых классов и объектов). |  | - |
| Cooperative Diagram | +  (Абстрактное описание) | +  (Более подробное описание, уровни взаимодействия должны совпадать с диаграммой пакетов) | +  (Полное описание прецедента с указанием вызываемых методов, используемых классов и объектов). |  | - |
| Package Diagram | - | + | - |  | - |
| Data Base Diagram | - | - | +  (Полная ER модель базы данных + её даталогическая модель) |  | - |
| Deployment Diagram | - | - | - |  | +  (Подробная диаграмма развертывания с указанием характеристик машин и интерфейсов взаимодействия) |
| Timeline diagramm |  |  |  | + |  |

#### **3.** **Architectural Goals and Constraints (Цели и ограничения архитектуры)**

Система должна быть защищена от несанкционированного доступа извне и поддерживать разделение ролей пользователе. Для это в системе используется JWT-токен. Решаемые JWT-токеном задачи:

● Аутентификация. Пользователь, для того чтобы пользоваться web-приложением должен быть зарегистрирован в данной системе.

● Разделение ролей. JWT токен содержит логин пользователя и роль.

● Автоматическая проверка валидности токена. При каждом запросе происходит валидация токена, поэтому имеется дополнительная “безопасность” запроса. Данная проверка происходит сторонней библиотекой (для нас это “+”).

● Простота использования и наличие литературы.

## 3.2 Постоянство данных

Хранение данных будет производиться с использованием реляционной базы данных - PostgreSQL.

Все данные хранятся на сервере и будут хранить резервные копии, которые позволяют в момент непредвиденной ошибки откатить систему к предыдущей версии. Для создания резервных копий данных, произведенных в результате работы программы, полученные базы данных сохраняются с помощью команды “**pg\_dump**”. Для сохранения конфигурации postgres будет написан скрипт, копирующий конфигурацию в отдельное хранилище. Запуск скрипта будет производиться вместе с командой создания резервной копии.

Программа и ее исходный код хранится с помощью истории версий в git системе. При обновлении версии программы происходит коммит нового кода, а старый записывается в виде предыдущей версии.

## 3.3 Качество и устойчивость

Система использует логирование для отслеживания и выявления ошибок при работе программы.

**Kotlin sfl4j wrapper**

2 ) Серверная часть системы, работающая на Spring Boot, логирует следующие события через движок Slf4j:

* Входящие и исходящие HTTP-запросы;
* Ошибки обработки запросов, включая исключения;
* Время обработки запросов и производительность сервисов;
* Ошибки в работе с базой данных;
* Важные события на сервере (например, перезапуск или обновление сервиса);
* Информационные сообщения, связанные с работой системных сервисов.

Систематическое логирование на серверной стороне позволяет выявлять не только явные ошибки, но и потенциальные проблемы с производительностью и нагрузкой, что способствует повышению устойчивости всей системы.

## 3.4 Производительность и нагрузка

Требования к производительности и нагрузке, указанные в документах Vision и SRS должны учитываться при разработке системы.

**3.5 Разработка**

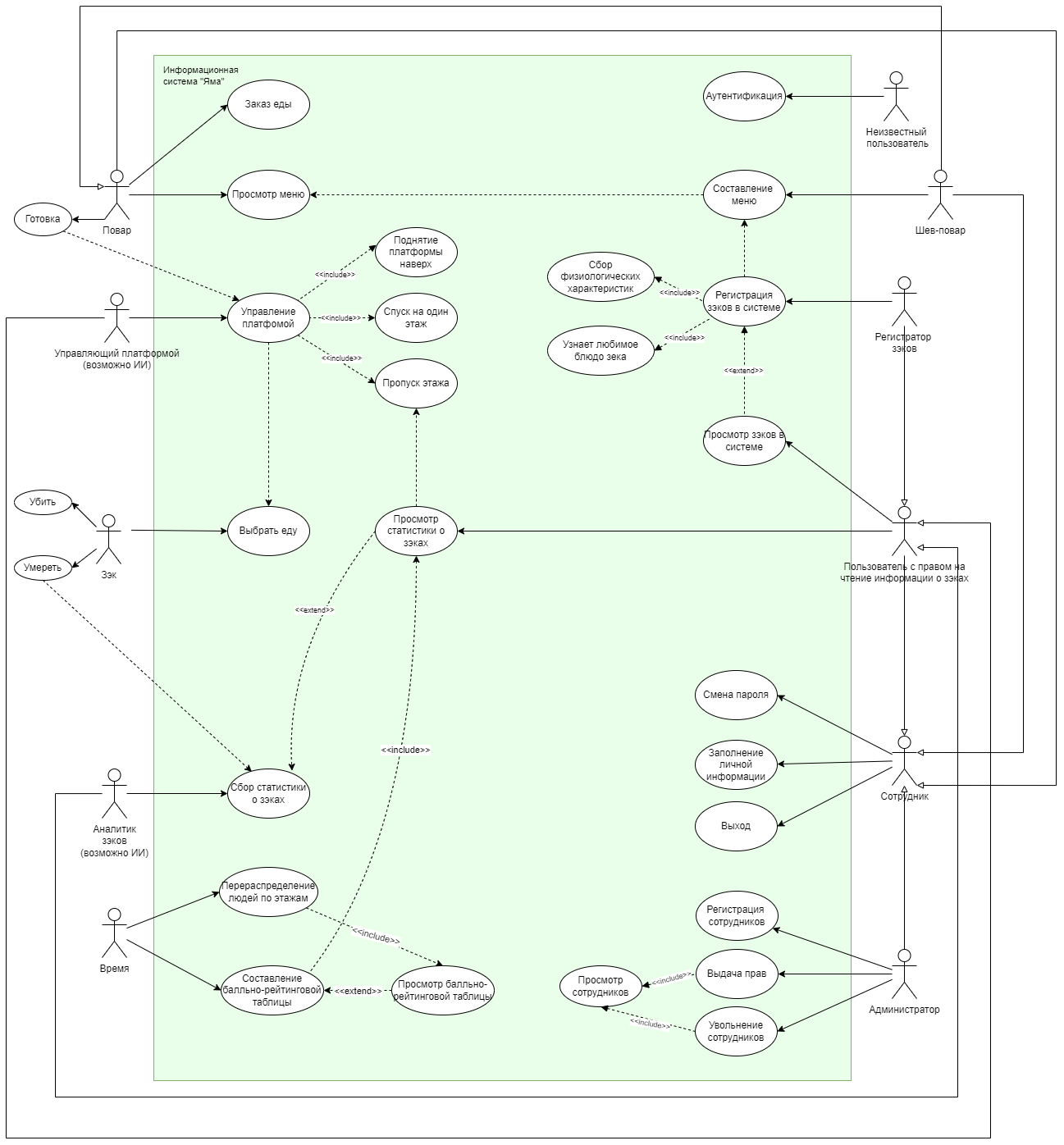
При разработке важно учитывать множество аспектов, чтобы создать качественный и эффективный пользовательский интерфейс и работу программы.

**3.6 Тестирование**

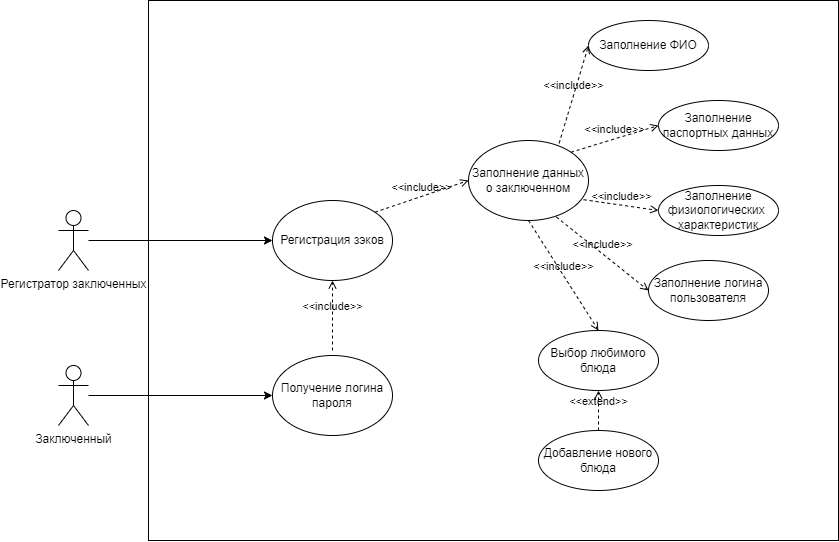
Тестирование системы включает несколько ключевых этапов:

1. Функциональное тестирование – проверка корректности работы всех функций и бизнес-логики системы.
2. Приемочное тестирование – проверка соответствия системы требованиям и ожиданиям пользователей.
3. Нагрузочное тестирование – оценка поведения системы под высокой нагрузкой для выявления узких мест и предельной производительности.
4. Модульные тесты (unit-тесты) – тестирование отдельных функций и компонентов системы в коде для раннего выявления ошибок и автоматизации процесса проверки.
5. Интеграционные тесты (E2E) – проверка взаимодействия всех компонентов системы в рамках реальных сценариев использования, охватывающая как фронтенд, так и бэкенд, для выявления возможных проблем интеграции.

#### **4.** **Use-Case View**

Use Case Diagram

##### Use Case diagram UC «Регистрация заключенных в системе»:



##### 

##### Class diagram UC «Управление баллами»



##### Activity Diagram UC «Составление меню для заключенных в соответствии с диетическими и пищевыми требованиями»:

##### State Machine Diagram UC «Управление местоположение платформой»:

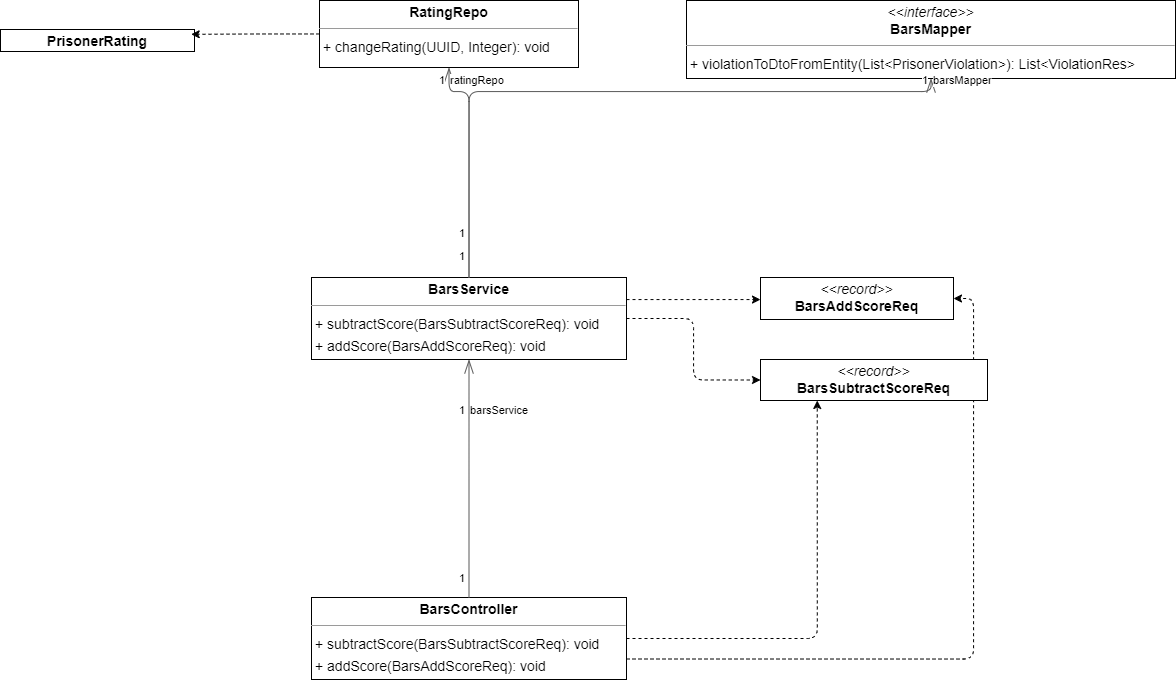
##### Sequence Diagram UC «Управление баллами, начисляемыми и списываемыми у заключенных за их поведение и участие в программах реабилитации» :

##### Collaboration Diagram «Отображение меню для заключенных на доступных платформах»:

##### 

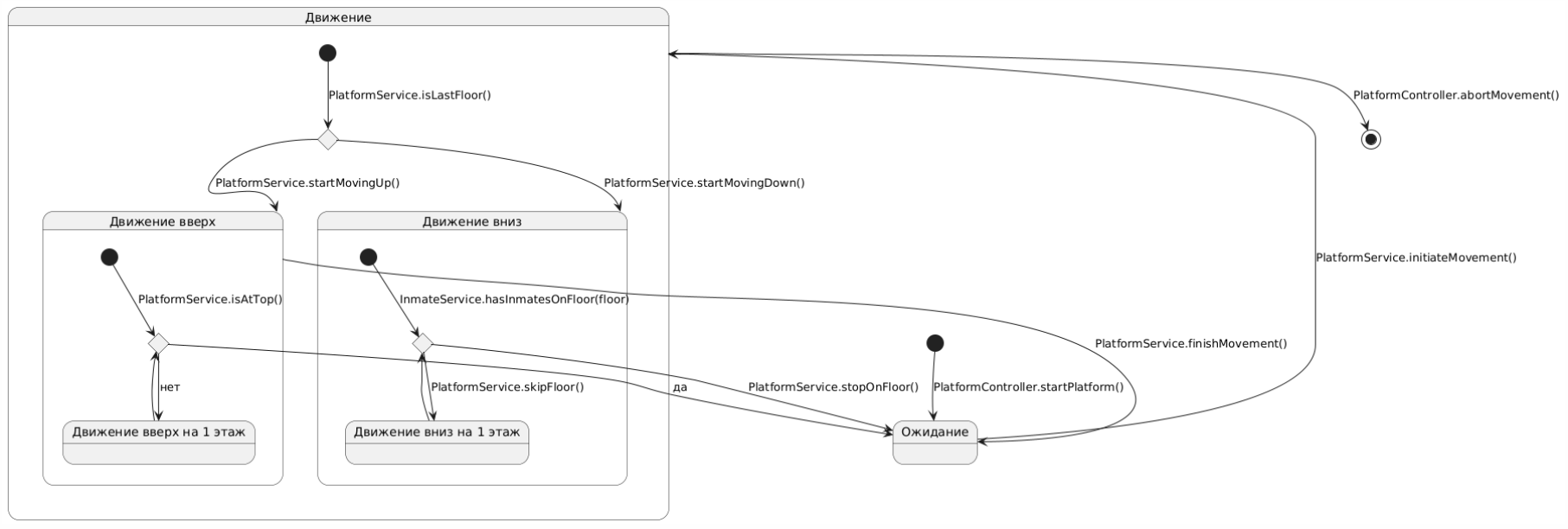
#### **5.** **Logical View**

##### Class diagram UC «Управление баллами»

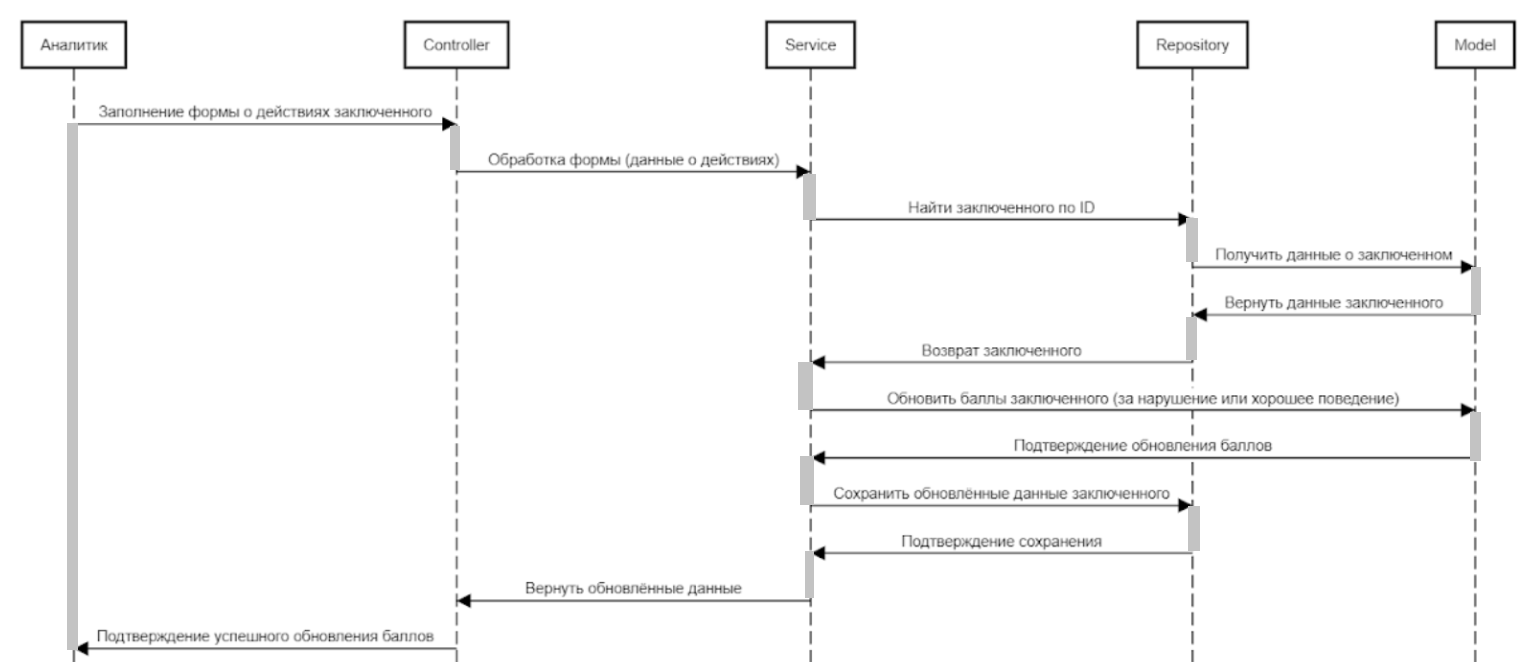


##### Activity Diagram UC «Составление меню для заключенных в соответствии с диетическими и пищевыми требованиями»:

##### State Machine Diagram UC «Управление местоположение платформой»:



##### Sequence Diagram UC «Управление баллами, начисляемыми и списываемыми у заключенных за их поведение и участие в программах реабилитации» :



##### Collaboration Diagram:

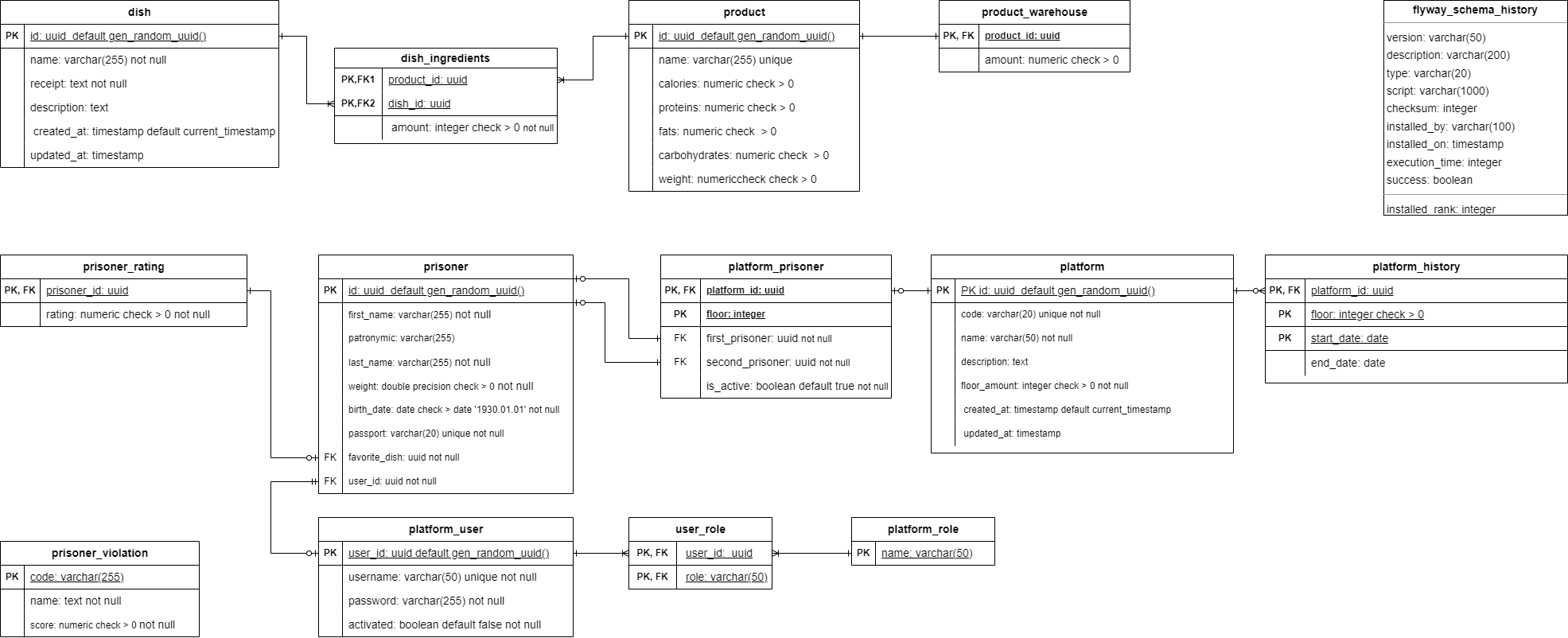
##### Package Diagram UC «Заказ продуктов для приготовления»:

#### **6.** **Process View**

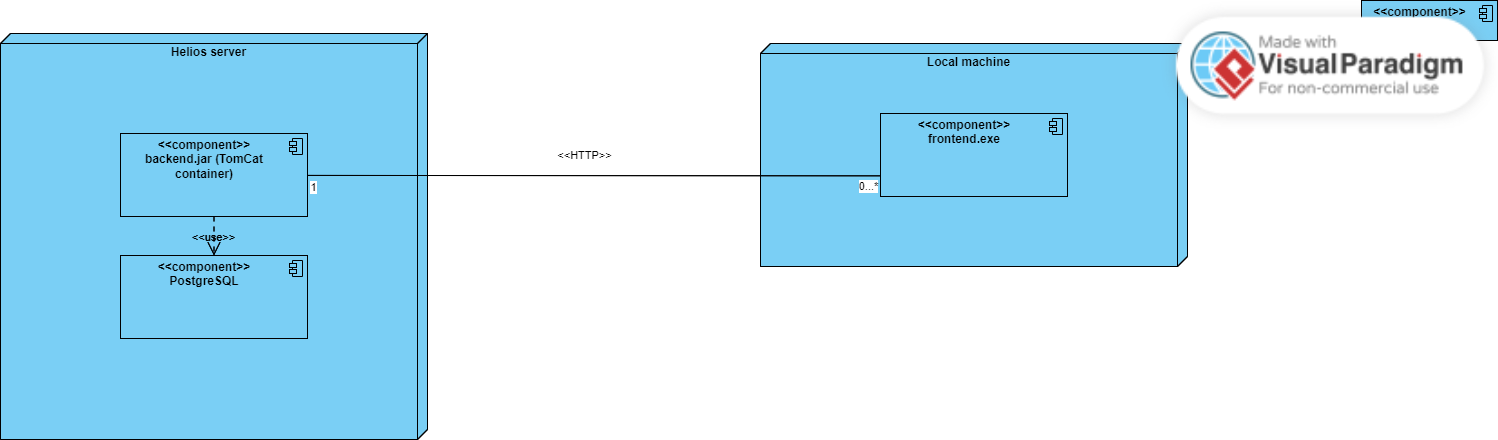
##### Инфологическая модель:

##### 

##### Даталогическая модель:

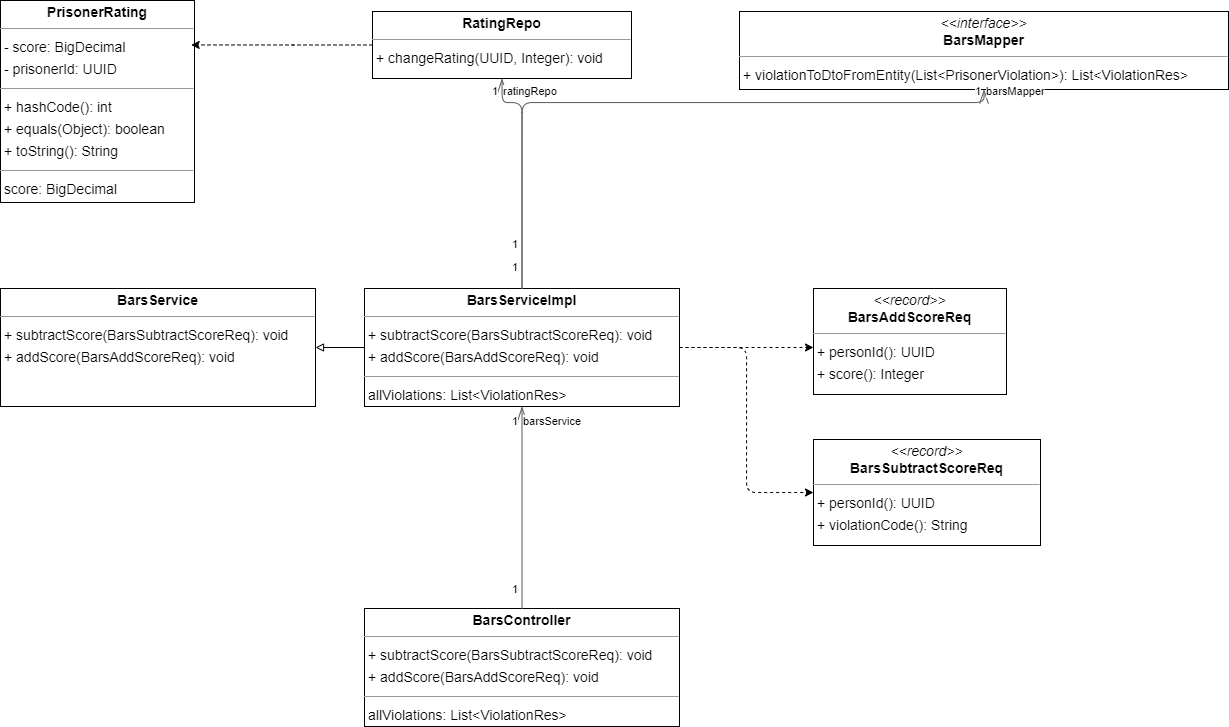


#### **7.** **Deployment View**



#### **8.** **Implementation View**

##### Class diagram UC «Управление баллами»



##### Sequence Diagram UC «Управление баллами, начисляемыми и списываемыми у заключенных за их поведение и участие в программах реабилитации»:

##### Activity Diagram UC «Составление меню для заключенных в соответствии с диетическими и пищевыми требованиями»:

##### State Machine Diagram UC «Управление местоположение платформой»:

##### Collaboration Diagram «Отображение меню для заключенных на доступных платформах»:

#### **9.** **Size and Performance (Производительность)**

Разрабатываемое десктоп приложение будет разработано согласно архитектуре MVI. Основная обработка будет производиться на сервере, а отображение результатов будет выполнено с помощью десктопного клиента, разработанного с помощью kotlin-фреймворка compose-multiplatform.

Следующие характеристики приложения будут подвергнуты измерению производительности:

* Производительность сервера. Для анализа производительности сервера будет использоваться инструмент Gatling (performance tests). При разработке будет использоваться инструмент профилирования среды IntelliJ Idea - IntelliJ Profiler. Для взаимодействия с сервером будут спроектированы маршруты, благодаря которым действия клиента не потребуют дополнительной подготовки (1 маршрут - 1 действие, дополнительные запросы производиться не будут).
* Безопасность. Для защиты от инъекций будет настроена валидация фильтров, которыми будут насыщаться нативные SQL запросы. Будут использованы библиотеки для аутентификации и авторизации (JWT) для обеспечения безопасности маршрутов и доступа к данным.
* Работа с базой данных. Работа с базой данных будет происходить через Jooq и flyway миграции для удобной и безопасной работы. Для ускорения работы с БД будут настроены индексы и триггеры.
* Логирование. Для логирования ошибок будет использована библиотека Log4j. При критических ошибках (отсутствует доступ к системе у пользователей) будет производиться анализ файла с логами. Также каждую среду и пятницу с 17:00-18:00 будет производиться анализ логов вне зависимости от наличия критической ошибки.
* При разработке прототипа интерфейса, было выбрано средство прототипирования Figma с использованием тулкита material3 из списка стандартных библиотек инструмента.

#### **10.** **Quality (Качество)**

* Надежность. Система разрабатывалась с учетом потребности устойчивости к сбоям, для этого производится сбор и аудит логов системы (при выявлении критической ошибки и каждую рабочую среду и пятницу с 17:00-18:00). Проводятся регулярные резервные копии базы данных для предотвращения потери данных в случае сбоя. Для создания бэкапа будет использоваться инструмент pg\_dump, а при обновлении конфигурации системы новая конфигурация будет размещена в git репозитории.
* Мобильность и платформонезависимость. Приложение будет выполнено в виде связки back и front частей, из-за чего работа с ним возможна на устройствах с операционными системами Linux, macOS, Windows и выходом в Интернет. Десктопный клиент, адаптирован под разные разрешения экрана с использованием адаптивного дизайна.
* Конвертация данных. Для передачи данных между частями системы (back & front) будет использоваться средство конвертации в формат JSON.
* Удобство и эффективность разработки. Используется популярный стек технологии: фреймворки Spring boot 3 и Kotlin Compose, которые упрощают разработку и обслуживание приложения. Применяются лучшие практики разработки, включая модульное тестирование и систему контроля версий.
* Версионирование API. Сервер будет поддерживать несколько версий API (текущая и 1 прошлая) для сохранения совместимости.