Оглавление

[Описание кода 2](#_Toc200983300)

[Как создать страницу 12](#_Toc200983301)

# Описание кода

**/server**

/index.js:

Это **REST API бэкенд** на Node.js + Express, который:

1. Принимает HTTP-запросы (GET/POST/PUT/DELETE)
2. Работает с базой данных через Sequelize (PostgreSQL/MySQL/SQLite)
3. Отдаёт JSON-ответы или файлы
4. Обрабатывает ошибки централизованно

/db.js:

Этот файл (db.js или аналогичный) отвечает за **настройку и экспорт подключения к базе данных PostgreSQL** с использованием ORM Sequelize.

**Как это работает?**

1. При импорте этого файла создается экземпляр Sequelize.
2. Sequelize автоматически устанавливает соединение с PostgreSQL, используя переданные параметры.
3. Если подключение успешно — можно работать с моделями. Если нет — выбрасывает ошибку (например, ConnectionRefusedError).

**/routes(нужен для роутинга, т.е. использует get и post, где запросы вызываемые функции):**  
userRouter.js:

Этот файл определяет **API-эндпоинты для работы с пользователями** (регистрация, авторизация, проверка авторизации) с использованием Express Router.

**Как это работает?**

1. **Запрос приходит** на один из эндпоинтов (/registration, /login, /auth).
2. **Роутер перенаправляет** его в соответствующий метод userController.
3. Для /auth сначала выполняется authMiddleware (проверка токена).
4. **Контроллер** обрабатывает запрос и возвращает ответ (успех/ошибка).

instrumentRouter.js:

Этот файл определяет **CRUD-эндпоинты для работы с инструментами** (создание, получение списка и одного инструмента) с использованием Express Router.

**Как это работает?**

1. **Запрос приходит** на один из эндпоинтов (/, /:id).
2. **Роутер перенаправляет** его в соответствующий метод instrumentController.
3. **Контроллер** взаимодействует с БД (через Sequelize) и возвращает результат.

index.js(объединяет все роуты в один файл, скажем так использует роуты):

Этот файл является главным роутером приложения, который объединяет все отдельные роутеры (инструментов, пользователей и брендов) в один основной роутер.

1. **Создание главного роутера**: Сначала создается экземпляр Express Router, который будет обрабатывать все входящие запросы.
2. **Импорт специализированных роутеров**: Каждый роутер отвечает за свою предметную область:
   * instrumentRouter - операции с инструментами (CRUD)
   * userRouter - операции с пользователями
   * brandRouter - операции с брендами
3. **Маршрутизация запросов**:
   * Запросы к /user/\* направляются в userRouter
   * Запросы к /brand/\* направляются в brandRouter
   * Запросы к /instrument/\* направляются в instrumentRouter
4. **Масштабируемость**: Такой подход позволяет:
   * Легко добавлять новые роутеры для новых сущностей
   * Поддерживать код в чистоте, разделяя логику по разным файлам
   * Упрощать тестирование и отладку

categoryRouter.js:

Этот файл представляет собой роутер для работы с категориями (categoryRouter.js). Он определяет конечные точки (endpoints) API для создания категорий и получения списка всех категорий.

**Такой подход обеспечивает четкое разделение ответственности:**

* Роутер только маршрутизирует запросы
* Логика обработки находится в контроллере
* Проверка прав доступа вынесена в middleware

brandRouter.js:

Этот файл представляет собой роутер для работы с брендами (brandRouter.js). Он определяет два основных эндпоинта API: создание бренда и получение списка всех брендов.

**/models(модели):**  
models.js(создаёт БД: таблицы, зависимости, поля и т.д.):

Этот файл определяет все модели базы данных и их взаимосвязи для музыкального магазина.

**Эта структура обеспечивает:**

* Четкое определение всех сущностей системы
* Правильные связи между моделями
* Возможность расширения функционала
* Поддержку всех необходимых операций CRUD
* Гибкость в запросах благодаря Sequelize

**/middleware(в проекте нужны для проверки роли, авторизации и вывода ошибок):**

ErrorHandlingMiddleware.js:

Этот файл представляет собой middleware для обработки ошибок в Express-приложении. Он перехватывает все ошибки, возникающие в процессе обработки запросов, и отправляет соответствующий HTTP-ответ клиенту.

Этот middleware является важной частью инфраструктуры обработки ошибок в Express-приложении, обеспечивая согласованный формат ответов и централизованную обработку исключений.

checkRoleMiddleware.js:

Этот файл представляет собой middleware для проверки JWT-токена и роли пользователя. Он используется для ограничения доступа к определенным маршрутам на основе роли пользователя.

Этот middleware обеспечивает безопасность API, проверяя права доступа пользователей перед выполнением защищенных операций. Он гибко настраивается под разные роли и может быть легко расширен для дополнительных проверок.

**Cross-Origin Resource Sharing (CORS)** — это механизм безопасности в веб-разработке, который позволяет или запрещает веб-браузерам делать запросы на серверы, находящиеся на другом домене.

**OPTIONS-запросы** — это HTTP-запросы, которые позволяют клиентам запрашивать информацию о доступных вариантах коммуникации для конкретного ресурса на сервере.

authMiddleware.js:

Этот файл представляет собой middleware для проверки JWT-токена и аутентификации пользователя. В отличие от предыдущего варианта с проверкой ролей, этот middleware только проверяет валидность токена без проверки конкретной роли.

**/error:**

AplError.js:

Этот файл реализует кастомный класс ошибок ApiError, который расширяет базовый класс Error и добавляет специфичные для API функции.

**Преимущества такого подхода**

1. **Стандартизация**:
   * Все ошибки API имеют одинаковую структуру
   * Упрощается обработка ошибок на клиентской стороне
2. **Удобство**:
   * Фабричные методы позволяют быстро создавать типовые ошибки
   * Не нужно запоминать коды статусов
3. **Расширяемость**:
   * Легко добавить новые методы для других статусов
   * Можно расширить функционал (например, добавить код ошибки)

**/controllers:**

userController.js:

Этот файл реализует логику работы с пользователями: регистрацию, авторизацию и проверку токена.

**Особенности реализации**

1. **Безопасность**:
   * Пароли хранятся в хешированном виде (bcrypt)
   * Используется JWT с ограниченным сроком действия
   * Секретный ключ берется из переменных окружения
2. **Работа с базой данных**:
   * Используется Sequelize ORM
   * При регистрации автоматически создается корзина пользователя
3. **Обработка ошибок**:
   * Используется кастомный класс ApiError
   * Единый формат ответов об ошибках

typeController.js:

Этот контроллер отвечает за операции с типами товаров в вашем интернет-магазине.

instrumentController.js:

Этот контроллер реализует CRUD-операции для музыкальных инструментов, включая загрузку изображений и работу с дополнительной информацией.

Этот контроллер обеспечивает полный функционал для работы с музыкальными инструментами в вашем интернет-магазине, включая загрузку изображений, фильтрацию и пагинацию.

categoryController.js:

Этот контроллер отвечает за операции с категориями товаров в вашем интернет-магазине.

brandController.js:

Этот контроллер отвечает за операции с брендами товаров в вашем интернет-магазине.

**/client**

routes.js:

Этот файл определяет маршруты для вашего интернет-магазина музыкальных инструментов, разделяя их на защищенные (authRoutes) и публичные (publicRoutes).

**Этот подход к маршрутизации обеспечивает:**

* Четкое разделение защищенных и публичных маршрутов
* Легкую масштабируемость при добавлении новых страниц
* Гибкость в настройке прав доступа
* Поддержку динамических маршрутов (для страниц инструментов)

index.js:

Этот файл является точкой входа в ваше React-приложение и устанавливает глобальный контекст для управления состоянием.

Этот подход к инициализации приложения обеспечивает:

* Централизованное управление состоянием
* Легкий доступ к данным из любого компонента
* Хорошую масштабируемость при добавлении новых хранилищ

App.js:

Этот компонент является корневым для вашего React-приложения и реализует ключевую логику инициализации.

**Этот компонент App обеспечивает:**

* Проверку авторизации при запуске
* Глобальную маршрутизацию
* Реактивное обновление состояния
* Гибкую структуру для расширения функционала

**/utils:**

consts.js:

Записаны константы, которые показывают пути к страницам.

export const *ADMIN\_ROUTE* = '/admin';  
export const *LOGIN\_ROUTE* = '/login';  
…

**/store(файлы хранилищ для управления состояниями):**UserStore.js:

Этот класс представляет собой MobX-хранилище для управления состоянием аутентификации и данными пользователя.

InstrumentStore.js:

Этот класс представляет собой MobX-хранилище для управления состоянием каталога музыкальных инструментов, включая типы, категории, бренды и сами инструменты.

**/pages(файлы с вёрсткой страниц):**

Shop.js:

Это главный компонент страницы магазина, который объединяет все элементы интерфейса для просмотра и фильтрации музыкальных инструментов.

**Этот компонент обеспечивает:**

* + Четкую структуру страницы магазина
  + Эффективную загрузку и отображение данных
  + Гибкую систему фильтрации и сортировки

InstrumentPage.js:

Страница инструмента

Basket.js:

Страница корзины

Auth.js:

Страница авторизации

Admin.js:

Админ панель

Компонент реализует административную панель с использованием React-хуков для управления состоянием модальных окон. Применен декларативный подход для рендеринга интерфейса, где каждое действие (добавление типа, бренда или инструмента) обрабатывается через отдельное модальное окно. Использованы Bootstrap-компоненты для единообразия стилей.

**/http:**

userAPI.js:

Модуль реализует базовые функции для работы с пользователями: регистрацию, авторизацию и проверку токена. Используется JWT-аутентификация с сохранением токена в localStorage. Для работы с защищенными роутами применяется $authHost, который автоматически добавляет токен к запросам. Все функции возвращают декодированные данные пользователя из токена.

instrumentAPI.js:

Модуль предоставляет API для работы с музыкальными инструментами и связанными сущностями (типы, категории, бренды). Использует два типа хостов: $authHost для защищенных операций (требующих авторизации) и $host для публичных запросов. Поддерживает пагинацию и фильтрацию при получении списка инструментов.

**Ключевые особенности:**

* Разделение на публичные и защищенные endpoint'ы
* Поддержка фильтрации инструментов по параметрам
* Полноценный CRUD для всех сущностей
* Стандартизированный интерфейс ответов

Для работы с JWT токенами используется библиотека jwt-decode, хотя в текущей реализации она не задействована напрямую в этих методах.

index.js:

Модуль настраивает два экземпляра axios-клиента с базовыми параметрами. Основные особенности:

1. **Базовый клиент ($host)**:
   * Для публичных запросов без авторизации
   * Использует базовый URL из переменных окружения
2. **Авторизованный клиент ($authHost)**:
   * Автоматически добавляет JWT-токен в заголовки
   * Включает логирование запросов в development-режиме
   * Перехватчики для унифицированной обработки запросов
3. **Безопасность**:
   * Токен берется из localStorage
   * Стандартный формат заголовка Authorization
   * Защита от отсутствия токена

**/components(все остальные страницы и компоненты сайта):**

TypeBar.js:

Компонент реализует панель выбора типов музыкальных инструментов с визуальным выделением активного элемента. Использует MobX для управления состоянием и React Bootstrap для стилизации. При выборе типа обновляет состояние в глобальном хранилище инструментов.

NavBar.js:

Компонент навигационной панели реализует:

* **Адаптивный дизайн** (использует Navbar.Collapse для мобильных устройств)
* **Условный рендеринг** (разные кнопки для авторизованных/неавторизованных пользователей)
* **Управление сессией** (очистка токена при выходе)
* **Доступ к глобальному состоянию** через MobX
* **Навигацию** между основными разделами приложения

Особенности реализации:

* Использование стандартных компонентов React Bootstrap
* Оптимизированная структура JSX
* Централизованное управление путями через константы
* Поддержка адаптивного дизайна
* Интеграция с системой аутентификации

InstrumentList.js:

**Детали реализации:**

1. **Глобальное состояние**:
   * Компонент получает доступ к хранилищу инструментов через MobX контекст
   * Использует observer для автоматического обновления при изменении данных
2. **Рендеринг списка**:
   * Для каждого инструмента создается компонент InstrumentItem
   * Ключевой атрибут key обязателен для оптимизации работы React
3. **Структура компонента**:
   * Row - контейнер сетки из Bootstrap
   * instrument.instruments - массив инструментов из глобального состояния
   * InstrumentItem - презентационный компонент элемента списка
4. **Особенности работы**:
   * Автоматически реагирует на изменения в списке инструментов
   * Оптимизирован для работы с большими списками
   * Не содержит бизнес-логики, только отображение

InstrumentItem.js:

**Ключевые особенности компонента:**

1. **Навигация**:
   * При клике перенаправляет на страницу детальной информации
   * Использует динамический роут с ID инструмента
2. **Отображение данных**:
   * Показывает изображение, название, бренд и рейтинг
   * Оптимизированное отображение текста (обрезание длинных названий)
3. **Стилизация**:
   * Адаптивная ширина (3 колонки на строку на средних экранах)
   * Эффекты hover (указатель при наведении)
4. **Доступность**:
   * Добавлены alt-тексты для изображений
   * Четкая визуальная иерархия элементов
5. **Оптимизации**:
   * Использует кэширование изображений
   * Компактный размер компонента

BrandBar.js:

Компонент BrandBar отображает горизонтальную панель брендов музыкальных инструментов, позволяя пользователям фильтровать товары по выбранному бренду. Активный бренд выделяется синим фоном с белым текстом, обеспечивая четкую визуальную обратную связь. Реализована плавная анимация переключения между состояниями для улучшения пользовательского опыта.

Компонент BrandBar представляет собой интерактивную панель для выбора брендов музыкальных инструментов. При нажатии на бренд он выделяется синим цветом, а состояние сохраняется в глобальном хранилище, позволяя фильтровать инструменты по выбранному производителю.

AppRouter.js:

Компонент AppRouter управляет навигацией в приложении, разделяя маршруты на защищенные (для авторизованных пользователей) и публичные. При попытке доступа к несуществующему пути происходит автоматическое перенаправление на главную страницу магазина.

**/components/modals(модальные окна для создания элементов):**

CreateType.js:

Компонент CreateType представляет собой модальное окно для добавления новых типов музыкальных инструментов. При вводе названия типа и нажатии кнопки "Добавить" отправляется запрос к API, после чего окно автоматически закрывается. Реализована базовая валидация и обработка действий пользователя.

CreateInstrument.js:

Компонент CreateInstrument представляет собой сложную форму для добавления новых музыкальных инструментов в каталог. Он включает выпадающие списки для выбора категории, типа и бренда, поля для ввода названия и цены, загрузку изображения и динамическое добавление характеристик. Все данные отправляются на сервер в формате FormData, что позволяет эффективно передавать как текстовую информацию, так и файлы.

CreateBrand.js:

Компонент CreateBrand предоставляет простой интерфейс для добавления новых брендов музыкальных инструментов. При вводе названия и нажатии кнопки отправляется запрос к серверу, после успешного создания бренда форма автоматически очищается и закрывается. Реализована базовая валидация и обработка ошибок.

# Как создать страницу

# Загрузка всех библиотек и фреймворков

**Серверная часть:**

PS D:\diplom\site\_test\_1> cd server

PS D:\diplom\site\_test\_1\server> npm init -y

PS D:\diplom\site\_test\_1\server> npm install express

PS D:\diplom\site\_test\_1\server> npm install pg pg-hstore sequelize cors dotenv

PS D:\diplom\site\_test\_1\server> npm -D nodemon

PS D:\diplom\site\_test\_1\server> npm i express-fileupload

PS D:\diplom\site\_test\_1\server> npm i bcrypt

PS D:\diplom\site\_test\_1> npm i jsonwebtoken

Далее изменяем файл package.json:

scripts": {  
 "test": "echo \"Error: no test specified\" && exit 1"  
},

Меняем на:

scripts": {  
 "dev": "nodemon index.js"  
},

**Клиентская часть:**

PS D:\diplom\site\_test\_1> cd client

PS D:\diplom\site\_test\_1\client> npx create-react-app .

PS D:\diplom\site\_test\_1\client> npm i axios react-router-dom mobx mobx-react-lite

PS D:\diplom\site\_test\_1\client> npm i [bootstrap@5.3.6](mailto:bootstrap@5.3.6)

PS D:\diplom\site\_test\_1\client> npm install react-toastify