Инструкция по деплою и правила разработкиа

Содержание

Общие принципы

1. Все конфигурации должны управляться через переменные окружения (`.env` файлы)
2. Переопределение конфигурации доступно через `config.override.json` (только в режиме разработки)
3. Запрещено использование сторонних CDN для любых ресурсов (включая шрифты, скрипты, стили)
4. Весь код должен следовать принятым соглашениям о стиле кодирования

Настройка окружения

Переменные окружения

Все конфигурации приложения должны быть вынесены в `.env` файлы:

- `.env` - базовые настройки, общие для всех окружений
- `.env.development` - настройки для разработки
- `.env.production` - настройки для продакшена

Пример структуры `.env` файла:

```
VITE_API_URL=https://api-id.AOHK.ru/
VITE_APP_NAME=AOHK ID
```

Переопределение конфигурации

В процессе разработки можно переопределить настройки без изменения исходного кода, используя файл `config.override.json`. Этот файл должен иметь следующую структуру:

```
{
  "config": {
    "apiUrl": "http://localhost:8000/",
    "appName": "AOHK ID (локальная разработка)"
  }
}
```

Важно: Механизм переопределения работает только в режиме разработки. В продакшн-среде он отключен для безопасности.

Персистентные папки

Персистентные папки - это директории, которые сохраняются между обновлениями приложения и не очищаются при повторном развертывании.

Персистентные папки в проекте создаются и обозначаются в Dockerfile с помощью директивы `VOLUME`. Пример из бэкенд-проекта:

```
# Создание персистентной директории uploads
RUN mkdir -p /var/www/aonk-id/runtime/uploads
VOLUME /var/www/aonk-id/runtime/uploads

# Создание персистентной директории drivers
RUN mkdir -p /var/www/aonk-id/runtime/drivers
VOLUME /var/www/aonk-id/runtime/drivers
```

Чтобы создать новую персистентную папку:

1. Создайте директорию в корне проекта
2. Добавьте путь к этой директории в файл `.gitignore` (чтобы избежать коммита содержимого)
3. Добавьте следующие строки в `contrib/Dockerfile`:

   ```
   # Создание персистентной директории
   RUN mkdir -p /var/www/aonk-id/my-persistent-dir
   VOLUME /var/www/aonk-id/my-persistent-dir
   ```

   

Директива `VOLUME` в Dockerfile гарантирует, что содержимое указанной директории будет сохраняться между перезапусками контейнера и обновлениями приложения.

Важно: Данные в томах (`VOLUME`) сохраняются даже при удалении контейнера, но могут быть удалены при выполнении команды `docker volume prune`. Убедитесь, что у вас настроено резервное копирование важных данных.

Инструкции для фронтенда

Структура проекта (Frontend)

```
frontend/
├── dist/              # Результат сборки (очищается при каждой сборке)
├── public/            # Статические файлы
│   └── config.override.json  # Файл переопределения конфигурации
├── src/               # Исходный код приложения
│   ├── config/        # Конфигурация приложения
│   │   ├── env.config.js      # Базовая конфигурация
│   │   ├── configRewrite.js   # Механизм переопределения
│   │   └── index.js           # Экспорт конфигурации
│   ├── components/    # Компоненты Vue
│   ├── plugins/       # Плагины Vue
│   ├── router/        # Маршрутизация
│   ├── stores/        # Хранилища Pinia
│   └── styles/        # Стили приложения
├── node_modules/      # Зависимости (очищаются при сборке)
├── contrib/           # Файлы для развертывания
│   ├── Dockerfile     # Dockerfile для сборки образа
│   └── aonk-id-frontend.nginx.conf  # Конфигурация Nginx
├── debian/            # Файлы для сборки Debian-пакета
│   ├── control        # Метаданные пакета
│   ├── aonk-id-frontend-bin.install  # Список файлов для установки
│   └── aonk-id-frontend-bin.dir      # Список создаваемых директорий
├── Makefile           # Скрипты сборки
└── build.sh           # Скрипт создания Debian-пакета
```

Добавление файлов и папок в сборку

При добавлении новых файлов и папок в проект, необходимо учитывать два аспекта:

1. **Включение файлов в сборку Vite**:
   - Все исходные файлы должны размещаться в директории `src/`
   - Статические файлы (изображения, шрифты и т.д.) должны размещаться в директории `public/`
   - Vite автоматически включит эти файлы в результирующую сборку (директория `dist/`)
2. **Включение файлов в установочный пакет**:
   - Для определения файлов, которые должны быть установлены на сервер, используется файл `debian/aonk-id-frontend-bin.install`
   - Каждая строка в этом файле содержит два элемента: исходный путь и путь назначения

   Пример содержимого файла `aonk-id-frontend-bin.install`:

   ```
   package.json /var/www/aonk-id
   package-lock.json /var/www/aonk-id
   dist/* /var/www/aonk-id/public
   ```

Если вам необходимо добавить новые файлы или директории в установочный пакет:

1. Добавьте новую строку в файл `debian/aonk-id-frontend-bin.install`, указав исходный путь и путь назначения

   ```
   my-config-file.json /var/www/aonk-id/configs
   ```

2. Убедитесь, что указанный каталог назначения существует или создайте его, добавив путь в файл `debian/aonk-id-frontend-bin.dir`:

   ```
   /var/www/aonk-id/configs
   ```

Управление состоянием

Для управления состоянием приложения используется Pinia. При добавлении нового хранилища:

1. Создайте файл в директории `src/stores/`
2. Используйте `defineStore` для определения хранилища
3. Импортируйте конфигурацию из `@/config`, если требуется доступ к настройкам

Пример:

```
import { defineStore } from 'pinia'
import config from '@/config'

export const useAuthStore = defineStore('auth', {
  state: () => ({
    user: null,
    apiUrl: config.apiUrl
  }),
  actions: {
    // ...
  }
})
```

Сборка и деплой фронтенда

1. Установка зависимостей:

   ```
   yarn install
   ```

2. Запуск в режиме разработки:

   ```
   yarn dev
   ```

3. Сборка для продакшена:

```
yarn build
```

4. Создание Debian-пакета:

```
./build.sh
```

Инструкции для бэкенда

Структура проекта (Backend)

```
backend/
├── config/            # Конфигурация приложения
│   ├── env.config.js   # Базовая конфигурация
│   └── configRewrite.js # Механизм переопределения
├── controllers/       # Контроллеры
├── models/            # Модели данных
├── routes/            # Маршруты API
├── middleware/        # Промежуточное ПО
├── helpers/           # Вспомогательные функции
├── drivers/           # Драйверы (персистентная папка)
├── uploads/           # Загруженные файлы (персистентная папка)
├── contrib/           # Файлы для развертывания
│   └── Dockerfile     # Dockerfile для сборки образа
├── debian/            # Файлы для сборки Debian-пакета
│   ├── control        # Метаданные пакета
│   └── aonk-id-backend-bin.install  # Список файлов для установки
├── Makefile           # Скрипты сборки
└── build.sh           # Скрипт создания Debian-пакета
```

Добавление файлов и папок в сборку (Backend)

Для включения файлов в установочный пакет бэкенда, используется файл `debian/aonk-id-backend-bin.install`. Каждая строка в файле определяет исходный путь и путь назначения:

```
package.json /var/www/aonk-id/runtime
package-lock.json /var/www/aonk-id/runtime
config /var/www/aonk-id/runtime
controllers /var/www/aonk-id/runtime
emails /var/www/aonk-id/runtime
helpers /var/www/aonk-id/runtime
middleware /var/www/aonk-id/runtime
models /var/www/aonk-id/runtime
readme /var/www/aonk-id/runtime
node_modules /var/www/aonk-id/runtime
routes /var/www/aonk-id/runtime
seeders /var/www/aonk-id/runtime
index.js /var/www/aonk-id/runtime
config.override.json /var/www/aonk-id/runtime
contrib/aonk-id.service /lib/systemd/system
contrib/aonk-id /etc/default
```

Работа с базой данных

Для работы с базой данных в проекте используется Sequelize ORM:

1. Модели данных размещаются в директории `models/`
2. Миграции размещаются в директории `migrations/`
3. Сидеры для заполнения тестовыми данными - в директории `seeders/`

Пример создания миграции:

```
npx sequelize-cli migration:generate --name create-users-table
```

Запуск миграций:

```
npx sequelize-cli db:migrate
```

Персистентные папки (Backend)

На бэкенде определены следующие персистентные директории:

- `/var/www/aonk-id/runtime/uploads` - для загруженных файлов
- `/var/www/aonk-id/runtime/drivers` - для драйверов

Сборка и деплой бэкенда

1. Установка зависимостей:
   ```
   npm install
   ```
2. Запуск в режиме разработки:
   ```
   npm run dev
   ```
3. Создание Debian-пакета:
   ```
   ./build.sh
   ```

Тестовые стенды (Staging)

Создание и обновление стендов

Для работы с тестовыми стендами используется механизм веток. Каждая ветка формата `stand/название` автоматически создает или обновляет соответствующий тестовый стенд:

1. Создайте новую ветку с префиксом `stand/`:
   ```
   git checkout -b stand/feature-name
   ```
2. Внесите необходимые изменения и закоммитьте их:
   ```
   git add .
   git commit -m "Описание изменений"
   ```

3. Отправьте ветку в удаленный репозиторий:

```
git push origin stand/feature-name
```

После пуша изменений в ветку `stand/feature-name`, система CI/CD автоматически развернет или обновит тестовый стенд с соответствующим именем.

Чистая установка стенда

Для полной очистки тестового стенда (включая базу данных и персистентные файлы) необходимо выполнить коммит с ключевым словом `CleanStand` в сообщении:

```
git commit --allow-empty -m "CleanStand: полная очистка тестового стенда"
git push origin stand/feature-name
```

Важно: Используйте эту операцию с осторожностью, так как она приведет к потере всех данных на тестовом стенде.

Взаимодействие фронтенда и бэкенда

API-интерфейсы

Взаимодействие между фронтендом и бэкендом осуществляется через REST API.

1. На бэкенде API-маршруты определяются в директории `routes/`
2. На фронтенде для работы с API используется axios

Аутентификация и авторизация

Для авторизации используется механизм JWT-токенов:

1. Фронтенд отправляет учетные данные пользователя на бэкенд
2. Бэкенд проверяет учетные данные и выдает JWT-токен
3. Фронтенд сохраняет токен и использует его для последующих запросов

Проверка перед деплоем

Перед деплоем в продакшн-среду необходимо выполнить следующие проверки:

1. Убедиться, что все тесты проходят успешно
2. Проверить сборку на наличие предупреждений или ошибок
3. Проверить работоспособность основных функций приложения
4. Удостовериться, что все переменные окружения для продакшн-среды корректно настроены

Правила разработки

Обязательные требования

1. **Конфигурации через переменные окружения**
   - Все настройки, зависящие от окружения, должны быть определены через `.env` файлы
   - Доступ к переменным должен осуществляться через объект конфигурации, а не напрямую через `import.meta.env` или `process.env`
2. **Механизм переопределения конфигурации**
   - Каждый проект должен поддерживать переопределение конфигурации через `config.override.json`
   - Логика загрузки переопределений должна работать только в режиме разработки

3. **Запрет на использование сторонних CDN**
   - Все ресурсы (шрифты, скрипты, стили) должны быть локальными
   - Шрифты должны размещаться в директории проекта, а не загружаться через Google Fonts или другие CDN
   - Все библиотеки должны быть установлены через npm/yarn
4. **Безопасность**
   - Не хранить секреты и ключи в исходном коде
   - Не включать учетные данные в репозиторий
   - Использовать HTTPS для всех внешних запросов

Лучшие практики

1. **Код**
   - Следовать принятому стилю кодирования
   - Писать тесты для критически важного функционала
   - Использовать типизацию где это возможно
2. **Производительность**
   - Минимизировать количество и размер внешних зависимостей
   - Оптимизировать изображения и другие медиа-ресурсы
   - Использовать ленивую загрузку для тяжелых компонентов
3. **Доступность**
   - Обеспечивать доступность интерфейса (WCAG)
   - Поддерживать корректное отображение на различных устройствах
   - Учитывать особенности разных браузеров

Версия #2
★Гудин Сергей Витальевич создал 13 мая 2025 08:40:14
Гудин Сергей Витальевич обновил 13 мая 2025 08:41:04