Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»

Отчет по междисциплинарному проекту «Bird»

Студент гр. РИМ-240950 Зверев А. Д.

Екатеринбург

2025

Оглавление

Проект Bird — учебное веб-приложение, представляющее упрощённый аналог социальной сети Twitter. Система построена на микросервисной архитектуре и включает два backend-сервиса (UMS и Twitter), отдельные базы данных для каждого сервиса, а также клиентское SPA-приложение. Основной акцент проекта — практическое объединение знаний по архитектуре, микросервисам, базам данных, безопасности и фронтенд-разработке, а также получение навыков контейнеризации и развёртывания.

Цель работы: разработать и продемонстрировать учебное веб-приложение «Bird» на микросервисной архитектуре, объединяющее практические навыки проектирования, работы с БД, разработки backend/frontend и развёртывания, включая безопасность.

Задачи:

1. Спроектировать микросервисную архитектуру
2. Реализовать API для пользователей/авторизации и для сообщений/подписок
3. Настроить безопасность: JWT, роли, GitHub OAuth, интроспекция токенов
4. Спроектировать и реализовать БД с миграциями Flyway
5. Разработать фронтенд (SPA) с сессией, роутингом и доступом по ролям
6. Обеспечить развёртывание и запуск через k8s

Требования и ограничения (функциональные и нефункциональные)

Функциональные требования

Система Bird должна обеспечивать:

- Регистрацию и вход пользователя по email/паролю через сервис UMS.
- Вход через GitHub OAuth (инициация авторизации, обработка callback, создание/привязка пользователя).
- Выдачу JWT-токена при успешной аутентификации и возможность его интроспекции (проверка валидности и получение ролей).
- Хранение и управление ролями пользователей (ADMIN, SUBSCRIBER, PRODUCER) и применение ролевого доступа к операциям.
- Создание сообщений (публикация) пользователем с ролью PRODUCER.
- Получение списка сообщений автора.
- Управление подписками: подписка/отписка подписчика на авторов, обновление списка подписок.
- Формирование ленты подписчика: получение сообщений авторов, на которых подписан пользователь (требуется роль SUBSCRIBER).
- Получение списка подписчиков автора (требуется роль PRODUCER).
- Предоставление веб-интерфейса (frontend) для логина, просмотра ленты, управления подписками и выполнения операций согласно ролям.

Нефункциональные требования

- Архитектура: микросервисная, с разделением ответственности на сервис пользователей/авторизации (UMS) и сервис контента/подписок (Twitter).
- Данные: отдельная база данных для каждого сервиса (database-per-service) на MySQL; управление схемой через миграции Flyway.
- Безопасность:
    - аутентификация на основе JWT;
    - проверка токенов в сервисе Twitter через интроспекцию в UMS;
    - авторизация по ролям на уровне API и UI;

    - возможность инвалидировать токены через ротацию пользовательского секрета.
- Развёртывание: поддержка локального запуска (Docker Compose для БД) и запуска в Kubernetes (манифесты, Ingress, Secrets, NetworkPolicy).

Общая архитектура и схема компонентов

1) Архитектурный подход

Bird реализован как микросервисная система, где ответственность разделена по доменным областям:

- **UMS** отвечает за пользователей и безопасность (аутентификация, роли, токены).
- **Twitter-сервис** отвечает за контент и социальные связи (сообщения и подписки).
- **Frontend** предоставляет единый пользовательский интерфейс и обращается к API сервисов.

Для каждого сервиса используется **собственная база данных**, что подчёркивает независимость сервисов и границы владения данными.

2) Состав компонентов

В системе выделяются следующие компоненты:

- **Frontend (React/Vite)** — SPA-приложение, работающее в браузере. Хранит JWT-сессию и выполняет запросы к API.
- **UMS (User Management Service)** — сервис регистрации/логина, управления ролями, выпуска JWT и GitHub OAuth. Также предоставляет интроспекцию токенов.
- **Twitter service** — сервис сообщений и подписок: создание/получение сообщений, управление подписками, формирование ленты подписчика.
- **MySQL (UMS)** — хранит пользователей, роли, секреты и связи OAuth-провайдера.
- **MySQL (Twitter)** — хранит сообщения, подписки и связанные сущности.
- **Ingress / reverse proxy (в Kubernetes)** — единая точка входа, публикующая фронтенд и проксирующая запросы к сервисам по префиксам.

3) Общая схема взаимодействия (логическая)

Потоки взаимодействия в системе можно представить так:

- **Пользователь (браузер) → Frontend**
- **Frontend → UMS**:
    - регистрация/логин;
    - получение токена;
    - (при необходимости) операции управления пользователями/ролями.
- **Frontend → Twitter service**:
    - запросы к сообщениям и подпискам;
    - получение ленты подписчика;
    - создание сообщений (для роли PRODUCER).
- **Twitter service → UMS**:
    - интроспекция JWT для проверки валидности токена и получения ролей;
    - запрос профиля пользователя для проверки роли перед выполнением операций.
- **UMS → MySQL (UMS)** (все операции с пользователями и авторизационными данными).
- **Twitter service → MySQL (Twitter)** (все операции с сообщениями и подписками).
- **UMS → GitHub** (OAuth2-логин).

4) Единый вход и адресация API

В сценарии развёртывания через Kubernetes используется единый домен и маршрутизация по путям:

- / → Frontend
- /api/ums/* → UMS (с переписыванием пути)
- /api/twitter/* → Twitter service (с переписыванием пути)

Это позволяет фронтенду работать с API на одном origin и упрощает конфигурацию клиентских запросов.

5) Роли компонентов в обеспечении безопасности

UMS выступает как **центр аутентификации**:

- выдаёт JWT и хранит данные, необходимые для его проверки;
- предоставляет интроспекцию токена, которую использует Twitter-сервис.

Twitter-сервис является **ресурсным сервисом**:

- принимает запросы только с валидным токеном;
- дополнительно проверяет права на бизнес-операции через обращение к UMS (роль SUBSCRIBER/PRODUCER).

Frontend реализует **клиентскую часть авторизации**:

- хранит токен/сессию;
- ограничивает доступ к разделам интерфейса в зависимости от ролей;
- отправляет токен в API через заголовок Authorization: Bearer <token>.

Backend-сервисы

Backend в проекте Bird состоит из двух микросервисов: **UMS** и **Twitter service**. Оба сервиса реализованы на **Java 21** и **Spring Boot (WebFlux)**, используют **MySQL** как хранилище данных и **Flyway** для миграций схемы. Взаимодействие сервисов между собой выполняется по HTTP.

1) UMS (User Management Service)

Назначение.
UMS — сервис управления пользователями и авторизации. Он является источником истины по идентификации пользователя и его ролям, отвечает за выпуск JWT-токенов, их проверку и интеграцию с GitHub OAuth.

Ключевые функции:

- регистрация и вход по email/паролю;
- выпуск JWT (в ответе возвращается токен, пользователь и время истечения);
- интроспекция токенов (проверка валидности и возврат ролей);
- ротация секрета пользователя (инвалидация ранее выданных токенов);
- управление пользователями и их ролями;
- OAuth-вход через GitHub с созданием пользователя.

Основные эндпоинты (API):

- POST /auth/register — регистрация пользователя и выдача JWT.
- POST /auth/login — вход и выдача JWT.
- POST /auth/introspect — интроспекция токена, ответ вида {active, sub, roles}.
- POST /auth/rotate-secret — ротация секрета текущего пользователя.
- POST /auth/rotate-secret/{user-id} — ротация секрета указанного пользователя (роль ADMIN).
- GET /users — список пользователей.
- GET /users/user/{user-id} — получить пользователя по id.
- POST /users/user — создать пользователя.
- PUT /users/user/{user-id}/roles — обновить роли пользователя.
- DELETE /users/user/{user-id} — удалить пользователя.

- GET /roles — справочник ролей.
- GET /oauth2/authorization/github — старт GitHub OAuth.

Безопасность доступа в UMS:

- публичные
  маршруты: /auth/register, /auth/login, /auth/introspect, /oauth2/**, /login/**, /actuator/**;
- остальные эндпоинты требуют Authorization: Bearer <token>.

Конфигурация (по умолчанию):

- порт сервиса: 9000;
- БД: MySQL ums (локально 3306);
- параметры JWT: issuer и ttl задаются в конфигурации;
- GitHub OAuth задаётся переменными окружения.

2) Twitter service (сообщения и подписки)

Назначение.
Twitter service — ресурсный сервис, отвечающий за сообщения и подписки. Он не управляет пользователями напрямую и не хранит их учётные данные, а использует UMS для проверки токенов и ролей.

Ключевые функции:

- хранение и выдача сообщений автора;
- создание сообщений (разрешено только пользователю с ролью PRODUCER);
- хранение подписок подписчика на авторов;
- выдача ленты подписчика (сообщения авторов, на которых он подписан; требуется роль SUBSCRIBER);
- выдача списка подписчиков автора (требуется роль PRODUCER).

Основные эндпоинты (API):
Сообщения:

- GET /messages/message/{message-id} — получить сообщение по id.
- GET /messages/producer/{producer-id} — сообщения автора.
- GET /messages/subscriber/{subscriber-id} — лента подписчика (проверка роли SUBSCRIBER).

- POST /messages/message — создать сообщение (проверка роли PRODUCER).
- DELETE /messages/message/{message-id} — удалить сообщение.

Подписки:

- GET /subscriptions/subscriber/{subscriber-id} — список подписок пользователя (роль SUBSCRIBER).
- GET /subscriptions/producer/{producer-id} — список подписчиков автора (роль PRODUCER).
- POST /subscriptions — создать подписки.
- PUT /subscriptions — обновить подписки.
- DELETE /subscriptions/subscriber/{subscriber-id} — удалить подписки пользователя.

Как обеспечивается безопасность в Twitter service:

- сервис работает как **Resource Server** и требует Bearer-токен на большинстве эндпоинтов (кроме /actuator/**);
- валидность токена проверяется через **интроспекцию в UMS**;

Конфигурация (по умолчанию):

- порт сервиса: 9001;
- БД: MySQL twitter (локально 3308);
- параметры подключения к UMS: host/port и пути интроспекции/получения пользователя;
- issuer JWT должен совпадать с UMS.

Безопасность (JWT, роли, интроспекция, GitHub OAuth)

В проекте Bird безопасность построена вокруг централизованного сервиса авторизации **UMS**, который выпускает и проверяет токены, хранит роли пользователей и реализует OAuth-вход через GitHub.
Сервис **Twitter** выступает ресурсным сервисом: он принимает запросы только с валидным токеном и проверяет права пользователя на операции.

1) JWT: модель аутентификации

JWT используется как основной механизм аутентификации для запросов к защищённым эндпоинтам UMS и Twitter. Клиент (frontend) после успешного входа получает токен и передаёт его в каждом запросе:

Authorization: Bearer <token>

Структура токена и claims:

- стандартные поля:
    - iss — идентификатор издателя (issuer);
    - iat — время выпуска;
    - exp — время истечения;
    - sub — идентификатор пользователя (UUID).
- пользовательские поля:
    - email — email пользователя;
    - roles — список ролей пользователя.

Особенность реализации:
В Bird используется подпись **HS256**, при этом **ключ подписи хранится на уровне пользователя**: у каждого пользователя есть собственный secret_key. Это позволяет точечно инвалидировать токены конкретного пользователя, но требует обращения к данным пользователя при проверке подписи.

2) Роли и авторизация

В системе выделены три роли:

- **ADMIN** — доступ к административным операциям (например, управление пользователями и ролями, ротация секрета другого пользователя).
- **SUBSCRIBER** — доступ к сценариям подписчика (управление подписками, получение ленты).

- **PRODUCER** — доступ к сценариям автора (публикация сообщений, просмотр подписчиков).

Где применяется контроль ролей:

- **Backend (обязательно):** сервисы проверяют роли перед выполнением операций.
- **Frontend (дополнительно):** интерфейс скрывает или блокирует недоступные действия и маршруты, но окончательное решение всегда остаётся за API.

3) Интроспекция токенов

Чтобы Twitter-сервис мог проверять токены, используется механизм **интроспекции** в UMS:

POST /auth/introspect с телом { "token": "<jwt>" }

UMS:

- проверяет токен (валидность подписи и срок действия);
- возвращает результат вида { active, sub, roles }.

Twitter:

- использует интроспекцию как основу доверия к токену;
- отклоняет запросы при active=false;
- применяет роли из ответа интроспекции для авторизации.

Такое решение демонстрирует централизованную безопасность в микросервисной системе: **UMS — источник истины по токенам и ролям**, а Twitter использует UMS для принятия решений.

4) GitHub OAuth

Помимо входа по email/паролю, в Bird реализован вход через **GitHub OAuth2**.

Поток OAuth:

1. Пользователь нажимает кнопку входа через GitHub во фронтенде.
2. Frontend направляет пользователя на UMS: GET /oauth2/authorization/github.
3. GitHub выполняет авторизацию и возвращает пользователя в UMS по callback-адресу.
4. UMS:

- получает профиль пользователя GitHub (provider user id);
- ищет существующую связь GitHub-аккаунта с пользователем системы;
- при отсутствии связи создаёт нового пользователя и сохраняет привязку.

5. UMS формирует ответ авторизации (JWT + данные пользователя) и делает редирект на фронтенд в формате:
    - /login?auth=<base64url(AuthResponse)>
6. Frontend декодирует параметр auth, сохраняет сессию и продолжает работу как для обычного логина.

5) Сессия на фронтенде и обработка ошибок

Фронтенд хранит полученную сессию (токен и время истечения) в localStorage или sessionStorage в зависимости от режима «запомнить меня». При получении ответа 401 Unauthorized приложение очищает сессию и переводит пользователя на страницу входа, предотвращая работу с недействительным токеном.

Таким образом, безопасность Bird включает полный цикл: получение токена (логин/регистрация/OAuth), хранение и использование токена, проверку токена в ресурсном сервисе через интроспекцию и контроль доступа по ролям на уровне API и интерфейса.

Базы данных

В проекте Bird используется подход **database-per-service**: каждый микросервис владеет своей базой данных MySQL и управляет её схемой независимо от других сервисов. Это помогает разделять ответственность и избегать прямых зависимостей между доменами на уровне данных.

1) Базы данных и разделение по сервисам

- **UMS DB (MySQL, схема ums)** — хранит пользователей, роли и данные, связанные с авторизацией.
- **Twitter DB (MySQL, схема twitter)** — хранит сообщения и данные подписок.

Сервисы не делают SQL-запросы к чужой базе данных: взаимодействие между доменами происходит через HTTP-API (Twitter обращается к UMS для проверки токенов/ролей).

2) Flyway как ключевой механизм управления схемой

Главный акцент в работе с БД — использование **Flyway миграций**. В обоих сервисах Flyway включён и применяется автоматически при запуске приложения. Это означает, что:

- схема БД создаётся и обновляется **версионированно** (через файлы миграций);
- развёртывание окружения становится повторяемым: достаточно поднять MySQL и запустить сервис — таблицы появятся сами;
- изменения структуры БД фиксируются как часть исходного кода и контролируются через историю миграций.

Миграции расположены в стандартном пути:

- ums/src/main/resources/db/migration/
- twitter/src/main/resources/db/migration/

Примеры версий:

- UMS: V1__init.sql, V2__auth.sql
- Twitter: V1__init.sql

3) Ключевые таблицы и связи

UMS (схема ums)

Основные сущности:

- users — пользователь (id, имя, email, пароль/хеш, секрет для токенов и т.п.).
- roles — справочник ролей (ADMIN, SUBSCRIBER, PRODUCER).
- users_has_roles — связь «многие-ко-многим» между пользователями и ролями.
- user_identities — привязки внешних провайдеров (GitHub OAuth) к пользователю.
- last_visit — вспомогательная таблица истории/времени входа (используется в выборках).

Twitter (схема twitter)

Основные сущности:

- messages — сообщения пользователя (контент до 140 символов, время создания).
- subscriptions — подписки (subscriber → producer).
- producers, subscribers — вспомогательные таблицы для идентификаторов участников.

Frontend

Frontend в проекте Bird — это SPA-приложение на **React + TypeScript**, собранное с помощью **Vite** и использующее **React Router** для навигации. Интерфейс выступает единой точкой работы пользователя с системой и взаимодействует с backend-сервисами UMS и Twitter через HTTP-запросы.

1) Маршруты и навигация

Маршрутизация организована в виде набора страниц, доступных в зависимости от наличия активной сессии и роли пользователя.

Основные маршруты:

- /login — страница входа (email/пароль и GitHub OAuth).
- / — главная страница (дашборд с лентой/сводкой), доступна только авторизованным.
- /console — «консоль» для работы с API (учебный интерфейс), доступна только авторизованным.
- /subscriptions — управление подписками (сценарий SUBSCRIBER).
- /messages — лента сообщений от подписок (сценарий SUBSCRIBER).
- /subscribers — просмотр подписчиков автора (сценарий PRODUCER).
- /admin — административная панель (только для роли ADMIN).
- /forbidden — страница отказа в доступе (например, при попытке открыть админ-панель без прав).

2) Хранение сессии и работа с JWT

После успешной аутентификации UMS возвращает объект сессии вида:

- token — JWT,
- expiresAt — время истечения,
- user — профиль пользователя и его роли.

Сессия хранится в браузере:

- при включённом «запомнить меня» — в localStorage,
- иначе — в sessionStorage.

При загрузке приложения:

- сессия читается из storage,
- если токен отсутствует или expiresAt истёк — сессия очищается.

Дополнительно реализована централизованная обработка 401 Unauthorized:

- при получении ответа 401 приложение выполняет logout, очищает сессию и переводит пользователя на /login с сообщением о необходимости войти заново.

3) Доступ по ролям

Роли пользователя (ADMIN, SUBSCRIBER, PRODUCER) приходят из UMS и используются:

- **для UI-логики** (что показать в меню и какие действия доступны);
- **для защиты критических разделов** (например, /admin).

Примеры:

- **SUBSCRIBER**: доступны разделы «Подписки» и «Сообщения», а также лента подписчика.
- **PRODUCER**: доступна публикация сообщений и раздел «Подписчики».
- **ADMIN**: доступна админ-панель (управление пользователями, ролями, ротация секрета).

4) Основные экраны и сценарии

- **Login**: вход по email/паролю и запуск GitHub OAuth. После успешного входа сессия сохраняется и выполняется переход в рабочую область.
- **Dashboard (Главная)**: сводная страница, где пользователь видит основные блоки (ленту, подсказки по ролям) и может выполнять действия в зависимости от ролей (например, публикация сообщения для PRODUCER).
- **Subscriptions**: управление подписками подписчика на авторов (добавление/удаление, обновление списка).
- **Messages**: лента сообщений от авторов, на которых подписан пользователь; реализован поиск/фильтрация по автору и содержимому.
- **Subscribers**: список подписчиков текущего автора (для роли PRODUCER).
- **Console / Admin**: «учебная консоль» для вызовов API UMS и Twitter и просмотра ответов; в режиме администратора доступны операции управления пользователями и ролями.

Развёртывание

Развёртывание Bird в Kubernetes организовано через набор манифестов в папке k8s/. Цель k8s-сценария — запустить все компоненты в изолированном namespace, обеспечить единый вход в приложение через Ingress, корректно хранить чувствительные данные через Secrets и ограничить сетевые взаимодействия между подами через NetworkPolicy.

1) Общая структура деплоя

Все ресурсы приложения размещаются в namespace **apps**. В кластере поднимаются:

- **MySQL для UMS** и **MySQL для Twitter** (StatefulSet + PVC для постоянного хранения данных),
- **UMS** и **Twitter service** (Deployment + Service),
- **Frontend** (Deployment + Service),
- **Ingress** для публикации фронтенда и API на одном домене,
- **Secrets** для паролей БД и параметров GitHub OAuth,
- **NetworkPolicy** для ограничения сетевого трафика (модель default deny).

2) Secrets (секреты БД и GitHub OAuth)

В k8s-контуре используются секреты для хранения чувствительных параметров:

- параметры подключения MySQL для UMS (root-пароль, имя БД, пользователь, пароль),
- параметры подключения MySQL для Twitter (root-пароль, имя БД),
- параметры GitHub OAuth для UMS (client id, client secret, redirect uri).

Secrets подключаются в контейнеры как переменные окружения, чтобы:

- не хранить пароли напрямую в Deployment-манифестах,
- упростить замену значений между окружениями

3) Ingress (единая точка входа)

Ingress публикует приложение на одном домене (в манифестах используется app.local) и маршрутизирует запросы по путям:

- / → сервис фронтенда (выдача статики),
- /api/ums/* → сервис UMS,
- /api/twitter/* → сервис Twitter.

Для API включено переписывание пути, чтобы сервисы получали запросы без префикса /api/.... Такой подход даёт:

- единый origin для браузера (упрощение работы фронтенда),
- отсутствие необходимости в отдельной CORS-настройке между фронтом и API,
- единообразные URL для внешнего доступа.

4) NetworkPolicy (сетевая изоляция и разрешённые потоки)

Для namespace apps применяется модель **“default deny”**: по умолчанию весь входящий и исходящий трафик запрещён, а необходимые взаимодействия разрешаются точечно.

Ключевые разрешённые потоки:

- **Ingress → Frontend / UMS / Twitter**: доступ извне допускается только через ingress-контроллер.
- **UMS → MySQL UMS**: доступ к базе разрешён только UMS-поду.
- **Twitter → MySQL Twitter**: доступ к базе разрешён только Twitter-поду.
- **Twitter → UMS**: разрешён трафик для интроспекции токенов и запросов профиля пользователя.
- **UMS → GitHub (443)**: разрешён исходящий HTTPS-трафик для OAuth.
- **DNS egress**: разрешены запросы к kube-dns, иначе сервисы не смогут резолвить имена.

Таким образом, NetworkPolicy фиксирует минимально необходимую сетевую связанность компонентов и демонстрирует практику сегментации микросервисов в кластере.

Выводы

В ходе выполнения проекта Bird была разработана учебная система, демонстрирующая полный цикл создания микросервисного веб-приложения: от проектирования архитектуры и модели данных до реализации backend-сервисов, фронтенда и вариантов развёртывания.

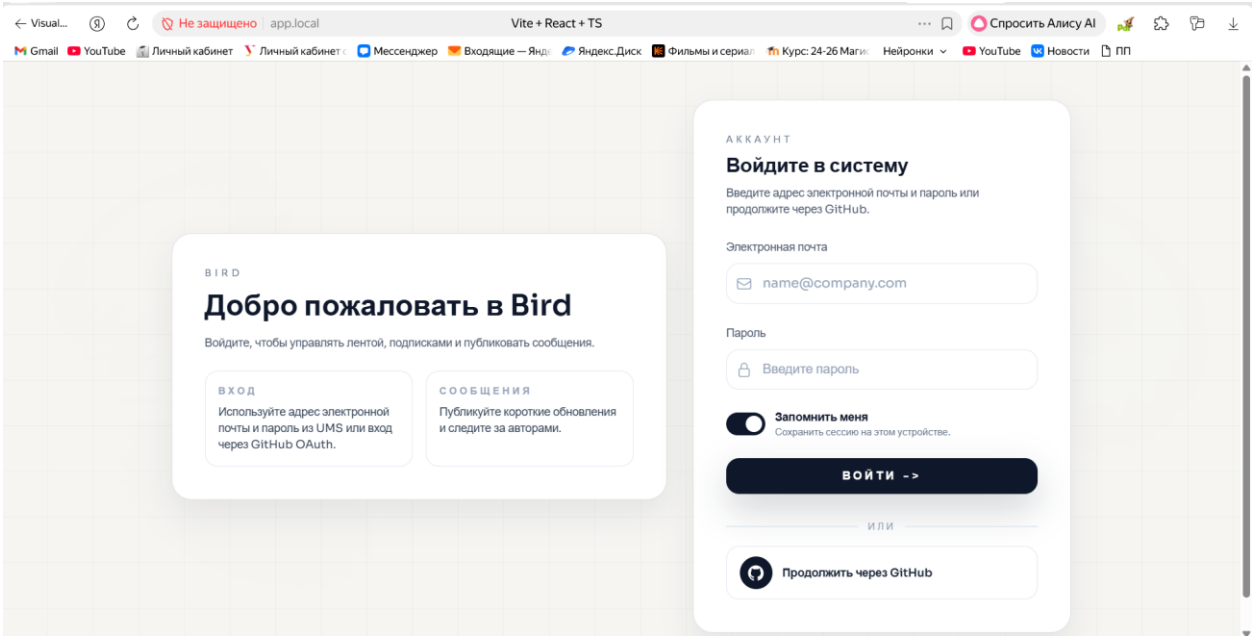
Основные результаты:

- Реализована микросервисная архитектура с разделением ответственности между сервисом авторизации и пользователей (UMS) и сервисом сообщений/подписок (Twitter), а также отдельным SPA-клиентом.
- Применён подход database-per-service: для каждого сервиса выделена отдельная база MySQL, что подчёркивает независимость сервисов и границы владения данными.
- Настроено управление схемой БД через Flyway-миграции, обеспечивающее повторяемый запуск и воспроизводимость окружения.
- Реализована модель безопасности на основе JWT с авторизацией по ролям и поддержкой GitHub OAuth; сервис Twitter проверяет валидность токенов через интроспекцию в UMS.
- Разработан фронтенд, включающий маршрутизацию, хранение сессии, ограничения доступа по ролям и основные пользовательские сценарии (вход, лента, подписки, управление).
- Подготовлены средства запуска и инфраструктурные элементы (в т.ч. Kubernetes-Ingress, Secrets и NetworkPolicy), что позволяет развернуть систему в едином контуре и ограничить сетевые взаимодействия между компонентами.

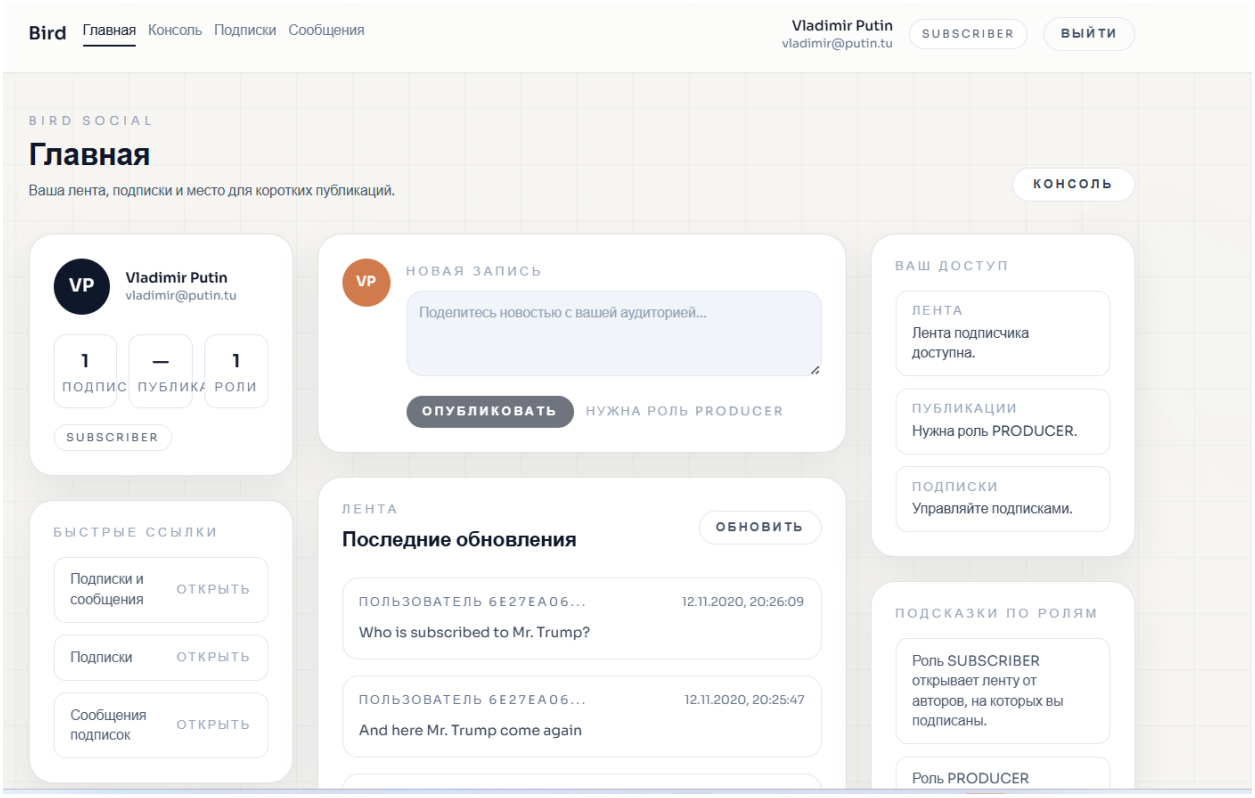
Таким образом закреплены навыки проектирования микросервисов, работы с БД и миграциями, реализации безопасного взаимодействия сервисов, разработки SPA-клиента и подготовки инфраструктуры для развёртывания.

Пример работы через UI

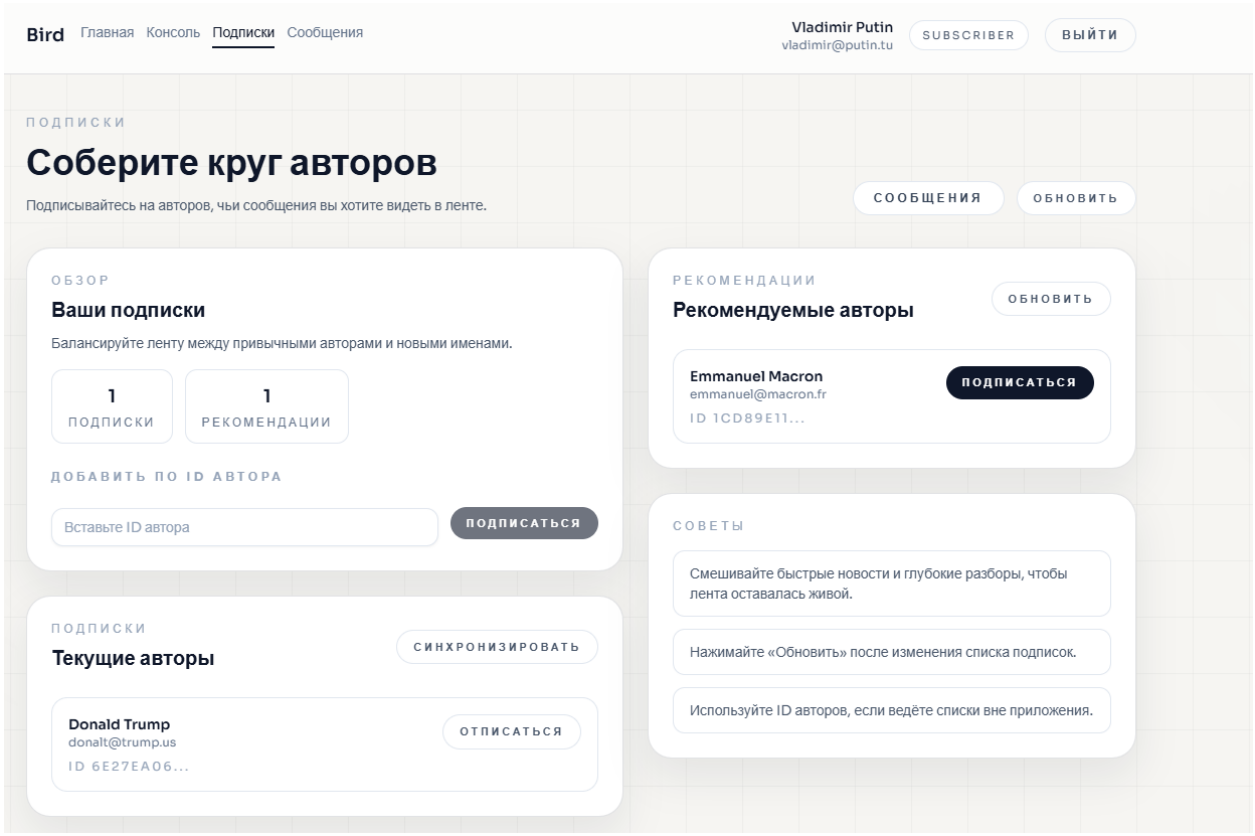
Страница логина



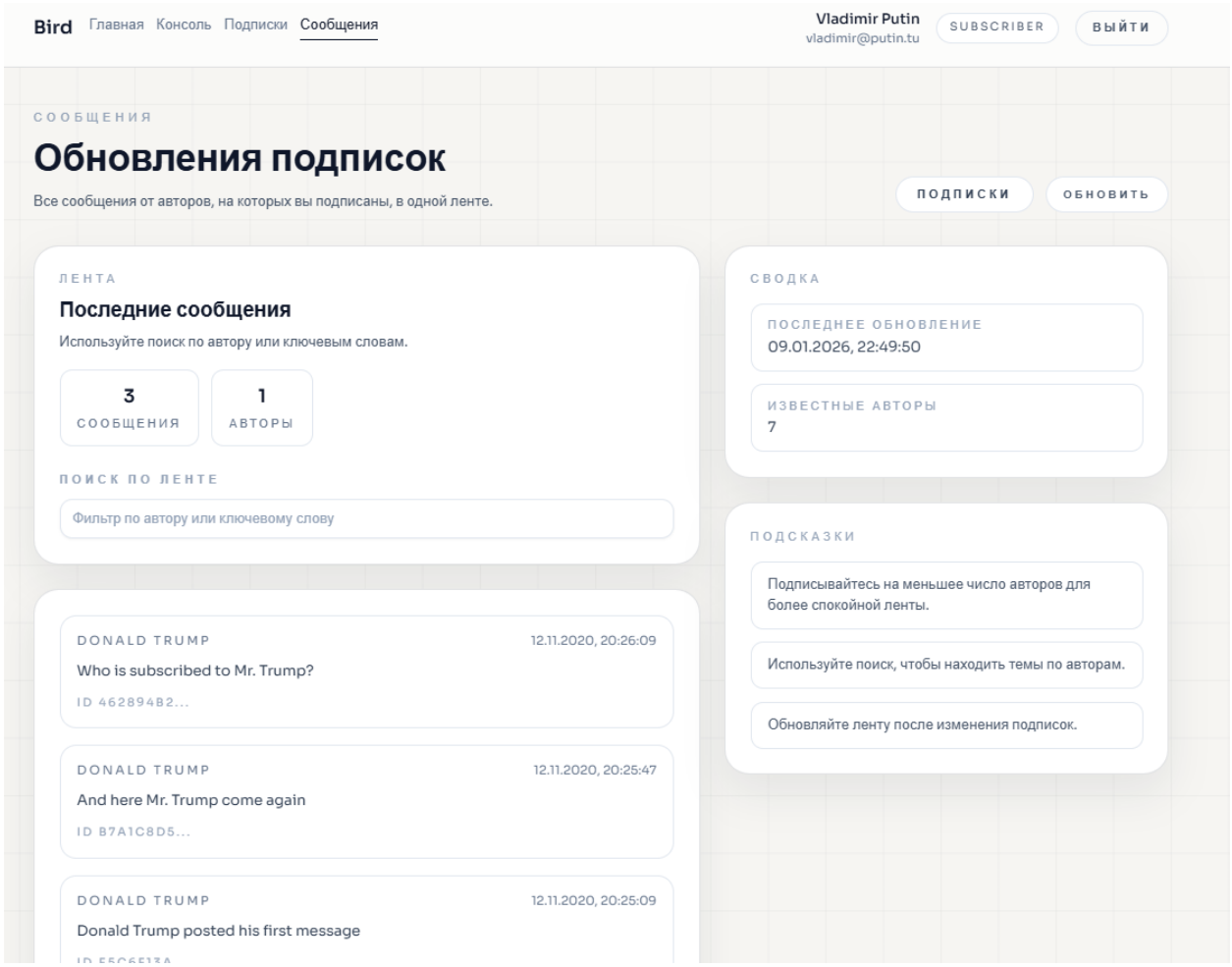
Главная страница с сообщениями



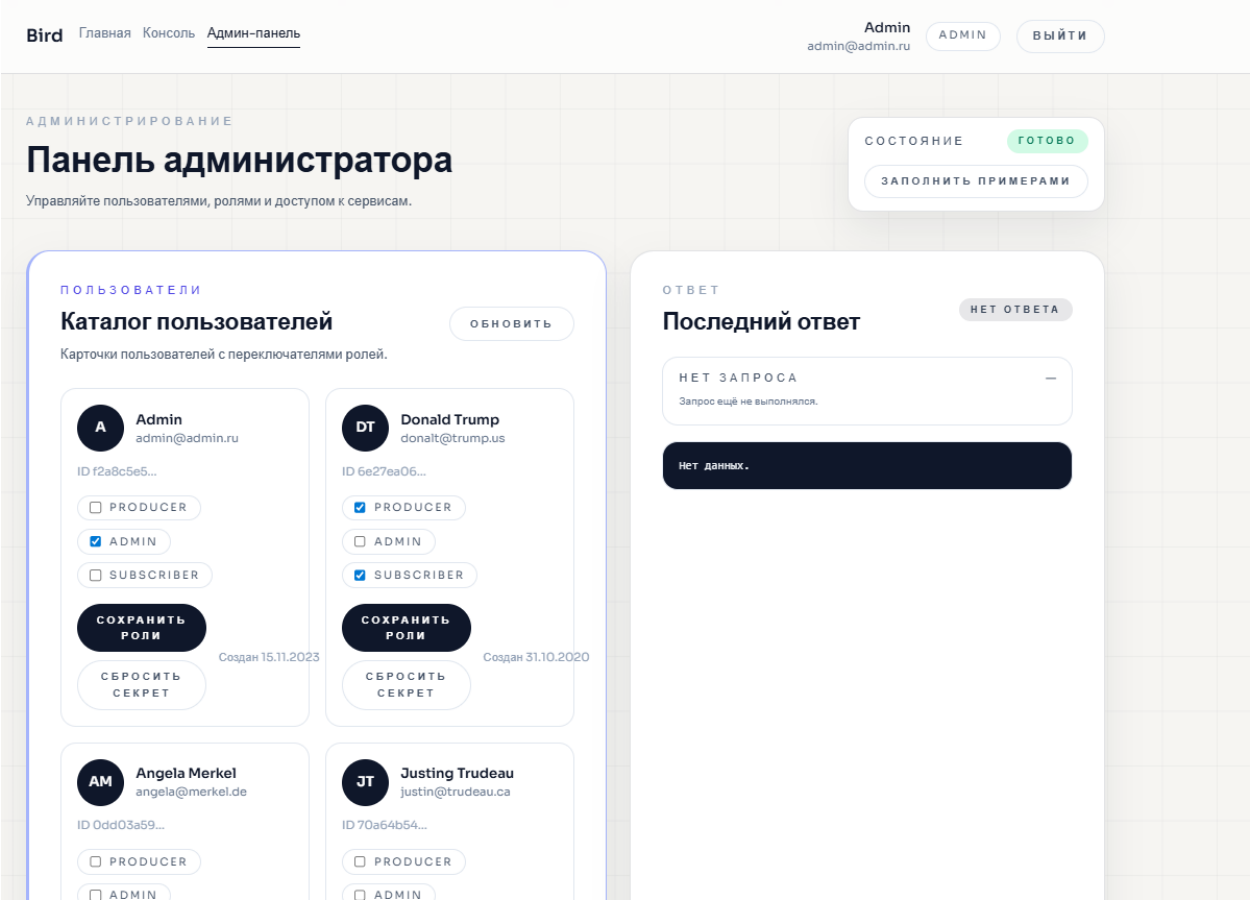
Страница подписок



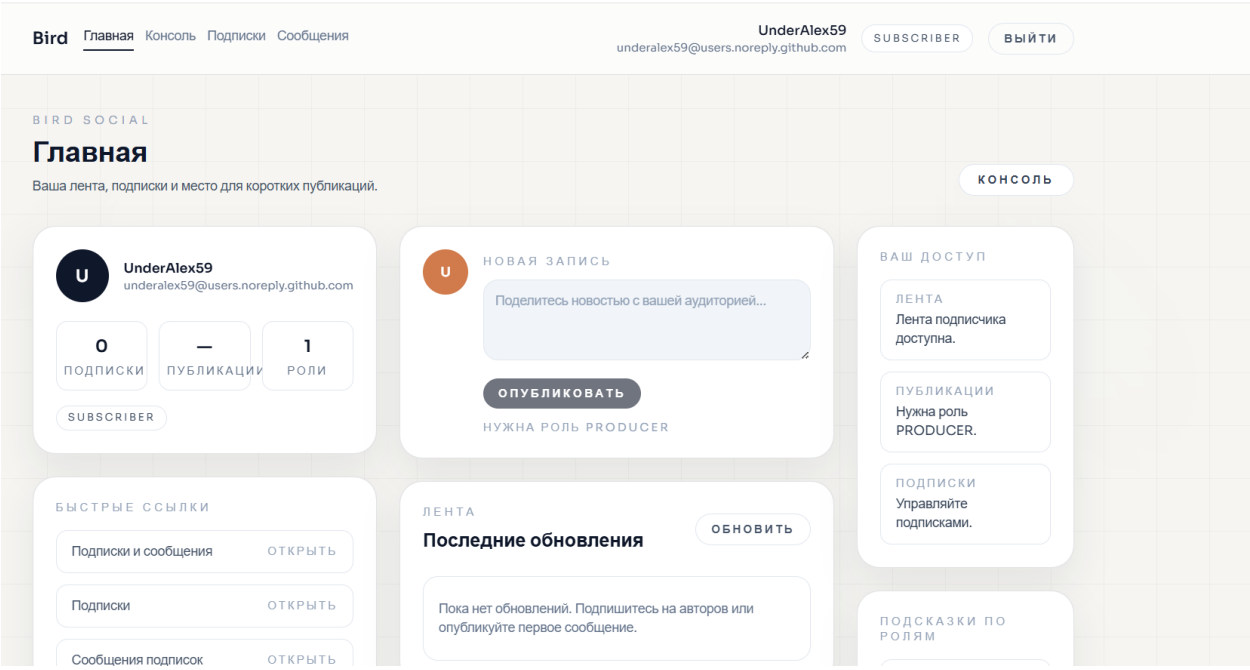
Страница сообщений



Админ панель с возможностью назначать роли и менять секреты пользователей



Результат логина через GitHub (смотреть на почту в верхнем правом углу)



Ссылка на GitHub

https://github.com/UnderAlex59/bird-clone-project?tab=readme-ov-file