

Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Жизненный цикл разработки программного обеспечения

ПРИЛОЖЕНИЕ ДЛЯ ОТСЛЕЖИВАНИЯ ВИНИЛОВЫХ ПЛАСТИНОК  
«VinTrack»

Студент гр. 350504

Амбросевич Р.П.

Проверил:

Внук О.М.

Минск 2025

## **СОДЕРЖАНИЕ**

<b>ВВЕДЕНИЕ.....</b>	<b>3</b>
<b>1 ТРЕБОВАНИЯ ПОЛЬЗОВАТЕЛЯ.....</b>	<b>4</b>
1.1 Программные интерфейсы.....	4
1.2 Интерфейс пользователя .....	4
1.3 Характеристики пользователей .....	5
1.4 Предположения и зависимости .....	5
<b>2 СИСТЕМНЫЕ ТРЕБОВАНИЯ.....</b>	<b>7</b>
2.1 Функциональные требования .....	7
2.2 Нефункциональные требования .....	7
<b>3 ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ .....</b>	<b>8</b>
3.1 Глоссарий.....	8
3.2 Акторы системы .....	9
3.3 Описание потока событий.....	10
3.4 Диаграмма компонентов .....	15
3.5 Диаграмма развертывания .....	15
<b>4 ТЕСТИРОВАНИЕ УЧЕБНОГО ПРОЕКТА .....</b>	<b>17</b>
4.1 Введение.....	17
4.2 Объект тестирования .....	17
4.3 Риски.....	18
4.4 Аспекты тестирования.....	18
4.5 Подходы к тестированию .....	18
4.6 Критерии приёмки .....	19
4.7 Тестовые сценарии.....	19
4.7 Вывод.....	21

## **ВВЕДЕНИЕ**

VinTrack – веб-приложение для учёта личной коллекции виниловых пластинок. Актуальность проекта обусловлена тем, что коллекционеры часто ведут учёт в таблицах или заметках: такие решения плохо масштабируются, не поддерживают единые справочники и не предупреждают о дубликатах. VinTrack предоставляет централизованный инструмент с понятной моделью данных и удобным интерфейсом для повседневной работы.

Цель приложения – дать пользователю быстрый способ добавлять пластинки в свою коллекцию, назначать им состояние и находить нужные записи по нескольким критериям.

Область проекта: проект охватывает разработку как клиентской (front-end), так и серверной (back-end) частей приложения. Back-end будет реализован в виде распределённой микросервисной архитектуры. В проект не входит создание мобильных приложений для iOS или Android, а также реализация офлайн-функциональности.

Практическая ценность VinTrack — снижение операционных ошибок (дубликаты, несогласованность записей), ускорение рутинных операций (добавление, пометки, фильтры) и формирование единой структуры данных, пригодной для дальнейшей аналитики (например, динамика покупок по жанрам или годам). Проект служит базой для последующего расширения: добавления отчётов, импорта/экспорта, интеграций с публичными каталогами и автоматизации рекомендаций.

# 1 ТРЕБОВАНИЯ ПОЛЬЗОВАТЕЛЯ

## 1.1 Программные интерфейсы

Продукт будет взаимодействовать со следующими внешними и внутренними системами:

- PostgreSQL: СУБД для хранения данных о пользователях и коллекциях.
- Spring Boot (Java): серверная часть для реализации REST API.
- Nginx: Веб-сервер, который будет обслуживать статические файлы клиентской части.
- Hibernate (JPA): ORM для работы с базой данных.
- React + Vite + MUI: фронтенд для интерфейса.
- Nx Monorepo – организация кода фронтенда.

## 1.2 Интерфейс пользователя

Система будет взаимодействовать с пользователем через веб-сайт. Основные элементы интерфейса и их поведение:

- Главная страница: предоставление каруселей с конкретными виниловыми пластинками, например одного года, а также показ рекомендованных пластинок.
- Навигационная панель: интерфейс, содержащий лого, ссылки на другие страницы, а также окно поиска.
- Профиль пользователя: раздел для просмотра добавленных из каталога пластинок и управления статусами данных пластинок.
- Формы для CRUD-операций: интерфейсы для создания, изменения и удаления пластинок, связей с пользователями и пользователями.

Мокап интерфейса представлен на рисунке 1.1.

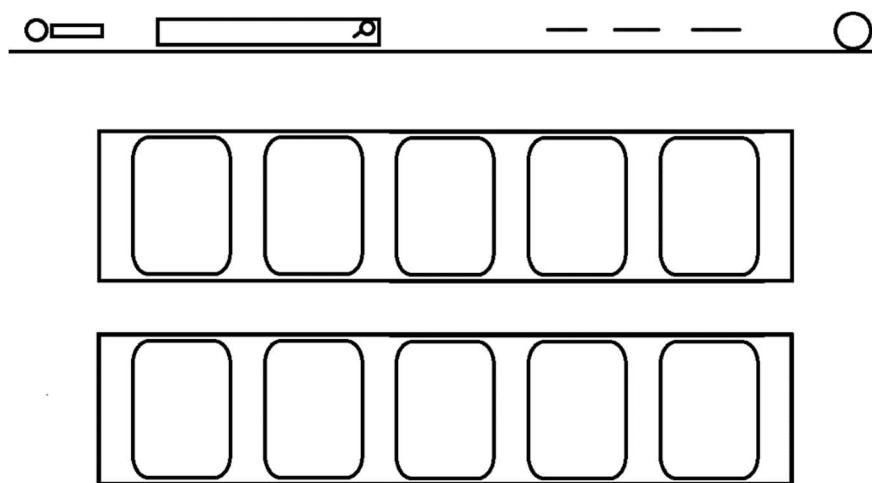


Рисунок 1.1 – мокап главной страницы

## **1.3 Характеристики пользователей**

Пользователей приложения можно классифицировать по нескольким параметрам.

1. Основная группа: любители музыки и коллекционеры виниловых пластинок.

2. Уровень образования: от среднего до высшего (особого значения не имеет)

3. Техническая грамотность: базовый пользователь ПК и веб-приложений.

4. Ожидания: удобный интерфейс, быстрый поиск по коллекции, возможность делиться своими данными.

## **1.4 Предположения и зависимости**

В рамках текущей версии VinTrack предполагается, что доступ к системе осуществляется через современный веб-браузер с включённым JavaScript и поддержкой HTML5. Пользователь работает онлайн; устойчивое интернет-соединение необходимо не только для первичной загрузки интерфейса, но и для всех операций с коллекцией, так как данные хранятся на сервере и передаются по REST-API. Рекомендуется использование HTTPS: это задаёт зависимость от корректной настройки TLS-сертификатов и времени на клиентах (расхождение системного времени может влиять на проверку токенов).

Серверная часть развёртывается в среде с установленной Java и доступом к экземпляру PostgreSQL. Приложение ожидает наличие подключения к базе данных и соответствия её схемы миграциям. При несогласованной схеме приложение может не стартовать. Конфигурация передаётся через переменные окружения (строка подключения к БД, секрет для подписи JWT, порты, CORS-политика). Корректная работа также зависит от доступности файловой системы/облака для логов, если включено журналирование.

Клиентская часть собирается инструментами фронтенда и публикуется как статические ресурсы; при этом её функционирование полностью зависит от доступности REST-эндпоинтов бэкенда и согласованности контрактов (форматы JSON, коды ответов, пагинация, поля фильтров). Любые изменения API требуют синхронного обновления фронта. Хранение токена на клиенте (в памяти приложения или в localStorage/cookie по выбранной политике) накладывает требования к настройке домена и CORS; блокировка третьих-сторонних cookie или жёсткие политики контента (CSP) на стороне организации пользователя могут нарушать авторизацию.

Производительность и задержки зависят от нескольких факторов: пропускной способности и стабильности сети между браузером и сервером, параметров хостинга (число CPU/память, тип диска, лимиты соединений к PostgreSQL), эффективной настройки пулов соединений и кэширования на уровне приложения/БД, а также от объёма клиентских данных (размер

страниц, количество одновременно отображаемых элементов). При росте нагрузки система предполагает горизонтальное масштабирование бэкенда и вынос БД на отдельный управляемый сервис; без этих шагов время отклика может превысить целевые значения.

## **2 СИСТЕМНЫЕ ТРЕБОВАНИЯ**

### **2.1 Функциональные требования**

- Система должна предоставлять регистрацию и аутентификацию пользователей.
- Система должна поддерживать CRUD-операции (создание, чтение, обновление, удаление) для записей о пластинках.
- Пользователь должен иметь возможность назначать статус пластинки («в наличии», «хочу», «в пути»).
- Должна быть реализована фильтрация и поиск по исполнителю, альбому, статусу.
- Интерфейс должен работать в браузере на устройствах с современными версиями Chrome, Firefox, Edge.

### **2.2 Нефункциональные требования**

- Надёжность: система должна обеспечивать целостность данных даже при сбоях (ORM + транзакции).
- Безопасность: аутентификация и авторизация должны защищать доступ к чужим коллекциям (JWT токены).
- Удобство использования: интерфейс должен быть интуитивно понятным (MUI компоненты).
- Производительность: время отклика при запросе не должно превышать 1 секунды при нормальной нагрузке.
- Масштабируемость: архитектура должна позволять подключать новые модули.
- Поддерживаемость: проект структурирован в виде монорепозитория, что упрощает поддержку и развитие.

## **3 ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ**

### **3.1 Глоссарий**

В данном разделе представлен глоссарий предметной области системы VinTrack, сформированный на основе анализа требований и модели вариантов использования.

Глоссарий служит основой для построения объектной модели: каждый термин отражает ключевую сущность или устойчивое понятие, которое далее будет реализовано в виде класса, объекта или компонента.

Пользователь (User) – зарегистрированный участник системы, ведущий учёт собственных виниловых пластинок. Имеет учётную запись и роль доступа.

Администратор (Admin) – пользователь с расширенными полномочиями: управляет справочниками (жанры), при необходимости редактирует карточки пластинок и выполняет операции администрирования.

Роль (Role) – тип доступа пользователя к функциям системы (USER, ADMIN). Используется для разграничения прав.

Аутентификация (Auth/JWT) – процесс проверки учётных данных и выдачи токена доступа (JWT) для последующих запросов к API.

Пластинка (Vinyl) – карточка музыкального релиза: название, артист, год издания, каталожный номер, ссылка на жанр. Базовая единица каталога.

Жанр (Genre) – справочник музыкальных жанров. Используется для категоризации пластинок и фильтрации.

Позиция коллекции (UserVinyl) – элемент личной коллекции пользователя, связывающий конкретную пластинку с владельцем и хранящий пользовательские атрибуты (статус, заметки, дата добавления).

Составной ключ позиции (UserVinylId) – идентификатор позиции коллекции, включающий userId и vinylId; обеспечивает уникальность пары пользователь–пластинка.

Статус позиции (VinylStatus) – состояние пластинки в контексте коллекции пользователя: WANTED, IN\_TRANSIT (в пути), OWNED (в наличии).

Каталог (Catalog) – совокупность карточек Vinyl и справочников, доступных для добавления в личную коллекцию.

Коллекция пользователя (User Collection) – множество объектов UserVinyl, принадлежащих одному пользователю; основной рабочий список в интерфейсе.

Поиск и фильтрация (Search/Filter) – механизм выборки элементов коллекции по атрибутам пластинки (артист, название, жанр) и по статусу; поддерживает пагинацию.

Панель пользователя (Dashboard) – стартовый экран после входа, отображающий текущую коллекцию, фильтры и операции над позициями.

Диаграмма классов представлена на рисунке 3.1.

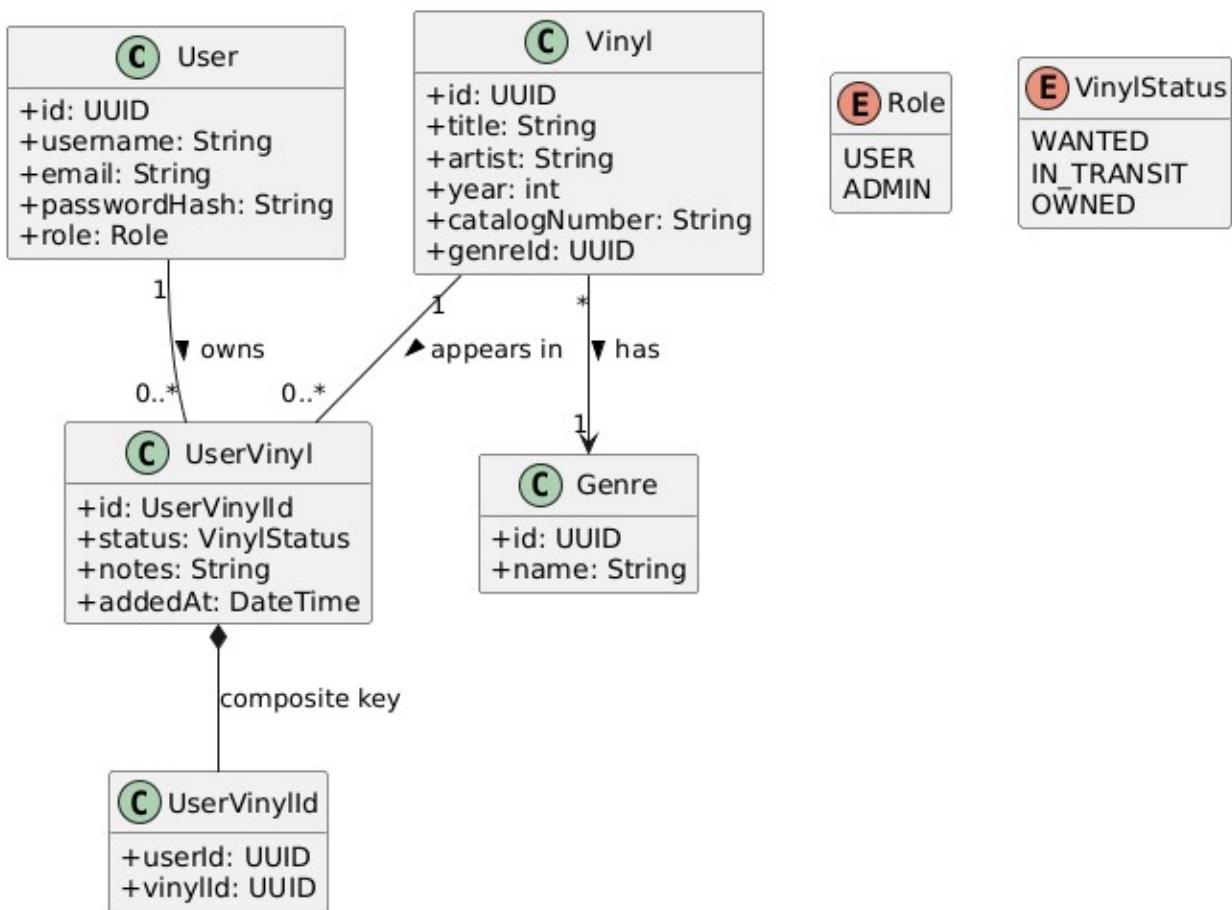


Рисунок 3.1 – Диаграмма классов

### 3.2 Акторы системы

В данном разделе определены основные акторы (роли взаимодействия) в рамках модели прецедентов VinTrack. Описание акторов задаёт границы системы и распределение функционала для последующего построения диаграмм вариантов использования.

**Guest** (Гость) – неаутентифицированный пользователь. Может регистрироваться и входить в систему. Доступ к данным коллекций отсутствует.

**User** (Пользователь) – аутентифицированный актор. Управляет собственной коллекцией: добавляет пластинки, изменяет статус, редактирует заметки, выполняет поиск/фильтрацию, удаляет позиции.

**Admin** (Администратор) – актор с расширенными правами. Управляет справочниками (жанры), при необходимости ведёт карточки Vinyl, контролирует корректность данных.

System (Система) – автоматические процессы: валидация данных, проверка дубликатов UserVinyl, применение правил переходов статусов, управление сессиями/токенами (JWT), обработка ошибок и транзакций.

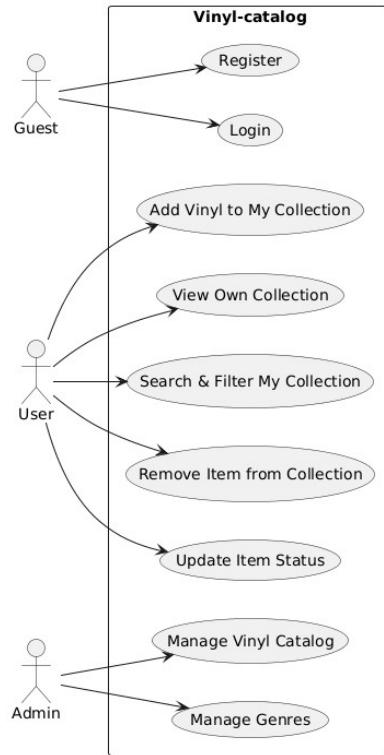


Рисунок 3.2 – Use-Case схема

### 3.3 Описание потока событий

#### 3.3.1 Регистрация

Регистрация создаёт новую учётную запись и открывает доступ к личной коллекции. Актор – гость; предполагается, что он не авторизован и находится на странице регистрации. Для данного прецедента основной поток выглядит следующим образом:

1. Открыть форму регистрации и ввести требуемые данные.
2. Нажать “зарегистрироваться”.
3. Отправить POST /auth/register.
4. Сервер проверяет уникальность электронной почты и имени пользователя, хэширует пароль и создает пользователя с базовой ролью.
5. Вернуть статус 201. Пользователь получает уведомление об успешной регистрации и переходит на форму входа.

К альтернативным или ошибочным вариантам относятся:

1. Некорректные данные на клиенте – форма подсвечивает поля, запрос не уходит.
2. Ошибка со статусом 409 – показать сообщение и предложить сменить значение

3. Сбой сети или сервера – уведомить и предоставить кнопку повтора.

### 3.3.2 Вход

Вход аутентифицирует пользователя и выдаёт JWT для доступа к защищённым операциям. Гость вводит учётные данные на странице логина. Основной поток:

1. Ввести электронную почту и пароль.
2. Нажать “авторизация”.
3. Отправить POST /auth/login.
4. Сервер валидирует данные, формирует JWT и возвращает статус 200 с токеном.

5. На стороне клиента сохраняется токен и открывается панель коллекции.

К альтернативным или ошибочным вариантам относятся:

1. Некорректные данные на клиенте – форма подсвечивает поля, запрос не уходит.
2. Проблемы времени или хранилища токена – пользователь уведомляется и получает предложение войти повторно.

### 3.3.3 Просмотр своей коллекции

Пользователь просматривает список собственных виниловых пластинок в своем профиле. Система считает его авторизованным и допущенным к endpoint. Основной поток:

1. Со стороны клиента запрашивается GET /me/collection?page=N&size=M, где N и M – некие значения.

2. Сервер возвращает страницу элементов с вложенной информацией о виниловых пластинках.

3. На стороне клиента совершается рендеринг таблицы, карточки виниловых пластинок и их статусов.

К альтернативным или ошибочным вариантам относятся:

1. Пустая коллекция – отобразить пустое состояние.
2. Истекший или некорректный токен – вернуть статус 401 и вернуться к странице логина.

### 3.3.4 Добавление пластинки в свою коллекцию

Пользователь добавляет выбранную виниловую пластинку в собственную коллекцию. Предполагается авторизация и доступ к каталогу. Основной поток:

1. Выбрать на нужную виниловую пластинку и нажать на нее.
2. В появившемся окне нажать “Добавить”.
3. Отправить POST /add?vinylId=N&statusId=2, где N – идентификатор выбранной пластинки.
4. Вернуть статус 201, на стороне клиента в профиле добавляется пластинка со статусом “Имею”.

К альтернативным или ошибочным вариантам относятся:

1. Можно без нажатия на карточку виниловой пластинки нажать на небольшую кнопку добавления в угол карточки.

2. Сбой БД – уведомление и просьба повторить действие.

Диаграмма активностей представлена на рисунке 3.3, диаграмма последовательности представлена на рисунке 3.4.

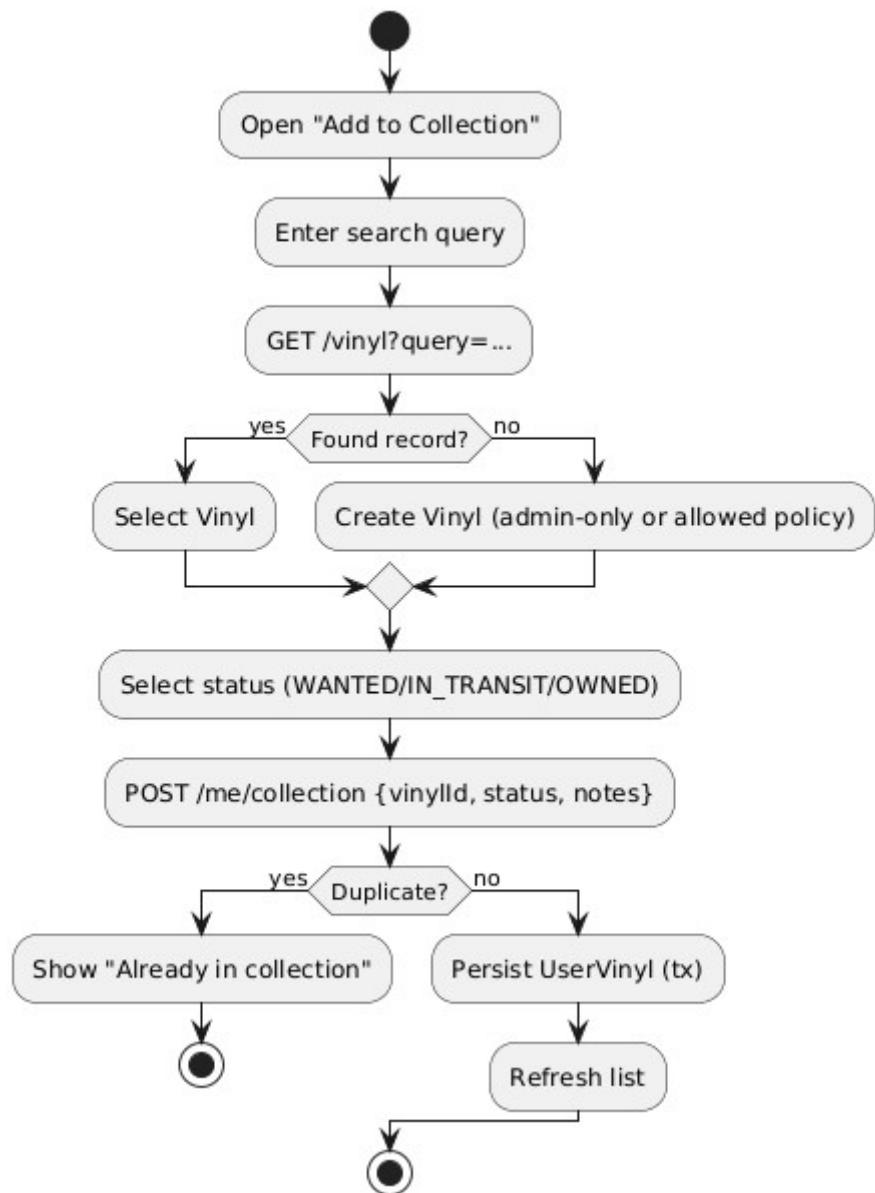


Рисунок 3.3 – Диаграмма активностей

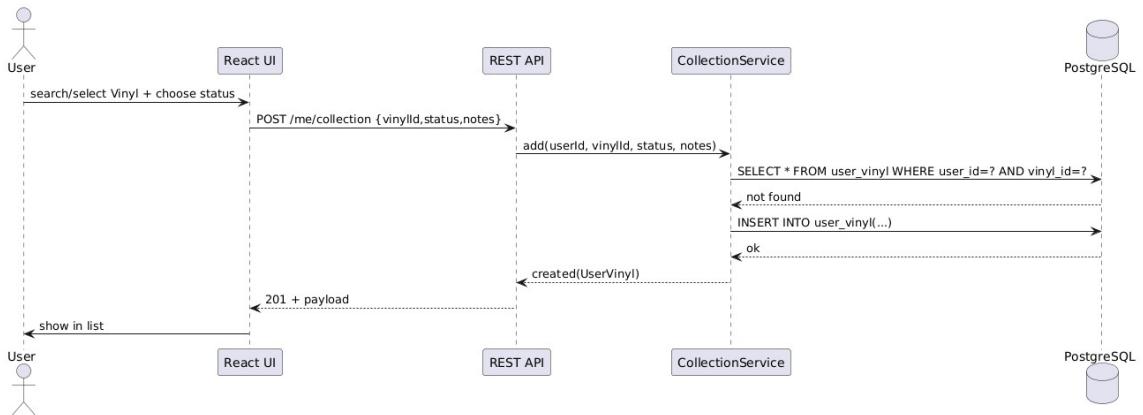


Рисунок 3.4 – Диаграмма последовательности

### 3.3.5 Изменить статус позиции

Пользователь меняет статус у виниловой пластинки у себя в профиле. Всего существует 3 статуса: “Имею”, “Хочу”, “В пути”. Основной поток:

1. В меню элемента выбрать новый статус.
2. Отправить PATCH /me/collection/vinylId=N&statusId=M, где N и M – идентификаторы пластинки и статуса соответственно.
3. Обновить запись и вернуть статус 200 с актуальными данными.
4. На стороне клиента обновляется плашка на карточке с показом нового статуса.

К альтернативным или ошибочным вариантам относятся:

1. Элемент не найден (например, был удален конкурентно) – выдать ошибку со статусом 404 и обновить список.
2. Конфликт версий – выдать ошибку со статусом 409, перезагрузить элемент и предложить повторить действие.

Диаграмма активностей представлена на рисунке 3.5, диаграмма состояний представлена на рисунке 3.6.

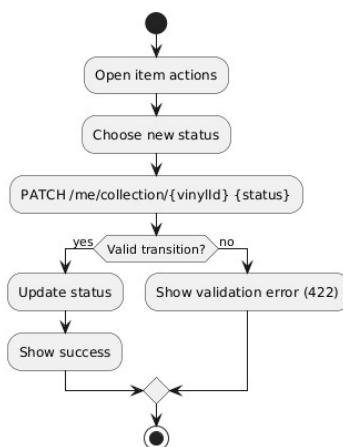


Рисунок 3.5 – Диаграмма активностей

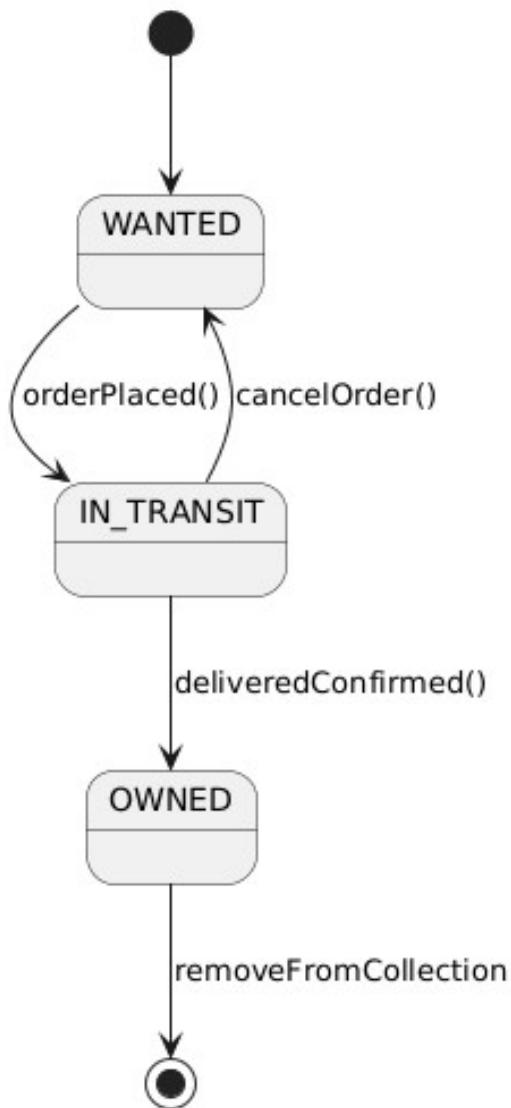


Рисунок 3.6 – Диаграмма состояний

### 3.3.6 Удалить позицию из своей коллекции

Действие удаляет связь пользователя с выбранной виниловой пластинкой. Пользователь авторизован и видит элемент в списке. Основной поток:

1. Выбрать пластинку, которую пользователь хочет удалить из своего профиля.
2. Нажать на крестик, который находится в углу карточки.
3. Отправить `DELETE /me/collection/vinylId=N`, где N – идентификатор виниловой пластинки.
4. Сервер удаляет связь пользователя и виниловой пластинки, возвращается статус 204.

5. На стороне клиента удаляется элемент из списка.

К альтернативным или ошибочным вариантам относятся:

1. Пользователь отменяет подтверждение – изменений нет.

2. Сетевой или серверный сбой – уведомление, состояние не меняется.

### 3.4 Диаграмма компонентов

Компонентная схема показывает разбиение по подсистемам:

1. React + Vite + MUI(Frontend) – SPA с прокси и арі на бэкенд.
2. REST API (Spring Boot) – контроллеры, сервисы, фильтрация аутентификации.
3. Auth (JWT) – генерация и валидация токенов.
4. Domain Services — Collection Service, Catalog Service.
5. Persistence – PostgreSQL.

Диаграмма компонентов представлена на рисунке 3.7.

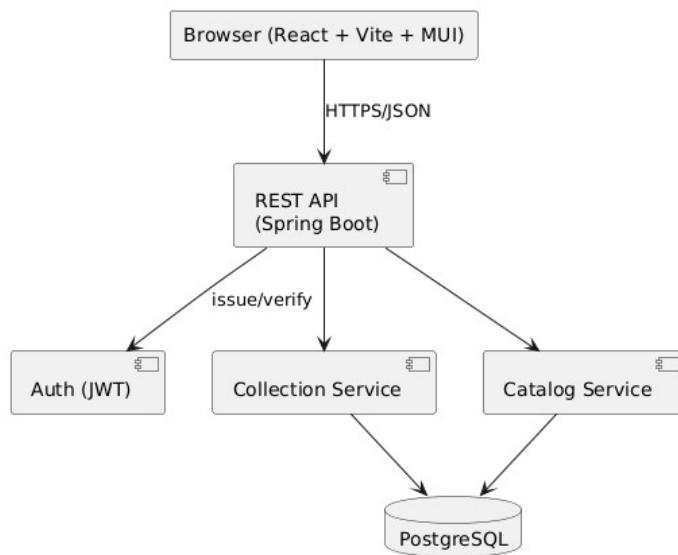


Рисунок 3.7 – Диаграмма компонентов

### 3.5 Диаграмма развертывания

Развертывание предполагает:

1. User Device: веб-браузер;
2. Front Host: статические сборки React (Vite);
3. Backend Host: контейнер Spring Boot (JDK) + контейнер PostgreSQL; взаимодействие по HTTP(S) и JDBC. Возможна контейнеризация (Docker Compose) и вынесение БД в управляемый сервис.

Диаграмма развертывания представлена на рисунке 3.8.

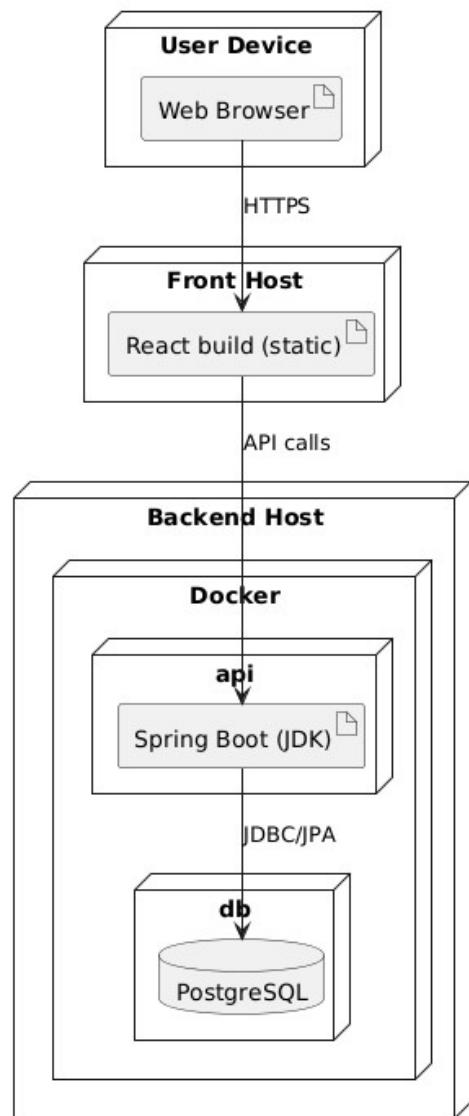


Рисунок 3.8 – Диаграмма развертывания

## **4 ТЕСТИРОВАНИЕ УЧЕБНОГО ПРОЕКТА**

### **4.1 Введение**

Цель тест-плана. Обеспечить качество веб-приложения VinTrack (учёт личной коллекции винилов) за счёт систематического тестирования функциональных и нефункциональных требований.

Область тестирования. Полное тестирование бэкенд-API (Spring Boot 3), точек интеграции с БД PostgreSQL и механизмов безопасности (JWT), а также проверка фронтенда (React + Vite + Nx) на корректное взаимодействие с REST API: регистрация/вход, работа с каталогом Vinyl, справочником Genre, управлением своей коллекцией (UserVinyl) и фильтрацией.

### **4.2 Объект тестирования**

Краткое описание проекта:

Веб-приложение для ведения личной коллекции виниловых пластинок. Пользователь регистрируется/входит, добавляет пластинки в свою коллекцию, задаёт их статус, ведёт заметки и фильтрует список.

Основные компоненты:

1. Бэкенд: Spring Boot 3.4.2, Spring Web, Spring Data JPA, Spring Security (JWT), OpenAPI.
2. Фронтенд: React 19, Vite, Nx, MUI.
3. База данных: PostgreSQL.
4. Аутентификация: JWT.

Атрибуты качества:

1. Функциональная корректность – точная реализация сценариев добавления/изменения/поиска.
2. Надёжность – устойчивость к ошибкам валидации и параллельным операциям.
3. Удобство использования – предсказуемые ответы API и читаемые сообщения об ошибках.
4. Безопасность – корректная авторизация по JWT, CORS, отсутствие несанкционированного доступа.
5. Производительность – отклик API  $\leq 1$  с для типовых запросов при нормальной нагрузке.

## **4.3 Риски**

Технические риски:

1. Несогласованность фронт/бэк по путям (например, /api префикс) и CORS (перенаправления на /login).
2. Неполная валидация данных при создании User, Vinyl, UserVinyl.
3. Ошибки управления статусами и гонки обновлений.
4. Неверная конфигурация безопасности (ошибки 302 вместо 401/403, неправильная обработка permitAll).

Бизнес-риски:

1. Потеря пользовательских данных коллекции при сбоях транзакций.
2. Дубликаты в коллекции (несоблюдение уникальности UserVinyl(userId, vinylId)).

Ограничения:

1. Отсутствуют Testcontainers/H2 в файле зависимостей – интеграционные тесты БД выполняются через моки/слайсы, либо на отдельном тестовом Postgres.
2. Ограничено время – приоритет high-risk/critical сценариев.

## **4.4 Аспекты тестирования**

Основной функционал:

1. Аутентификация/авторизация: /api/auth/register, /api/auth/login; доступ к защищённым ресурсам с JWT.
2. Каталог Vinyl: CRUD (админские операции) и выбор при добавлении в коллекцию.
3. Жанры (Genre): CRUD и привязка к Vinyl.
4. Коллекция пользователя (UserVinyl): добавление в коллекцию, смена статуса, удаление, запрет дубликатов.
5. Фильтрация/поиск: по артисту/названию/жанру/статусу; пагинация.
6. Безопасность: CORS, permitAll для /api/auth/\*\*, 401/403 вместо перенаправлений, корректная работа JwtAuthenticationFilter.
7. Техсервисы: кэш/логирование – корректность ключей и формата логов.

## **4.5 Подходы к тестированию**

Виды тестирования:

1. Модульное (Unit): JUnit 5 + Mockito. Тесты для сервисов: UserService, VinylService, GenreService, UserVinylService, VinylStatusService, фасада UserVinylFacade, утилит (CacheService, CacheKeyTracker, LogGenerationService).

2. Интеграционное (API-уровня): `@WebMvcTest/MockMvc` для контроллеров с подключением `spring-security-test` (проверка 200/201/401/403/409/422, CORS-preflight).

3. Системное (сквозные сценарии):

Smoke-набор: регистрация, вход, добавление в коллекцию, фильтрация, смена статуса, удаление.

Инструменты:

1. Бэкенд (покрытие): JUnit 5, Mockito, Spring Boot Test, Spring Security Test, JaCoCo.

2. API (ручное): Postman/HTTPie/curl.

3. Фронтенд (ручное): React DevServer + Vite proxy, проверка CORS/Network.

## 4.6 Критерии приёмки

1. Покрытие unit-тестами критического сервиса слоя  $\geq 80\%$  (JaCoCo).
2. Все high-priority тест-кейсы пройдены; нет critical/blocker дефектов.
3. Для `permitAll` маршрутов (`/api/auth/**`) ответы не перенаправляются на `/login`; ошибки – в формате 4xx/5xx без CORS-блокировок.
4. Уникальность `UserVinyl(userId, vinylId)` соблюдается; статусы меняются по правилам.

## 4.7 Тестовые сценарии

Ниже будут рассмотрены несколько групп сценариев тестирования:

1. Группа А: Аутентификация/авторизация

TC-AUTH-001 – Успешная регистрация

Цель: проверить создание пользователя.

Шаги: POST `/api/auth/register` с валидным `{username,email,password}`.

Ожидаемо: 201 Created, в БД создан User с ролью USER; тело содержит DTO пользователя.

TC-AUTH-002 – Валидация при регистрации

Цель: проверить ошибки формата/уникальности.

Шаги: отправить пароль короче минимума/дубликат email.

Ожидаемо: 400 Bad Request (валидация) или 409 Conflict (дубликат).

TC-AUTH-003 – Успешный вход

Шаги: POST /api/auth/login с корректными данными.

Ожидаемо: 200 OK, LoginResponse {token, role, username}.

TC-AUTH-004 – Доступ к защищённому ресурсу без токена

Шаги: GET /api/user/... без Authorization.

Ожидаемо: 401 Unauthorized (без перенаправления на /login),  
корректные CORS-заголовки.

TC-AUTH-005 – Доступ с просроченным/битым токеном

Ожидаемо: 401 Unauthorized, понятное сообщение от  
CustomAuthenticationEntryPoint.

2. Группа В: Каталог Vinyl и жанры

TC-VINYL-001 – Создание Vinyl (ADMIN)

Предусловие: роль ADMIN.

Шаги: POST /api/admin/vinyls с валидным телом.

Ожидаемо: 201 Created, карточка доступна для поиска/добавления в коллекцию.

TC-VINYL-002 – Дубликат каталожного номера

Ожидаемо: 409 Conflict (по принятой политике уникальности).

TC-GENRE-001 – Создание жанра (ADMIN)

Шаги: POST /api/admin/genres {name}.

Ожидаемо: 201; дубликат имени, выдаем 409.

3. Группа С: Коллекция пользователя (UserVinyl)

TC-UV-001 – Добавление в коллекцию

Шаги: POST /api/user/collection {vinylId, status?, notes?}.

Ожидаемо: 201, создан UserVinyl со статусом по умолчанию WANTED  
(или переданным).

TC-UV-002 – Запрет дубликатов

Шаги: повторный POST с тем же vinylId.

Ожидаемо: 409 Conflict и указание на существующую запись.

TC-UV-003 – Смена статуса (валидный переход)

Шаги: PATCH /api/user/collection/{vinylId} {status: IN\_TRANSIT}.

Ожидаемо: 200, статус обновлён.

TC-UV-004 – Смена статуса (недопустимый переход)

Ожидаемо: 422 Unprocessable Entity с сообщением о нарушении правил  
перехода.

TC-UV-005 – Удаление позиции

Шаги: DELETE /api/user/collection/{vinylId}.

Ожидаемо: 204 No Content.

#### 4. Группа D: Поиск/фильтрация

TC-SEARCH-001 – Фильтрация по артисту/жанру/статусу

Шаги:

GET/api/user/collection?artist=A&genre=G&status=OWNED&page=1&size=20.

Ожидаемо: 200, корректная пагинация и сортировка.

TC-SEARCH-002 – Пустой результат

Ожидаемо: 200 с пустой страницей.

#### 5. Группа E: Безопасность/CORS

TC-SEC-001 – Preflight для POST /api/auth/register

Шаги: OPTIONS с Origin: http://localhost:4200.

Ожидаемо: 200 + Access-Control-Allow-Origin: http://localhost:4200, методы/заголовки разрешены.

TC-SEC-002 – Запрет перенаправления на /login

Шаги: любой 401/403 сценарий.

Ожидаемо: статус 401/403 без Location: /login.

#### 6. Группа F: Техсервисы

TC-CACHE-001 – Формирование ключей кэша (CacheKeyTracker)

Ожидаемо: детерминированные ключи по входным данным (без коллизий).

TC-CACHE-002 – Поведение CacheService

Ожидаемо: читается из кэша при повторных обращениях; инвалидация сбрасывает записи.

TC-LOG-001 – Формат логов (LogGenerationService)

Ожидаемо: соответствие шаблону (включая идентификаторы запросов, уровни).

## 4.8 Вывод

По результатам выполнения набора тестов ожидается подтверждение, что VinTrack корректно реализует основные бизнес-сценарии: регистрация/вход, операции с собственной коллекцией, фильтрация и управление справочниками. Конфигурация безопасности обеспечивает доступ к публичным конечным точкам без перенаправлений, возврат корректных кодов ошибок и соблюдение CORS. Юнит-тесты покрывают ключевые сервисы и фасады; интеграционные тесты контроллеров подтверждают контракт REST. При достижении критериев из раздела 6 система может считаться готовой к учебной демонстрации и дальнейшему расширению (отчёты, импорт/экспорт, интеграции).