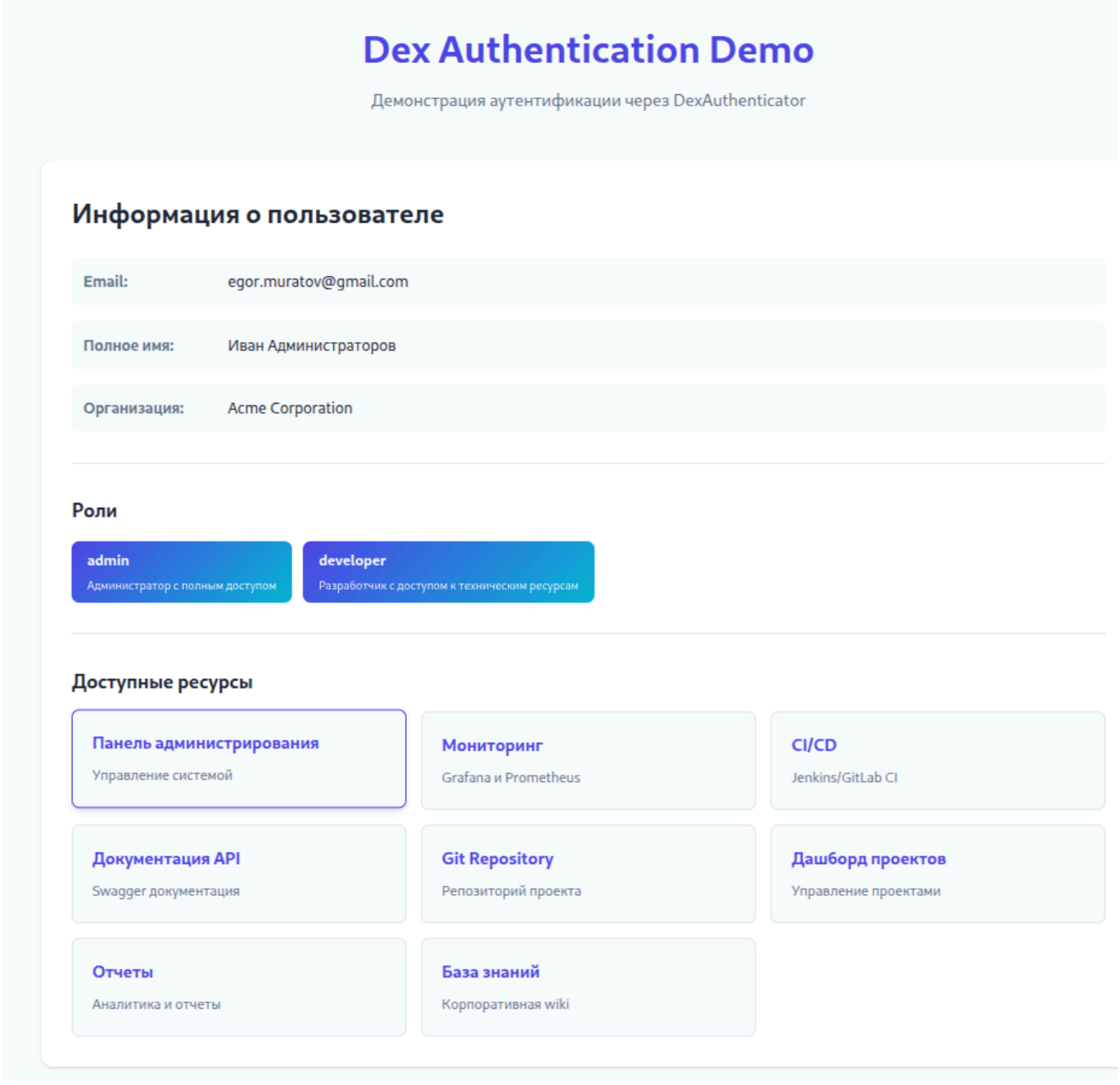
Dex Demo Application

Демонстрационное приложение для аутентификации через DexAuthenticator в Kubernetes кластере с Deckhouse.



Описание

Простое приложение, демонстрирующее интеграцию с DexAuthenticator:

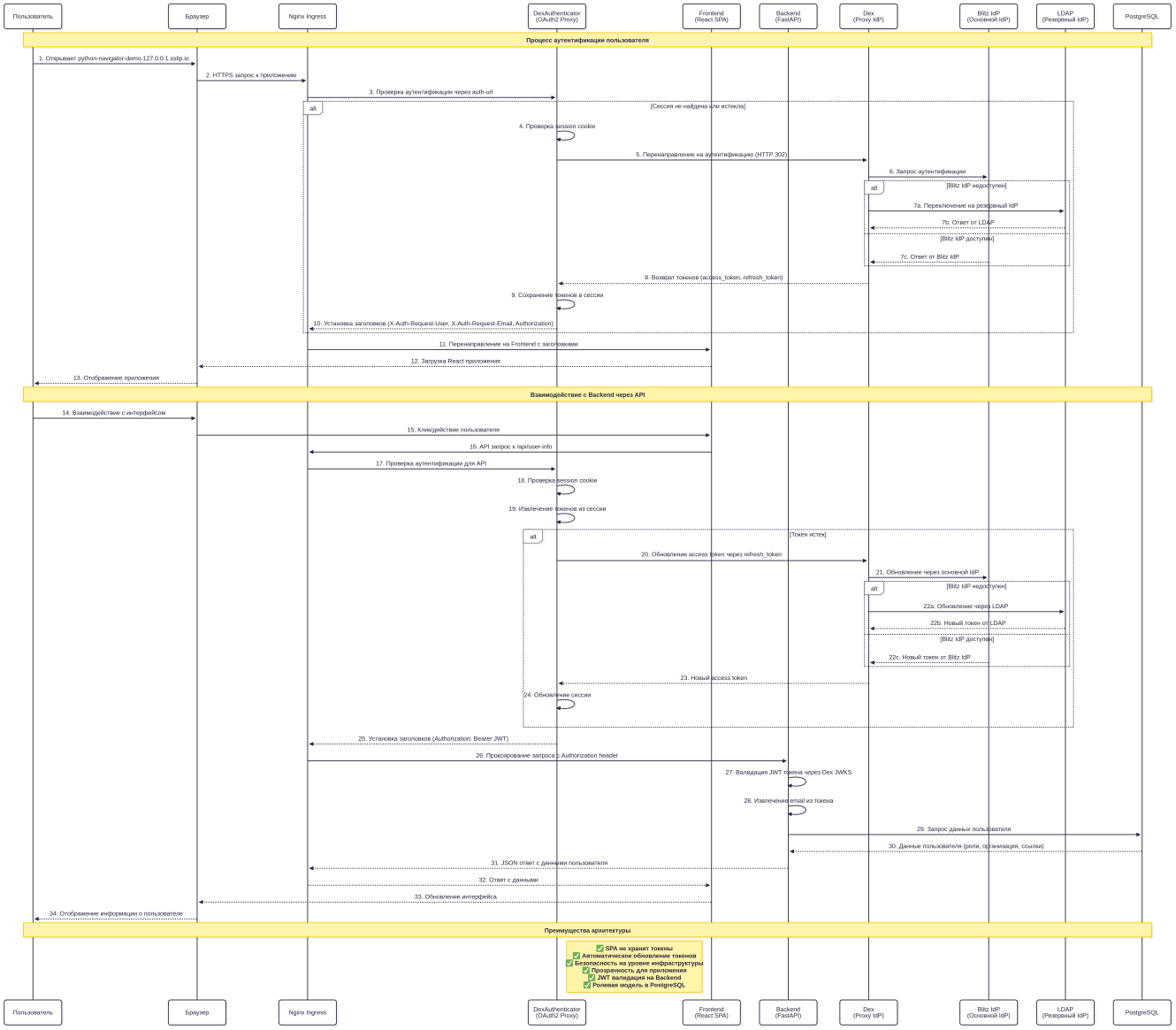
- **Backend (Python/FastAPI)**: Валидирует JWT токены, получает данные пользователя из PostgreSQL
- **Frontend (React/Vite)**: Отображает информацию о пользователе и доступные ресурсы на основе ролей
- **PostgreSQL**: Хранит пользователей, роли и доступные ссылки
- **DexAuthenticator**: Обеспечивает аутентификацию через Dex

Работа приложения

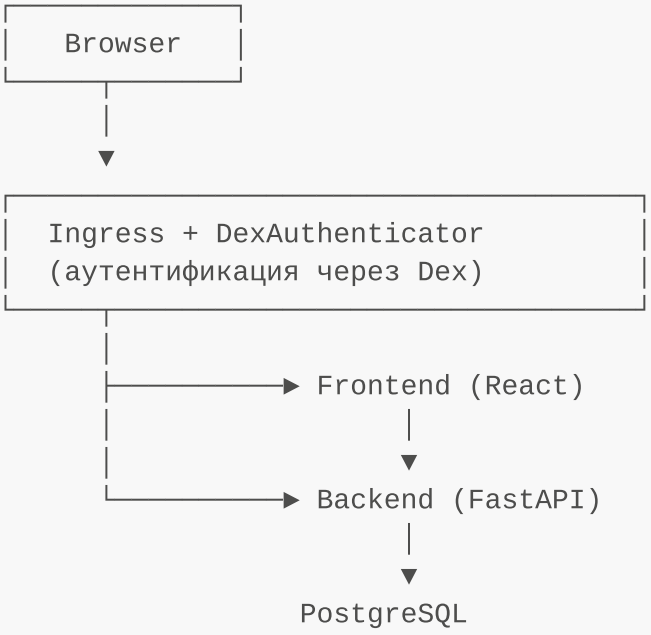
1. Пользователь открывает `https://python-navigator-demo.127.0.0.1.sslip.io`
2. Ingress перенаправляет на DexAuthenticator для аутентификации
3. Если не аутентифицирован происходит редирект на Dex (HTTP 302) для входа
4. Если не аутентифицирован в Dex происходит редирект на Blitz IdP (HTTP 302) для входа
5. После аутентификации в Blitz IdP → возврат в Dex
6. После успешной аутентификации Dex возвращает токен в DexAuthenticator
7. DexAuthenticator устанавливает заголовки (`X-Auth-Request-Email`,`X-Auth-Request-User`, `Authorization`) и cookie
8. После успешной аутентификации Frontend загружается
9. Frontend делает запрос к `/api/user-info`
10. DexAuthenticator устанавливает заголовки (`X-Auth-Request-Email`,`X-Auth-Request-User`, `Authorization`) и cookie
11. Backend:

- Валидирует JWT токен из заголовка `Authorization`
- Извлекает email/id пользователя
- Получает данные из PostgreSQL
- Возвращает информацию о пользователе и доступных ресурсах

12. Frontend отображает информацию о пользователе и доступные ресурсы



Архитектура



Предварительные требования

- Kubernetes кластер с Deckhouse
- Настроенный Dex по адресу `https://dex.127.0.0.1.sslip.io`
- Docker
- kubectl
- make

Быстрый старт

1. Сборка Docker образов

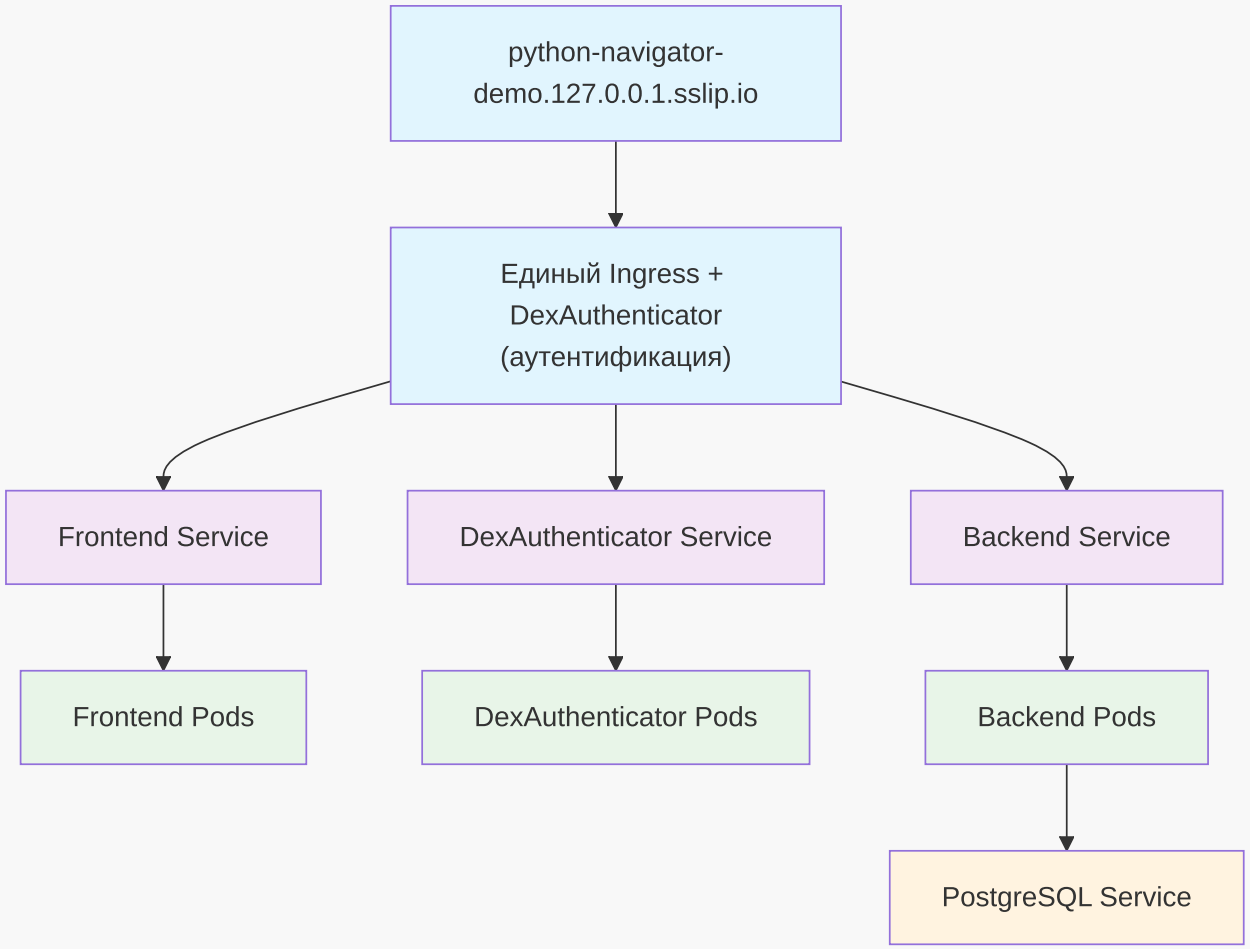
```
# Собрать все образы
make build-all

# Или по отдельности:
make build-backend
make build-frontend
```

2. Развертывание в Kubernetes

```
make deploy
```

Приложение будет доступно по адресу: `https://python-navigator-demo.127.0.0.1.sslip.io`

Финальная архитектура в Kubernetes:



3. Удаление приложения

```
make undeploy

# Или полная очистка (включая Docker образы):
make clean
```

Структура проекта

```
.
├── backend/              # Python бэкенд
│   ├── main.py           # FastAPI приложение
│   ├── requirements.txt # Python зависимости
│   └── Dockerfile        # Docker образ
├── frontend/             # React фронтенд
│   ├── src/
│   │   ├── App.jsx      # Главный компонент
│   │   └── App.css      # Стили
│   ├── nginx.conf       # Nginx конфигурация
│   └── Dockerfile       # Docker образ
├── db/                   # База данных
```

```
│   └── init.sql        # SQL скрипт инициализации
├── k8s/                # Kubernetes манифесты
│   ├── namespace.yaml
│   ├── postgres.yaml
│   ├── backend.yaml
│   ├── frontend.yaml
│   ├── dex-authenticator.yaml
│   └── ingress.yaml
├── Makefile            # Команды для сборки и развертывания
└── README.md           # Документация
```

Компоненты

Backend (FastAPI)

Эндпоинты:

- `GET /api/health` - Проверка здоровья сервиса
- `GET /api/user-info` - Получение информации о пользователе

Функциональность:

- Валидация JWT токенов от Dex (проверка подписи, issuer, exp)
- Извлечение email пользователя из токена или заголовков
- Получение данных пользователя из PostgreSQL (организация, полное имя)
- Получение ролей пользователя
- Получение доступных ссылок на основе ролей

Frontend (React + Vite)

Функциональность:

- Отображение информации о пользователе
- Список ролей
- Доступные ресурсы на основе ролей пользователя
- Никакой логики аутентификации (вся аутентификация на стороне DexAuthenticator)

База данных (PostgreSQL)

Схема:

- `organizations` - Организации
- `users` - Пользователи
- `roles` - Роли
- `user_roles` - Связь пользователей и ролей
- `links` - Доступные ссылки
- `role_links` - Связь ролей и ссылок

Тестовые данные

По умолчанию в БД загружаются следующие тестовые пользователи:

1. **admin@example.com** (Иван Администраторов)
    - Роли: admin, developer
    - Организация: Acme Corporation
    - Доступ: ко всем ресурсам
2. **developer@example.com** (Мария Разработчикова)
    - Роли: developer, user
    - Организация: Tech Innovators Inc
    - Доступ: технические ресурсы (CI/CD, Git, Docs, Wiki)
3. **user@example.com** (Петр Пользователев)
    - Роли: user
    - Организация: Global Solutions Ltd
    - Доступ: только база знаний
4. **manager@example.com** (Анна Менеджерова)
    - Роли: manager, user
    - Организация: Acme Corporation
    - Доступ: управленческие ресурсы (Проекты, Отчеты, Wiki)

Важно: Убедитесь, что эти email совпадают с пользователями в вашем Dex, или измените данные в `db/init.sql`

Конфигурация

Backend переменные окружения

- `DB_HOST` - Хост PostgreSQL (по умолчанию: `postgres`)
- `DB_PORT` - Порт PostgreSQL (по умолчанию: `5432`)
- `DB_NAME` - Имя БД (по умолчанию: `dexdemo`)
- `DB_USER` - Пользователь БД (по умолчанию: `dexdemo`)
- `DB_PASSWORD` - Пароль БД
- `DEX_ISSUER` - URL Dex issuer (по умолчанию: `https://dex.127.0.0.1.sslip.io/`)

DexAuthenticator

Настройки в `k8s/dex-authenticator.yaml`:

- `applicationDomain` - Домен приложения
- `sendAuthorizationHeader` - Отправка заголовка Authorization с JWT
- `keepUsersLoggedInFor` - Время сессии (24h)

Разработка

Локальная разработка backend

```
cd backend
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
export DB_HOST=localhost
export DEX_ISSUER=https://dex.127.0.0.1.sslip.io/

python main.py
```

Для удобной локальной разработки без настройки OIDC/Dex есть режим разработки.

```
export INSECURE_DEV_MODE=true
export INSECURE_DEV_EMAIL=developer@example.com
```

Что происходит в режиме разработки

- **Отключается проверка JWT токенов** - не требуется настройка Dex или OIDC
- **Используется фиксированный email** - задается через переменную `INSECURE_DEV_EMAIL`
- **Логирование** - в консоли будет выводиться сообщение о том, какой email используется

Никогда не используйте INSECURE_DEV_MODE=true в продакшене! Это отключает всю аутентификацию.

Локальная разработка frontend

```
cd frontend
npm install
npm run dev
```

Frontend будет доступен на `http://localhost:5173` с проксированием API на `http://localhost:8000`

Ошибка "User not found in database"

Убедитесь, что email пользователя из Dex совпадает с email в таблице `users` в PostgreSQL.

JWT валидация не работает

Проверьте:

1. Доступность Dex JWKS endpoint: `https://dex.127.0.0.1.sslip.io/keys`
2. Переменную окружения `DEX_ISSUER` в backend
3. Логи backend для деталей ошибки

Дополнительная настройка

Изменение тестовых пользователей

Отредактируйте `db/init.sql` или `k8s/postgres.yaml` (ConfigMap `postgres-init`), затем:

```
kubectl delete pod -n navigator-demo -l app=postgres
```

Использование собственного домена

Измените `applicationDomain` в `k8s/dex-authenticator.yaml` и `host` в `k8s/ingress.yaml`

Production deployment

Для production окружения:

1. Используйте secrets для паролей БД
2. Включите TLS сертификаты
3. Настройте resource limits
4. Добавьте HorizontalPodAutoscaler
5. Используйте внешний PostgreSQL