Министерство образования и науки Республики Марий Эл

ГБПОУ РЕСПУБЛИКИ МАРИЙ ЭЛ

«ЙОШКАР-ОЛИНСКИЙ ТЕХНОЛОГИЧЕСКИЙ КОЛЛЕДЖ»

|  |  |
| --- | --- |
| Специальность: | 09.02.07 "Информационные системы и программирование" |
| Квалификация: | Специалист по информационным системам |
| Группа: | И-31 |

**РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ**

**«ОНЛАЙН МАГАЗИН ВИНИЛОВЫХ ПЛАСТИНОК»**

Пояснительная записка к курсовому проекту  
ЙОТК.09.02.07. И-1538 ПЗ-КП

Руководитель

\_\_\_\_\_\_\_\_\_\_ Е.С. Чернова

Разработал

\_\_\_\_\_\_\_\_\_\_ Е.А. Яндулов

Оценка по защите курсового проекта\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Чернова Е.С. /

Йошкар-Ола 2024

# Содержание

[Содержание 2](#_Toc165891612)

[1. ВВЕДЕНИЕ 3](#_Toc165891613)

[1.1 Актуальность 3](#_Toc165891614)

[1.2 Объект исследования 4](#_Toc165891615)

[1.3 Предмет исследования 4](#_Toc165891616)

[1.4 Цели исследования 4](#_Toc165891617)

[1.5 Задачи исследования 4](#_Toc165891618)

[2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 5](#_Toc165891619)

[2.1 Фронтенд и бэкенд 5](#_Toc165891621)

[2.2 Реляционная база данных 6](#_Toc165891622)

[2.3 API сервер 8](#_Toc165891623)

[2.4 Фреймворк 10](#_Toc165891624)

[2.5 Теоретические аспекты построения web-сайта 17](#_Toc165891625)

[3. ОСНОВНАЯ ЧАСТЬ 19](#_Toc165891626)

[3.1 Требования к системе 19](#_Toc165891628)

[3.2 Тестовый сценарий 19](#_Toc165891629)

[3.3 Этапы разработки ИС 20](#_Toc165891630)

[4. ЗАКЛЮЧЕНИЕ 42](#_Toc165891631)

[5. ЛИТЕРАТУРА И ПРОЧИЕ ИСТОЧНИКИ 43](#_Toc165891632)

[Приложение А 44](#_Toc165891633)

[Приложение Б 46](#_Toc165891634)

# ВВЕДЕНИЕ

В современном мире онлайн-магазины являются неотъемлемой частью нашей повседневной жизни. В них есть широкий ассортимент различной продукции: начиная от еды, заканчивая автомобилями и недвижимостью. Есть универсальные площадки, в которых продаются и еда, и одежда, и прочее. Но также есть и узконаправленные площадки: доставка еды, магазины одежды и так далее.

В своей курсовой работе я решил разработать простой онлайн-магазин виниловых пластинок. До сих пор виниловые пластинки сохраняют свою актуальность, многие музыкальные артисты выпускают свои альбомы и другие материалы на виниле. Виниловые пластинки больше ценятся как коллекционная вещь, так как в современном мире существует множество музыкальных стриминговых сервисов.

## Актуальность

Веб-приложение предназначено для поиска, выбора и покупки виниловых пластинок с музыкой многих известных и малоизвестных исполнителей разных жанров.

Актуальность онлайн-магазина виниловых пластинок в настоящее время растет, и вряд ли в ближайшее время спадет. Вот несколько причин:

1. Многие исполнители выпускают свои альбомы на виниловых пластинках, добавляют туда также новые треки, которые не вошли в сам альбом на стриминговых площадках. Также создают специальные оформления для винила, что привлекает коллекционеров.
2. Звучание на виниловых пластинках уникальное, звучит совсем по-другому, сравниваю со стриминговыми сервисами.
3. На данный момент в России нет официальных дистрибьютеров виниловых пластинок известных лейблов, а спрос на пластинки все также есть. Создание магазина виниловых пластинок закроет спрос.
4. Создание магазина именно в онлайн очень удобно не только для продавца, но и для пользователей в первую очередь: не нужно идти в сам магазин для выбора и приобретения пластинки, на сайте будет реализован механизм оформления доставки.
   1. Объект исследования**:** весь процесс приобретения виниловой пластинки: начиная от его выбора, заканчивая оформлением заказа.
   2. Предмет исследования**:** люди, которые хотят приобрести виниловую пластинку своего любимого музыкального артиста.
   3. Цели исследования: разработать онлайн-магазин виниловых пластинок. Изучить функциональность существующих площадок по продаже виниловых пластинок.
   4. Задачи исследования:
5. Создать реляционную базу данных для хранения данных пользователей, их заказов и ассортимента онлайн-магазина.
6. Написать API интерфейс для взаимодействия веб-приложения с базой данных.
7. Наполнить базу данных минимальным количеством данных для демонстрации работы всего проекта.
8. Реализовать в веб-приложении следующую функциональность:

* Отображение виниловых пластинок на странице веб-приложения;
* Поиск и фильтрация среди всего ассортимента онлайн-магазина;
* Корзина для заказа;
* Оформление заказа;
* Вход и регистрация в профиль пользователя;
* Отображение истории заказов в профиле пользователя;
* Отображение контактов онлайн-магазина для обратной связи.

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

В теоретической части курсовой работы будет представлена информация об используемых технологиях, инструментах и фреймворках в процессе разработки веб-приложения. Будет произведен анализ основных методов и подходов, применяемых в веб-разработке, а также рассмотрены принципы выбора конкретных технологий для реализации проекта.



## Фронтенд и бэкенд

Фронтенд и бэкенд — это две ключевые составляющие веб-приложений и сервисов, работающие в тесной взаимосвязи.

Фронтенд (англ. Front-end) относится к той части приложения, с которой непосредственно взаимодействует пользователь. Это включает в себя пользовательский интерфейс (UI), представленный в браузере, и логику, управляющую этим интерфейсом. Фронтенд-разработчики используют языки программирования, такие как HTML, CSS и JavaScript, для создания интерактивных веб-страниц и приложений.

Бэкенд (англ. Back-end) отвечает за обработку данных, хранение информации, выполнение бизнес-логики и обеспечение взаимодействия с внешними сервисами. Бэкенд-разработчики используют различные языки программирования, такие как Python, Ruby, PHP, Java, и базы данных, например, MySQL, PostgreSQL, для создания серверной части приложения.

Взаимодействие между фронтендом и бэкендом осуществляется через HTTP-запросы и ответы, где фронтенд отправляет запросы на сервер, а бэкенд обрабатывает эти запросы и возвращает данные, которые затем отображаются на фронтенде. Это позволяет создавать динамические и интерактивные веб-приложения.

## Реляционная база данных

База данных — это организованная структура для хранения, управления и обработки больших объёмов информации. Она позволяет систематизировать данные, обеспечивая быстрый доступ к ним и их изменение.

Базы данных используются в различных областях, включая бизнес, науку, медицину и образование. Существует несколько типов баз данных, но одним из самых распространённых является реляционная база данных. В ней данные представлены в виде таблиц, связанных друг с другом.

Для работы с базами данных используются системы управления базами данных (СУБД). Они предоставляют инструменты для создания, редактирования, удаления и поиска данных, а также обеспечивают их защиту и целостность.

В большинстве веб-приложений используется реляционная база данных. Реляционная база данных — это тип базы данных, который организует данные в одну или несколько таблиц или отношений, каждая из которых имеет уникальное имя и состоит из набора строк и столбцов. Данные в реляционной базе данных структурированы и организованы, что облегчает их поиск, извлечение и управление.

Данные в реляционной базе данных обычно хранятся в нормализованном виде. Данные разбиваются на более мелкие связанные таблицы, каждая из которых имеет свой уникальный ключ или идентификатор. Связи между этими таблицами определяются с помощью внешних ключей, которые связывают данные в одной таблице с данными в другой.

Реляционные базы данных обеспечивают гибкий и масштабируемый способ хранения и управления большими объемами данных, гарантируя при этом целостность и непротиворечивость данных с помощью ограничений, таких как первичные ключи и внешние ключи.

В качестве системы управления базой данных я использую MySQL - система управления базами данных (СУБД), которая широко используется для хранения и управления структурированными данными. Она является одной из наиболее популярных и широко распространенных открытых реляционных баз данных.

MySQL поддерживает множество операционных систем и предоставляет возможность эффективного управления базами данных любого размера. Обладает мощными возможностями, такими как поддержка транзакций, хранение процедур и функций, а также масштабируемость, что позволяет ей удовлетворить потребности различных видов приложений - от небольших веб-сайтов до крупных корпоративных систем.

Все операции с организацией базы данных будут происходить в MySQL Workbench — инструмент для визуального проектирования баз данных, который объединяет в себе проектирование, моделирование, создание и эксплуатацию баз данных MySQL. Он был разработан корпорацией Oracle и доступен в трёх редакциях: Community Edition (распространяется под свободной лицензией GNU GPL), Standard Edition (доступна по ежегодной платной подписке) и Enterprise Edition.

MySQL Workbench предлагает следующие возможности:

* наглядное представление модели базы данных в графическом виде;
* удобный механизм установки связей между таблицами, в том числе «многие ко многим»;
* возможность восстановления структуры таблиц из уже существующей на сервере БД;
* редактор SQL-запросов с возможностью отправки их серверу и получения ответа в виде таблицы;
* редактирование данных в таблице в визуальном режиме.

## API сервер

API (Application Programming Interface) — это инструмент часть бэкенда, который определяет методы и структуры для взаимодействия различных программ между собой. С помощью API разработчики могут интегрировать функционал и данные из одного приложения или сервиса в другое. API применяется для достижения различных целей:

* Взаимодействие с внешними системами. API позволяет приложениям обмениваться данными и взаимодействовать с другими системами, сервисами и приложениями.
* Разработка расширений. API даёт возможность разработчикам создавать расширения и плагины для существующих приложений, расширяя их функционал.
* Создание веб-сервисов. API используется для создания веб-сервисов, предоставляющих доступ к данным и функционалу через интернет.
* Интеграция со сторонними сервисами. API позволяет интегрировать данные и функционал сторонних сервисов в приложение, таких как социальные сети, платёжные системы и другие API.

В своём проекте я использую решение от Майкрософт – ASP.NET Core. С его помощью можно создавать веб-сайты, веб-службы и мобильные приложения.

Фреймворк открытый и модульный, он поддерживает разные операционные системы: Windows, macOS и Linux. Также у него есть встроенная поддержка облака, микросервисной архитектуры и контейнеризации, поэтому он подходит для современных развёртываний приложений.

Вот основные возможности ASP.NET Core:

* встроенная поддержка веб-сервера Kestrel и возможность интеграции с другими веб-серверами, например IIS;
* поддержка модульного и гибкого конфигурирования приложений;
* использование современных шаблонов разработки, таких как Model-View-Controller (MVC) и Razor Pages;
* встроенная поддержка веб-сервисов, API и аутентификации;
* возможность развёртывания приложений в контейнерах.

Сам API сервер будет писаться в Visual Studio 2024 – интегрированная среда разработки (IDE) от компании Майкрософт, предназначенная для создания различных типов приложений. Она включает в себя инструменты для разработки как консольных приложений, так и игр и приложений с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, UWP, а также веб-сайтов, веб-приложений и веб-служб.

Visual Studio позволяет разрабатывать приложения для различных платформ, поддерживаемых Windows, включая Windows, Windows Mobile, Windows CE, а также для .NET Framework, .NET Core, .NET, MAUI, Xbox, Windows Phone и SilverLight. После покупки компании Xamarin корпорацией Microsoft появилась возможность разработки приложений для iOS и Android.

Основные возможности Visual Studio включают в себя:

* редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода;
* встроенный отладчик, способный работать как на уровне исходного кода, так и на уровне машинного кода;
* инструменты для создания и подключения сторонних дополнений (плагинов) для расширения функциональности;
* поддержку систем контроля версий исходного кода, таких как Subversion и Visual SourceSafe;
* поддержку новых наборов инструментов для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования.

## Фреймворк

Фреймворк в разработке программного обеспечения представляет собой программную платформу, которая определяет структуру программной системы. Он упрощает процесс разработки и интеграции различных компонентов крупного программного проекта, предоставляя разработчикам набор готовых решений и абстракций для ускорения разработки.

В качестве основного фреймворка для разработки веб-приложения я использовал Next.js — это открытый JavaScript фреймворк, созданный поверх React.js (JavaScript библиотека с открытым исходным кодом, разработанная и поддерживаемая Facebook\*, Instagram\* (\*Признаны экстремистскими организациями и запрещены на территории РФ) и сообществом разработчиков. Она предназначена для создания пользовательских интерфейсов и позволяет разработчикам создавать интерактивные и динамичные веб-приложения. Он был разработан компанией Vercel (ранее известной как ZEIT). В проекте он используется для написания фронтенда и части бэкенда (отправка, получение и обработка данных, которые получаются через API запросы).

Фреймворк предназначен для решения проблемы React.js, связанной с отрисовкой приложения на стороне сервера (SSR), и дополняет React.js удобными решениями по навигации между страницами. Он работает как на сервере, так и в браузере, что позволяет ускорить загрузку страниц и улучшить SEO-оптимизацию. Благодаря принципу SSR, большая часть операций, связанных с отображением приложения, выполняется на сервере, что снижает нагрузку на клиентское устройство. Это особенно полезно для мобильных устройств и пользователей с медленным интернетом.

Next.js предлагает файловый подход к маршрутизации, что упрощает навигацию между страницами.

Разработка веб-приложений на Next.js основывается на нескольких ключевых принципах:

1. Использование библиотеки React: Next.js построен на базе React, что позволяет разработчикам использовать всю мощь этой библиотеки для создания пользовательских интерфейсов.
2. Server Side Rendering (SSR): Next.js автоматически генерирует HTML на сервере, что позволяет ускорить загрузку страниц и улучшить их SEO-оптимизацию. Это особенно полезно для статических страниц, где контент не меняется часто.
3. Code-Splitting: Next.js позволяет разбивать код на отдельные части, которые загружаются только тогда, когда они необходимы. Это помогает уменьшить размер начального пакета и ускорить загрузку приложения.
4. Оптимизация производительности: Next.js предоставляет инструменты для оптимизации производительности, такие как автоматическая предварительная загрузка данных и оптимизация запросов к серверу.
5. Интеграция с внешними API: Next.js упрощает взаимодействие с внешними API, позволяя разработчикам легко получать данные и отображать их в приложении.
6. Легкая настройка и развертывание: Next.js предоставляет простой способ настройки проекта и его развертывания на различных хостинговых платформах.

Написание фронтенда веб-приложения будет происходить в Visual Studio Code — бесплатный редактор исходного кода, разработанный компанией Майкрософт. Он доступен на платформах Windows, Linux и macOS и поддерживает множество языков программирования, включая JavaScript, TypeScript, Python, C#, C++, Java и другие.

Visual Studio Code обладает широким набором функций, среди которых:

* подсветка синтаксиса;
* IntelliSense;
* рефакторинг кода;
* отладка;
* интеграция с системами управления версиями Git;
* поддержка множества тем оформления;
* возможность создания пользовательских настроек.

## Тестирование, отладка

**Тестирование** в контексте разработки программного обеспечения — это процесс, направленный на проверку и оценку качества программного продукта. Основная цель тестирования заключается в выявлении ошибок, дефектов или недостатков в программном обеспечении для их последующего устранения, что помогает убедиться в том, что продукт соответствует поставленным спецификациям и требованиям как функциональным, так и нефункциональным.

Тестирование позволяет не только обнаружить дефекты в программе, но и выявить, насколько продукт удобен для пользователя, безопасен, и способен работать эффективно в различных условиях. Оно включает в себя ряд методик и подходов, таких как модульное тестирование, интеграционное тестирование, системное тестирование, приемочное тестирование, а также статические методы тестирования, такие как анализ кода и ревью кода.

Процесс тестирования может быть автоматизирован или проводиться вручную и часто разделяется на несколько уровней, каждый из которых направлен на определенные аспекты программы.

**Отладка безопасности** — это процесс анализа программного обеспечения с целью идентификации и устранения уязвимостей, которые могут быть эксплуатированы злоумышленниками для нанесения вреда системе или нарушения конфиденциальности, целостности и доступности данных. Этот процесс включает в себя проверку кода на предмет известных уязвимостей, проверку на наличие неправильно реализованных функций, которые могут привести к брешам в безопасности, а также использование различных методик и инструментов для тестирования безопасности приложений.

Процесс отладки безопасности может включать, но не ограничивается, следующими действиями:

1. Проведение анализа исходного кода (Static Application Security Testing, SAST) для обнаружения уязвимостей, таких как буферные переполнения, инъекции SQL, кросс-сайтовый скриптинг (XSS) и другие.

2. Динамический анализ приложений (Dynamic Application Security Testing, DAST), который проводится во время выполнения программы для выявления ошибок, которые не были обнаружены на этапе статического анализа.

3. Тестирование на проникновение (Penetration Testing), которое имитирует атаки злоумышленников для проверки устойчивости системы к внешним угрозам.

4. Анализ зависимостей (Dependency Scanning) для проверки сторонних библиотек и компонентов на предмет известных уязвимостей.

5. Обучение и разработка включает в себя создание программного обеспечения с учетом принципов безопасной разработки и использование методик безопасного кодирования.

6. Постоянное обновление и патчинг программного обеспечения для устранения найденных уязвимостей и минимизации риска атак.

Отладка безопасности является важной частью процесса разработки и поддержки программного обеспечения, стремящейся к созданию более безопасных и надежных IT-решений. Этот процесс помогает гарантировать, что приложения и системы защищены от потенциальных угроз, что особенно критично для обеспечения защиты персональных данных и предотвращения возможных финансовых потерь.

В то время как тестирование безопасности направлено на защиту программы от внешних атак, отладка ориентирована на обеспечение корректной работы самой программы внутренне.

**Отладка** — это процесс поиска и устранения ошибок в программном коде, используемый для обнаружения и исправления дефектов, которые препятствуют корректной работе программного обеспечения. Отладка помогает программистам улучшить функциональность, эффективность и стабильность приложений или систем, тем самым упрощая разработку и повышая качество конечного продукта.

В процессе отладки программисты анализируют код приложения, чтобы выяснить причины неожиданного поведения, ошибок или сбоев. Это может включать просмотр журналов ошибок, использование дебаггеров для пошагового выполнения программы, добавление вывода отладочной информации и проверку потоков данных и переменных во время выполнения программы. Отладка часто требует глубокого понимания логики программирования и может быть долгим процессом, особенно в сложных системах с множеством взаимодействующих компонентов.

Существуют различные инструменты и техники для эффективной отладки, включая:

Интерактивные дебаггеры: позволяют разработчикам делать "паузы" в выполнении программы на определенных точках (брейкпоинты), шаг за шагом просматривать исполнение кода, изменять значения переменных и анализировать стек вызовов.

* Логирование: запись данных о работе программы, которые можно анализировать для обнаружения аномалий и ошибок.
* Профайлирование: оценка производительности различных частей программы для выявления узких мест и оптимизации работы.
* Unit тесты: написание и использование тестов для автоматической проверки отдельных блоков программы на соответствие ожидаемым результатам.
* Статический анализ кода: инструменты для анализа источников кода на предмет нарушений стилей кодирования, потенциальных ошибок и уязвимостей без его выполнения.

## Теоретические аспекты построения web-сайта

Построение и функционирование веб-сайта охватывает множество технологий и дисциплин, которые можно разделить на несколько ключевых теоретических аспектов:

1. Архитектура веб-сайта:

* Frontend (клиентская сторона): занимается всем, что пользователь видит и взаимодействует непосредственно в своем браузере. Основные технологии включают HTML (структура страницы), CSS (визуальное оформление), и JavaScript (интерактивность).
* Backend (серверная сторона): обеспечивает логику работы веб-сайта, обработку данных, которые вводит пользователь, и выдачу данных в формате, который может быть обработан клиентской стороной. Часто используются серверные языки программирования, такие как PHP, Python, Ruby, Java, JavaScript (Node.js), и базы данных такие как MySQL, PostgreSQL, MongoDB.

2. Клиент-серверная модель:

Веб-сайты работают по модели запроса и ответа. Клиент (браузер) отправляет запрос на сервер посредством HTTP/HTTPS-протоколов, сервер обрабатывает запрос и возвращает ответ, который клиент затем интерпретирует и отображает.

3. Дизайн пользовательского интерфейса и опыта (UI/UX):

* UI означает создание эффективного и приятного визуального представления интерфейса сайта.
* UX фокусируется на опыте пользователя при взаимодействии с сайтом, включая удобство и интуитивность использования.
* Принципы проектирования включают создание прототипов, макетов и wireframes.

4. Производительность и оптимизация:

Включает в себя минимизацию ресурсов (сжатие изображений, минимизация CSS и JavaScript), использование кэширования, оптимизацию времени загрузки страниц и улучшение общей отзывчивости сайта.

5. Доступность (Accessibility):

Создание веб-сайтов, доступных для людей с ограниченными возможностями, включая соответствующие стандарты (например, WCAG - Руководство по обеспечению доступности контента).

6. Безопасность:

Основные аспекты безопасности включают работы с HTTPS, защиту от XSS (cross-site scripting), CSRF (cross-site request forgery) атак и SQL инъекций, обеспечение данных и идентификация пользователя через безопасные сессии и аутентификацию.

7. SEO оптимизация:

Обеспечение того, чтобы сайт был виден и хорошо индексируемым поисковыми системами. Оптимизация включает правильное использование тегов, метаданных, адаптивный дизайн для мобильных устройств, скорость загрузки страниц и ценное контентное наполнение.

Эти аспекты формируют основу для разработки эффективных, безопасных и удобных для пользователя веб-сайтов. Понимание каждого из этих элементов и их взаимодействие является ключевым для создания профессионального и успешного онлайн-присутствия.

# ОСНОВНАЯ ЧАСТЬ



## Требования к системе

У большинства онлайн-магазинов должны быть: поиск по ассортименту с фильтрами, разработан механизм работы корзины, оформление заказа (адреса доставки, их способы, также формы оплаты), создание профиля и вход в него для сохранения основных данных пользователя, история заказов и другое.

В плане графического дизайна приложение должно быть интуитивно понятно, не нагружено, адаптировано под разные размеры экранов. Вёрстка должна быть стабильна, не съезжать при изменении платформы, пополнения контентом. Желательно иметь единую айдентику для всех разделов приложения, единую цветовую палитру и элементы управления (кнопки, указатели, поля ввода, карточки товаров и другое).

Система должна быть надёжна и отказоустойчива: выдерживать высокие нагрузки, масштабируемость, выдерживание хакерских атак, предотвращение утечек персональных данных пользователей.

## Тестовый сценарий

Для проверки своего веб-приложения я провел некоторые ручные тесты, чтобы выявить ошибки в проектировании, либо в коде. Они будут прикреплены в приложениях к пояснительной записке в виде таблицы по стандартам WorldSkills (Приложение А).

Тестовый сценарий — это подробное описание последовательности действий, которые необходимо выполнить для проверки работоспособности и функциональности приложения или сайта. Он включает в себя стартовые условия, входные данные, ожидаемые действия пользователя и ожидаемые результаты. Тестовые сценарии помогают выявить ошибки и недочеты в работе системы, а также обеспечивают ее стабильность и надежность.

## Этапы разработки ИС

**База данных**

Как указано в теоретической части, я использовал для создания базы данных MySQL Workbench.

В моей базе данных всего шесть таблиц, каждая из них отвечает за конкретную информацию, с которой взаимодействует информационная система (Приложение Б).

|  |  |
| --- | --- |
| id\_orders | Первичный ключ. Уникальное числовое значение, которое даётся каждому заказу |
| user | Внешний ключ. Он связывает таблицы Orders и Users |
| status | Символьный тип. Он хранит в себе статус заказа (не оплачен, принят, в сборке, отправлен, доставлен) |
| cost | Числовой тип. Хранит сумму всего заказа |
| payment\_method | Символьный тип. Хранит в себе метод оплаты (картой онлайн, наличные при получении) |
| payment\_status | Символьный тип. Хранит в себе статус оплаты (оплачен, не оплачен) |
| ship\_method | Символьный тип. Хранит в себе способ доставки (Почта России, СДЕК, Пункт выдачи) |
| adress | Символьный тип. Хранит в себе адрес доставки |

1. Orders.  
   Эта таблица хранит в себе данные о заказах пользователей (таблица 1).

Таблица 1 – Таблица orders в БД приложения.

1. Users.  
   Эта таблица хранит в себе данные о зарегистрированных пользователях (таблица 2).

Таблица 2 – Таблица users в БД приложения.

|  |  |
| --- | --- |
| id\_users | Первичный ключ. Хранит в себе уникальный ID пользователя |
| email | Символьный тип. Хранит в себе электронную почту пользователя |
| phone\_number | Символьный тип. Хранит в себе номер телефона пользователя |
| name | Символьный тип. Хранит в себе имя пользователя |
| password | Символьный тип. Хранит в себе пароль пользователя |

|  |  |
| --- | --- |
| id\_adresses | Первичный ключ |
| index | Символьный тип. Хранит в себе индекс адреса доставки |
| address | Символьный тип. Хранит в себе адрес доставки |
| iduser | Внешний ключ. Связывает таблицы Adresses и Users |

1. Adresses.  
   Эта таблица хранит в себе адреса доставки пользователей (таблица 3).

Таблица 3 - таблица addresses в БД приложения.

1. Order\_item.  
   Эта таблица хранит в себе данные о позициях в заказе пользователя (таблица 4).

|  |  |
| --- | --- |
| idorder\_item | Первичный ключ |
| orderid | Внешний ключ. Связывает таблицы Order\_item и Orders |
| item\_vinylid | Численный тип. Хранит в себе ID виниловой пластинки |

Таблица 4 – таблица order\_item в БД приложения.

1. Tracklists.  
   Эта таблица хранит в себе список треков в пластинке (таблица 5).

Таблица 5 – таблица tracklists в БД приложения.

|  |  |
| --- | --- |
| id\_tracklists | Первичный ключ |
| vinylid | Внешний ключ. Связывает таблицы Tracklists и Vinyls |
| tracks | Символьный тип. Хранит в себе список треков |

1. Vinyls.  
   Эта таблица хранит в себе данные виниловых пластинок, которые представлены в интернет-магазине (таблица 6).

Таблица 6 – таблица vinyls в БД приложения.

|  |  |
| --- | --- |
| id\_vinyls | Первичный ключ |
| name | Символьный тип. Хранит в себе название пластинки |
| artist | Символьный тип. Хранит в себе имя исполнителя |
| year | Численный тип. Хранит в себе год выпуска |
| genre | Символьный тип. Хранит в себе жанр пластинки |
| cost | Численный тип. Хранит в себе стоимость пластинки |
| edition | Символьный тип. Хранит в себе версию издания пластинки |
| cover | Символьный тип. Хранит в себе файловый путь до обложки пластинки |
| status | Символьный тип. Хранит в себе статус наличия пластинки (instock, outstock) |

**UX/UI веб-приложения**

В приложении всего 5 разделов: Главная, Каталог, Профиль, Корзина и Контакты. Также есть страницы со списком треков пластинки и авторизации.

На каждой странице закреплен header с панелью навигации по приложению (рисунок 1).

Рисунок 1 - Header по всему приложению

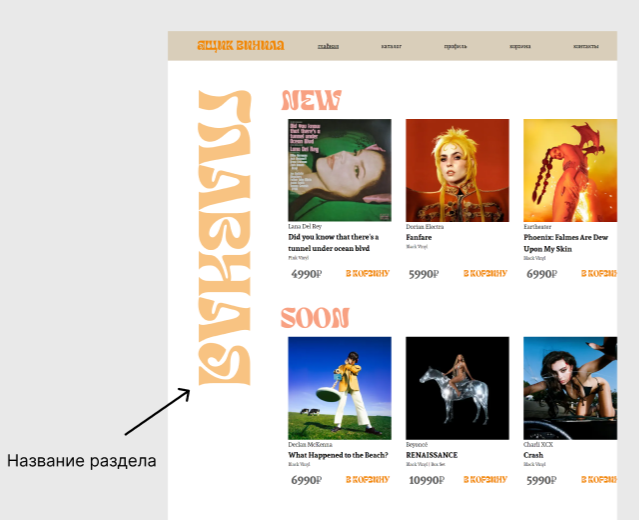
Слева от центра страницы указано **название раздела** (рисунок 2).

Рисунок 2 - Расположение названия раздела

На **главной странице** представлены блоки с винилами: новые и будущие поступления. Также в конце страницы указана информация о самом магазине.

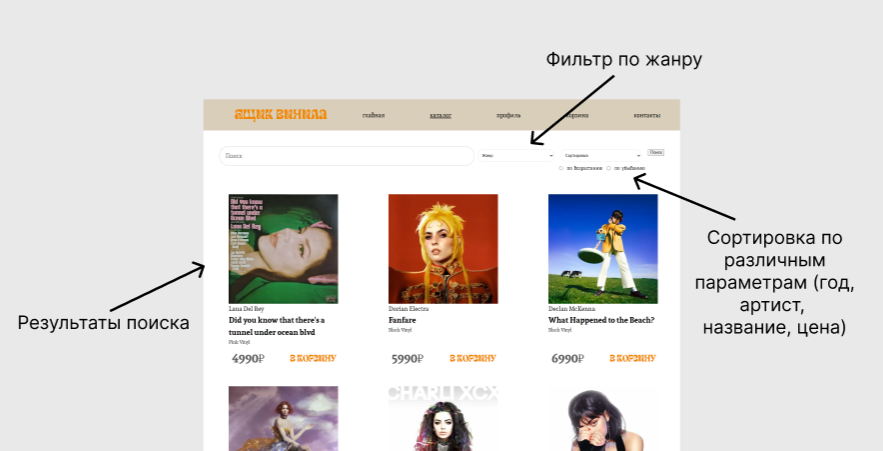
На **странице Каталог** есть компонент для ввода данных поиска по ассортименту магазина: поле поиска по ключевым словам, фильтр по жанру, сортировка по различным данным (год, артист, название, цена), также параметры сортировки: по возрастанию и по убыванию (рисунок 3).

Рисунок 3 - Описание компонентов в разделе Каталог

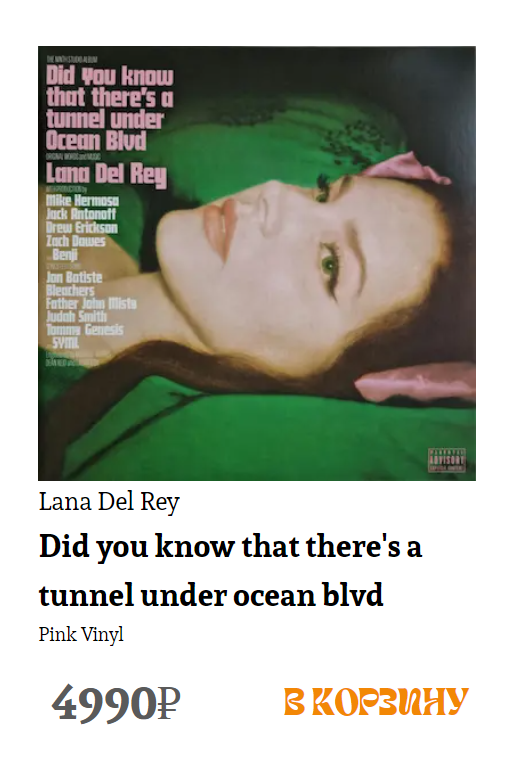
На карточке каждой пластинки есть информация: название пластинки, исполнитель, цена пластинки и его издание, обложка (рисунок 4). Также есть кнопка **В корзину**, которая добавляет пластинку в корзину.

Рисунок 4 – Карточка пластинки

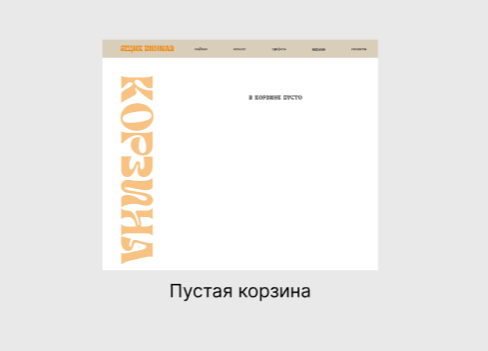
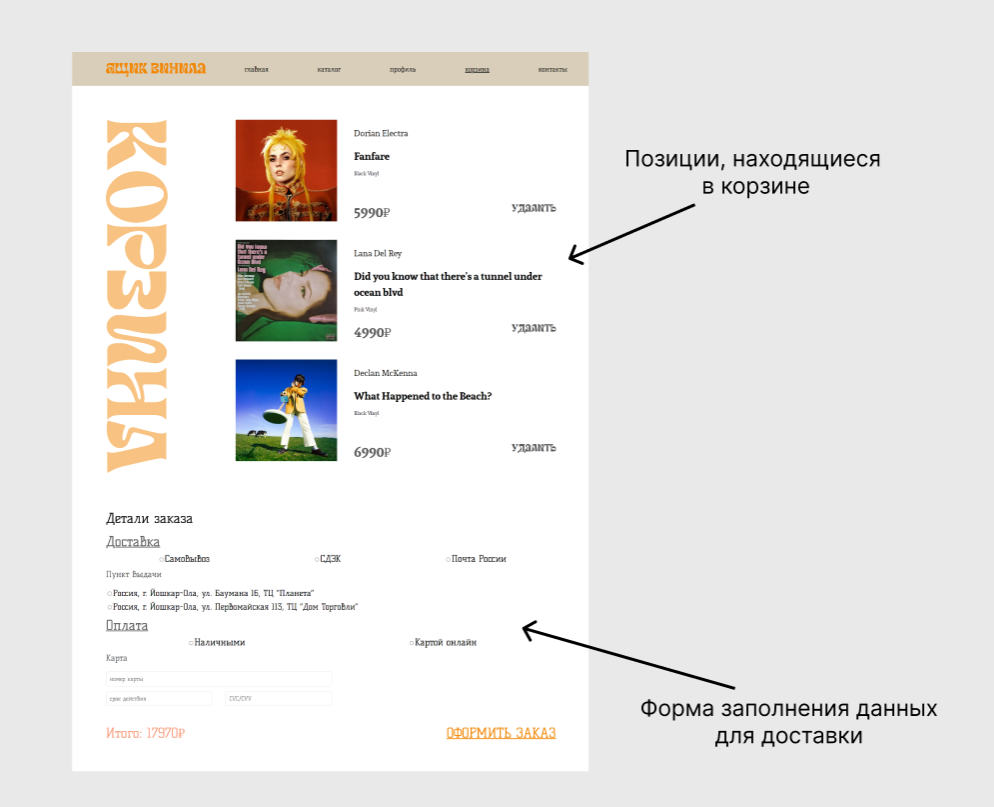
В **разделе Корзина** имеются два состояния страницы: пустая корзина (рисунок 5) и с позициями (рисунок 6). На странице корзины, где уже имеются позиции, есть два блока: позиции в корзине и форма ввода данных для доставки и оплаты. Для каждой позиции есть кнопка удаления, благодаря которой пользователь может удалить не нужную позицию из корзины.

Рисунок 5 - Страница пустой корзины

Рисунок 6 - Страница корзины, в которую добавили позиции

Далее идёт форма ввода, в ней указывается следующая информация: способ и адрес доставки, способ оплаты и ввод реквизитов карты. В зависимости от выбора способа доставки появляются разные поля: при выборе **Самовывоз**, появляется список адресов пунктов выдачи магазина; при выборе **Почта России/СДЕК**, появляется поле для ввода почтового адреса. Такая же логика есть для способа оплаты: если выбирается **Наличными**, то появляется напоминание, чтобы покупатель подготовил наличные перед забором заказа; если выбирается **Картой онлайн**, то появляется форма с вводом данных карты. При выборе оплаты онлайн далее должна открываться страница банка для подтверждения оплаты, но это реализуется при использовании банковского API оплаты.

Под самой формой отображается конечная сумма заказа и кнопка оформления заказа. Доставка бесплатная для всех способов доставки.

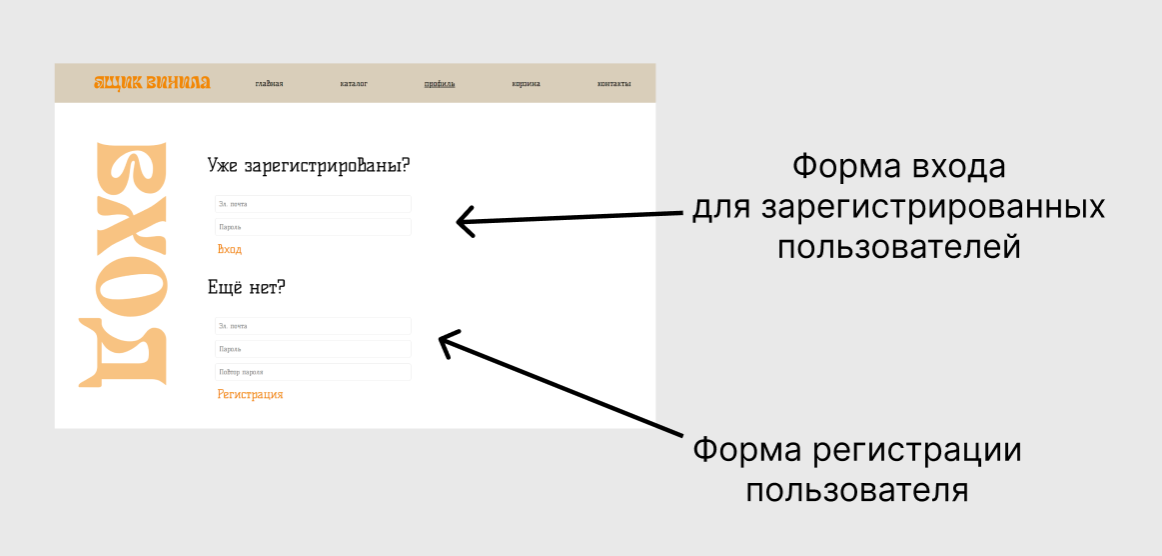
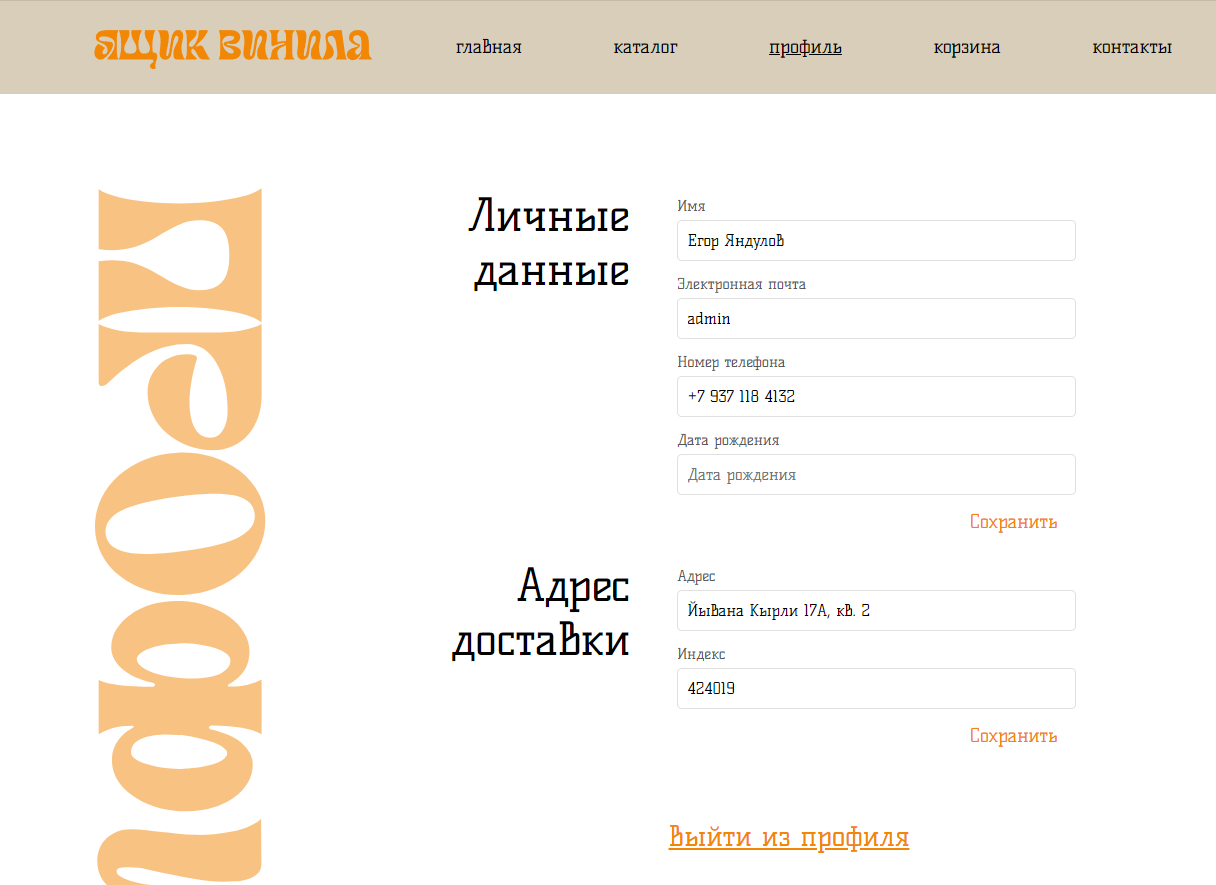
Для **раздела Профиль** есть два состояния: авторизованный пользователь (рисунок 8) и не авторизованный пользователь (рисунок 7). При не авторизованном открывается страницы с двумя формами: для авторизации в уже зарегистрированный профиль и регистрации нового. Регистрация и вход происходят при помощи электронной почты и пароля. Для регистрации имеется простой механизм проверки пароля.

Рисунок 7 - Раздел Профиль для не авторизованного пользователя

Рисунок 8 - Блок с личными данными авторизованного пользователя

На странице для авторизованных пользователей есть два основных блока: личные данные и адреса доставки (рисунок 9).

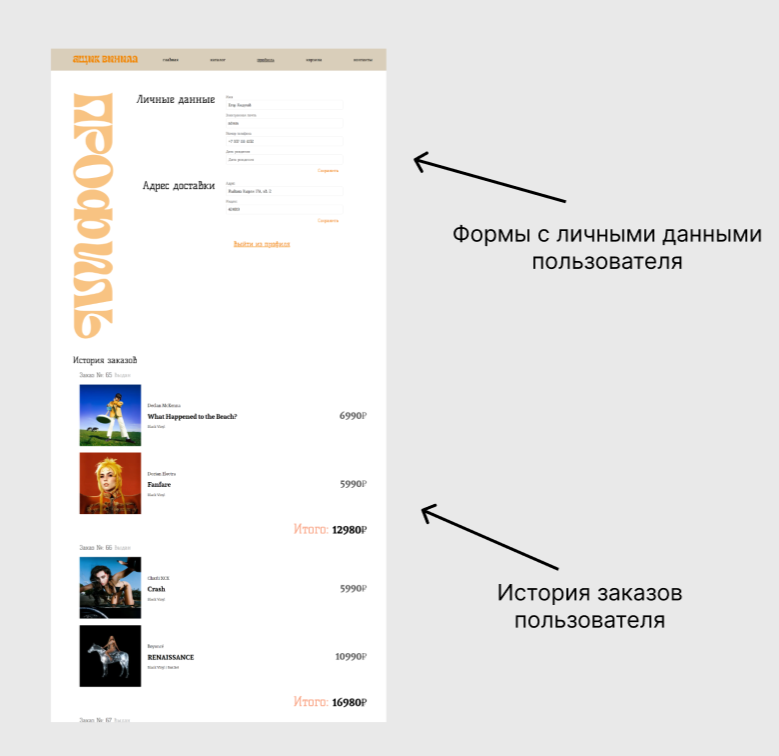
В блоке с личными данными указана следующая информация: имя, электронная почта, номер телефона, дата рождения. В отдельном блоке выделен индекс и адрес для доставки. Ниже блока есть кнопка для выхода из профиля.

Рисунок 9 - Страница Профиль для авторизованного пользователя

Далее идёт блок с историей заказов пользователя (рисунок 10). В карточке каждого заказа имеется следующая информация: номер заказа, его статус, позиции, сумма заказа.

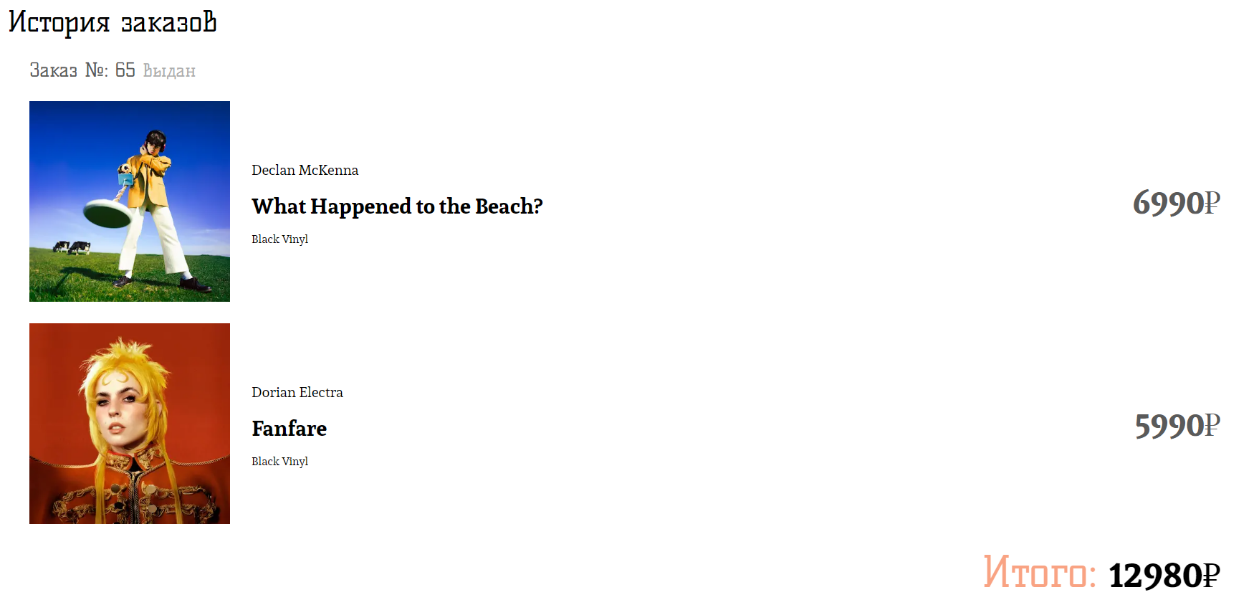


Рисунок 10 - Пример заказа из истории заказов пользователя

В **разделе Контакты** представлены ссылки на социальные сети магазина для связи (рисунок 11).

Рисунок 11 - Раздел Контакты

На странице каждой пластинки представлен треклист (рисунок 12).

Рисунок 12 - Страница пластинки

Для вёрстки веб-приложения используется JavaScript фреймворк Next.js, основанный на React.

В веб-приложении также реализована адаптивная верстка под различные размеры экрана.

Для каждого раздела приложения создаётся своя папка в проекте. В каждой из таких папок находятся как минимум два файла: один JS с логикой и вёрсткой, а второй CSS для стилей.

В фреймворке Next.js используется система маршрутизации, основанная на файловой системе. Это одна из особенностей Next.js, которая делает его простым в использовании, позволяя разработчикам быстро создавать и масштабировать приложения. Вот как работает маршрутизация в Next.js:

1. Структура каталогов: Маршрутизация в Next.js строится вокруг структуры каталогов внутри папки pages. Каждый файл JavaScript, TypeScript, или JSX в этой папке автоматически становится доступным как маршрут.

2. Файлы и пути:

* Если вы создаете файл pages/about.js, он будет доступен по пути /about.
* Главная страница проекта обычно находится в pages/index.js, которая соответствует корневому маршруту (/).

3. Динамические маршруты:

Для создания динамических маршрутов используются квадратные скобки. Например, файл pages/posts/[id].js отвечает за маршруты вида /posts/1, /posts/2 и так далее, где id может быть любой строкой или числом. Такая логика используется для отображения страницы пластинки с треклистом.

Для получения значения id внутри компонента необходимо использовать хук useRouter из next/router, который позволяет получить доступ к router.query.id.

4. Вложенные маршруты:

Папки внутри pages используются для группировки и вложенных маршрутов. Например, файл pages/blog/first-post.js будет доступен по маршруту /blog/first-post.

5. Обработка ошибок:

Next.js автоматически обрабатывает ошибки 404, подставляя собственную страницу ошибки, если файл для запрашиваемого маршрута не найден. Вы также можете переопределить эту страницу, создав pages/404.js.

6. Программная навигация:

Навигацию между страницами можно осуществлять программно с помощью useRouter или компонента Link из next/link. <Link> обеспечивает клиентскую навигацию без полной перезагрузки страницы, что позволяет приложению работать быстрее и плавнее.

7. Префетчинг страниц:

Next.js автоматически предварительно загружает страницы в фоновом режиме, которые могут быть посещены пользователем. Это делает переходы между страницами быстрыми. Предварительная загрузка выполняется при обнаружении ссылок на страницы, видимых пользователю.

В некоторых случаях в проекте создаётся отдельный JS файл для программирования логики отдельного компонента страницы. Например, в разделе Каталог для пластинок есть файл vinylblocksearched.js. В нём находится код для запроса данных по API. Далее полученные данные используются для рендеринга **компонента** VinylCard.

В Next.js компоненты являются основными строительными блоками приложений и используются для организации пользовательского интерфейса. Все компоненты в Next.js основаны на React, что означает, что они используют те же основные концепции, такие как JSX, состояние, пропсы, жизненный цикл компонентов и хуки. Вот основы по работе с компонентами в Next.js:

**Основные типы компонентов**

1. Функциональные компоненты:

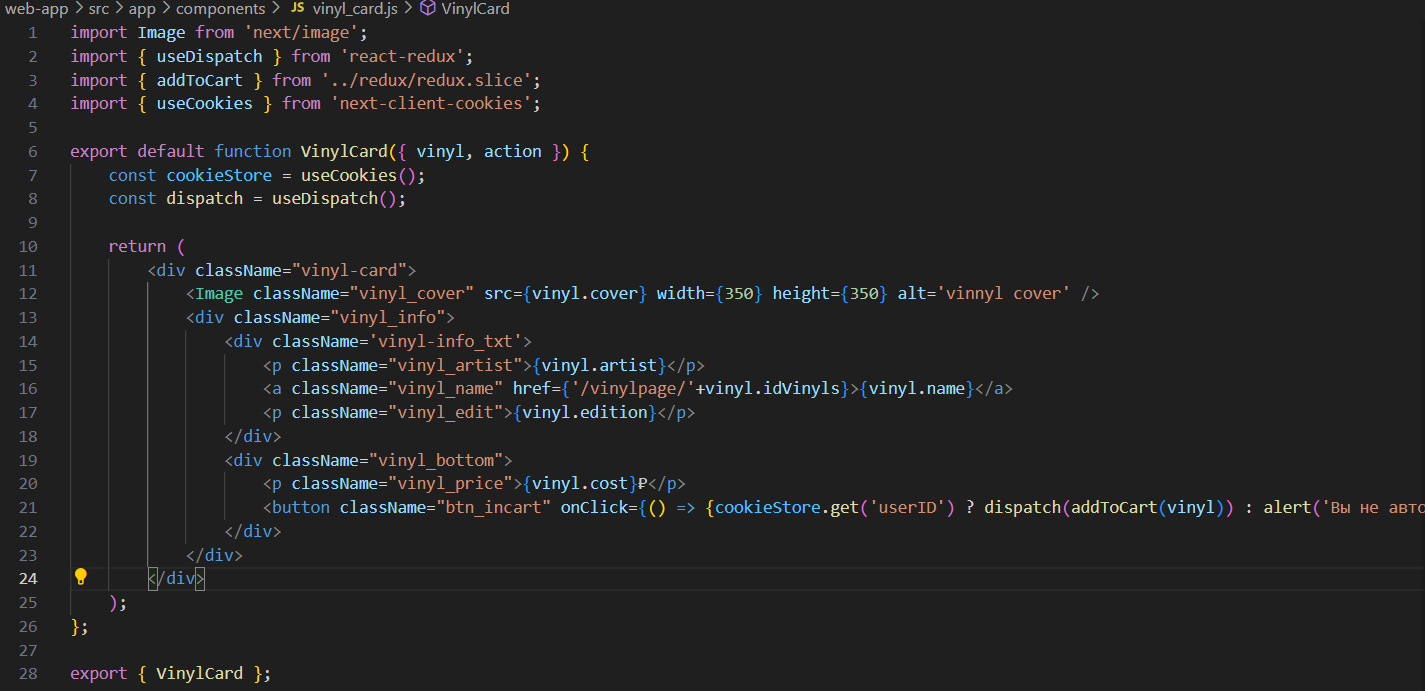
Это наиболее распространенный и современный способ создания компонентов в React и Next.js. Они представляют собой функции JavaScript, которые возвращают JSX. Функциональные компоненты легко читать и тестировать, а также они поддерживают хуки, что делает их мощным инструментом для управления состоянием и другими фичами React.

Рисунок 3.4 - Пример кода для компонента vinylcard

2. Классовые компоненты:

Это более старый способ создания компонентов в React. Классовые компоненты могут быть более знакомыми тем, кто пришел в Frontend из ООП языков, но они чаще всего заменяются функциональными компонентами.

**Использование компонентов**

Размещение: компоненты могут быть размещены в любом месте вашего проекта, но обычно они находятся в каталоге components. Это не строгое правило, но помогает поддерживать структуру проекта организованной.

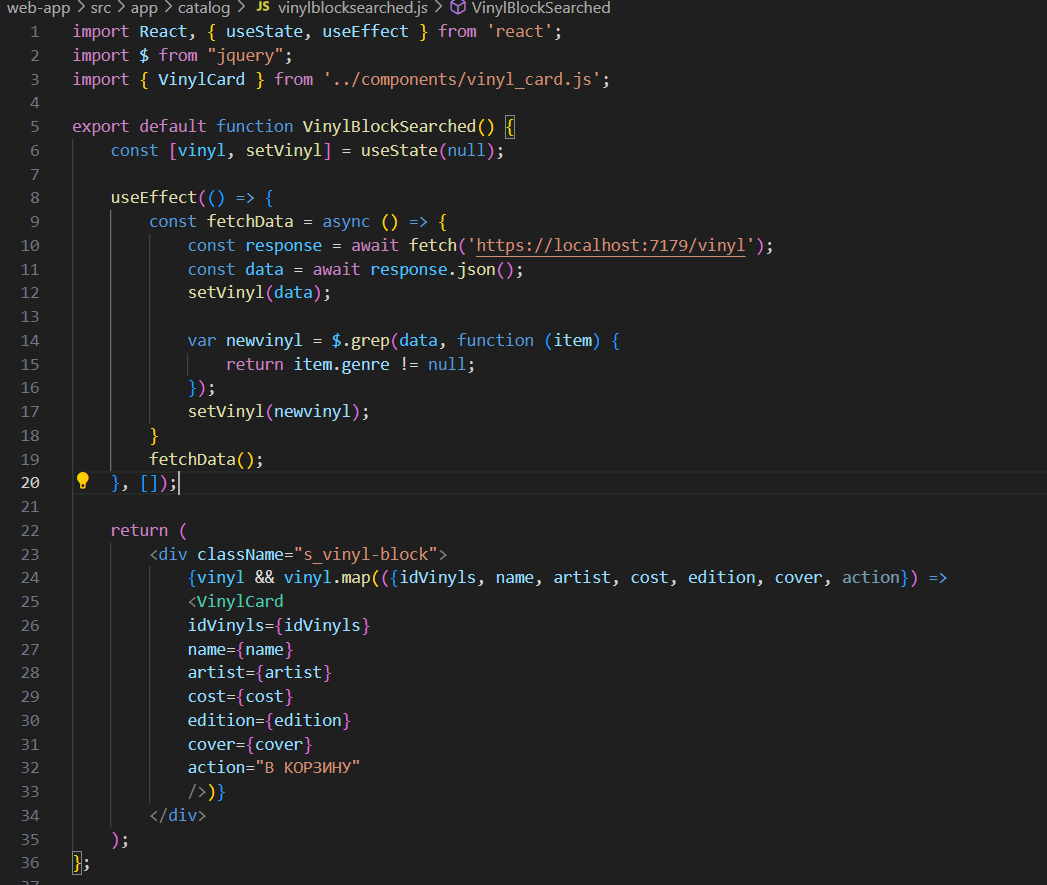
Импорт и использование компонентов: Компоненты импортируются и используются в других компонентах или страницах для сборки приложения.

Рисунок 3.5 - Использование компонента vinylcard в разделе Каталог

**Особенности компонентов в Next.js**

Стили: в Next.js вы можете стилизовать компоненты с помощью CSS-модулей или использовать библиотеки стилей, такие как styled-components или emotion. CSS-модули позволяют ограничить стили компонентов, чтобы они не влияли глобально на всё приложение.

Хуки: например, useRouter для работы с маршрутизацией, useState и useEffect для управления состоянием и жизненным циклом компонентов.

Оптимизация: Next.js автоматически оптимизирует загрузку компонентов с помощью кодового разделения и ленивой загрузки, что повышает производительность приложения.

В написании кода я использовал плагин для Visual Studio Code - GigaCode от Сбера.

GigaCode — это AI-ассистент разработчика, который позволяет ускорить процесс написания кода, предлагая наиболее вероятные и релевантные подсказки. GigaCode анализирует имеющийся контекст и предлагает полные конструкции функций, циклов, условий и других элементов кода, что позволяет ускорить процесс разработки, повысить производительность, а также сделать процесс программирования более продуктивным.

В настоящее время GigaCode поддерживает более 15 языков программирования, а также совместим со множеством популярных редакторов и интегрированных сред разработки, что делает инструмент полезным для большинства разработчиков и позволяет по максимуму использовать возможности GigaCode для улучшения рабочего процесса.

Для написания API приложения использовался фреймворк ASP.NET Core.

Это приложение предоставляет REST API с защитой роутов через базовую аутентификацию. В нём задействованы различные функциональности и методы из ASP.NET Core и Entity Framework для управления данными через API.

**1. Инициализация WebApplication и конфигурация сервисов**

Создание WebApplication:

* Метод WebApplication.CreateBuilder(args) инициализирует новый экземпляр WebApplicationBuilder, который является основой для настройки и создания приложения. Здесь устанавливаются начальные параметры, зависимости и сервисы, которые будут использоваться приложением.
* args: Это массив аргументов командной строки, которые могут передаваться при запуске приложения.

**2. Настройка сериализации JSON**

Добавление и настройка Newtonsoft.Json для сериализации:

* Newtonsoft.Json - это библиотека для работы с JSON в .NET, которая предоставляет широкие возможности для настройки процесса сериализации и десериализации.
* В данном случае используется конфигурация ReferenceLoopHandling.Ignore для предотвращения проблем с циклическими ссылками при сериализации объектов, что часто встречается в объектных графах с взаимосвязями.

**3. Настройка CORS (Cross-Origin Resource Sharing)**

Добавление политики CORS:

CORS (Cross-Origin Resource Sharing) позволяет веб-приложениям осуществлять запросы к серверу из разных источников (доменов). Это особенно важно для SPA (Single Page Applications), которые обычно запрашивают API, расположенные на другом домене или порту.

Применение политики CORS в приложении:

Метод UseCors применяет ранее определенные настройки CORS ко всем входящим запросам. Настраивается так, чтобы разрешить запросы с любого источника (AllowAnyOrigin), использовать любой метод (AllowAnyMethod) и принимать любые заголовки (AllowAnyHeader).

Такие настройки CORS максимально открыты и не стоит их использовать в продакшен-среде без конкретных ограничений, так как это может повысить риски безопасности.

**4. Swagger для документации API:**

Настраивается Swagger и добавляется базовая аутентификация к документации API.

**5. Аутентификация и Авторизация:**

Аутентификация:

app.MapPut(

"/user/{id:int}",

(int id, User postUser) =>

{

using (var context = new MydbContext())

{

var user = context.Users.Find(id);

if (user == null)

return Results.NotFound();

user.Name = postUser.Name;

user.Email = postUser.Email;

user.PhoneNumber = postUser.PhoneNumber;

context.Update(user);

context.SaveChanges();

}

return Results.Ok();

})

.RequireAuthorization()

.WithTags("Пользователи")

.Produces(StatusCodes.Status200OK)

.Produces(StatusCodes.Status401Unauthorized)

.Produces(StatusCodes.Status404NotFound);

Добавление и настройка базовой аутентификации (Basic Authentication):

* В базовой аутентификации логин и пароль передаются через заголовок HTTP запроса, закодированные в формате Base64. На серверной стороне эта строка декодируется, и происходит проверка учётных данных пользователя.
* В коде используется BasicAuthenticationHandler, который является пользовательским обработчиком аутентификации. Этот класс должен быть реализован разработчиком и, как правило, содержит логику для валидации учётных данных.

**Регистрация сервиса аутентификации:**

- Сервис аутентификации регистрируется в контейнере DI (Dependency Injection), что позволяет его использовать в различных частях приложения.

**Авторизация**

Настройка политик авторизации:

* Авторизация в ASP.NET Core позволяет определить, кто может делать что в приложении, часто используя различные политики, которые проверяются перед выполнением определённого действия.
* FallbackPolicy: Это политика по умолчанию, которая применяется в случае, если для конкретного роута не указана иная политика. В данном случае политика требует, чтобы пользователь был аутентифицирован.

Применение и использование политик авторизации:

Для того чтобы убедиться, что политика авторизации применяется, используются конструкции .RequireAuthorization() и .AllowAnonymous() при настройке маршрутов (endpoints).

* RequireAuthorization() применяется к маршрутам, доступ к которым должен быть ограничен только аутентифицированными пользователями.
* AllowAnonymous() используется для маршрутов, которые должны быть доступны всем пользователям, вне зависимости от их аутентификации.

app.MapPost( "/login",

async (context) =>

{

var formData = await context.Request.ReadFormAsync();

var login = formData["login"].ToString();

var password = formData["password"].ToString();

var contextUser = new MydbContext();

User user = contextUser.Users.FirstOrDefault(x => x.Email == login);

if (user != null && password == user.Password)

{

context.Response.StatusCode = 200;

await context.Response.WriteAsJsonAsync(user);

}

else

context.Response.StatusCode = 401;

})

.AllowAnonymous();

В этом примере гарантируется, что доступ к информации о пользователе имеют только аутентифицированные пользователи, что помогает обеспечить безопасность и конфиденциальность данных.

**Маршрутизация и Обработка запросов**

Статические и динамические маршруты для сущностей User, Vinyl, Order:

* CRUD операции для пользователей и пластинок.
* Аутентификация и авторизация на уровне отдельных маршрутов.
* Использование Entity Framework для взаимодействия с базой данных и выполнения операций над данными (создание, чтение, обновление, удаление).

**Примеры реализации API**

1. API для пользователя (/user/{id:int}):

Выполнение запросов к базе данных для получения данных о пользователях вместе с их заказами и адресами.

2. API для пластинок (/vinyl/{id:int}):

CRUD операции для управления данными о пластинках, включая их треклисты.

3. API для заказов (/orders и /order):

Создание и получение информации о заказах и деталях заказа.

**Использование Middleware**

CORS, аутентификация, авторизация:

Конфигурация и использование middleware для обработки CORS, аутентификации и авторизации.

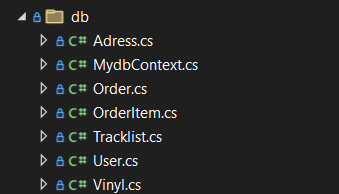
Сущность каждой таблицы из базы данных представляются отдельным файлом с классом.

Рисунок 3.6 - Файлы таблиц в древе проекта

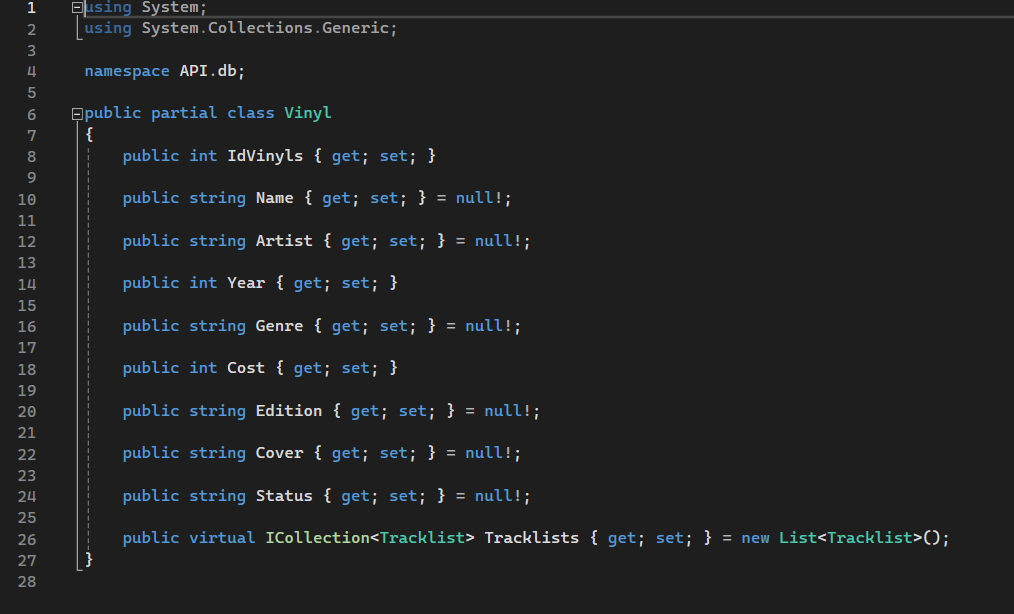
Класс Vinyl представляет собой модель данных для объекта Vinyl в базе данных и является частью контекста Entity Framework в проекте ASP.NET Core. Он определяет структуру таблицы Vinyls, которая сохраняет информацию о различных музыкальных пластинках.

Рисунок 3.7 - Пример класса для таблицы Vinyl

# ЗАКЛЮЧЕНИЕ

В заключении можно отметить, что разработка информационной системы «Онлайн магазин виниловых пластинок» была успешно выполнена. Задачи проекта, поставленные перед началом разработки, были достигнуты, что подтверждается функциональностью созданной системы.

Основные цели проекта, включая создание реляционной базы данных, написание API интерфейса для взаимодействия веб-приложения с базой данных, и наполнение базы данными для демонстрации работы системы были выполнены. Веб-приложение обладает всем необходимым функционалом для простого и удобного интерфейса пользователя, включая отображение виниловых пластинок, поиск по ассортименту, корзину для заказов, оформление заказа, регистрацию и авторизацию пользователей.

В разработке использовались современные программные решения и технологии, такие как ASP.NET Core для backend-разработки и фреймворк Next.js на основе React для frontend-разработки, что обеспечило высокую производительность и удобство использования веб-приложения. Также была реализована система баз данных MySQL с использованием MySQL Workbench, что обеспечило эффективное управление данными и их безопасность.

Тестирование системы показало, что все основные функции работают корректно, и пользователи могут эффективно взаимодействовать с системой. В целом, проект демонстрирует успешное решение задачи создания функционального онлайн-магазина виниловых пластинок с комфортным интерфейсом для пользователя и высоким уровнем выполнения технических требований.

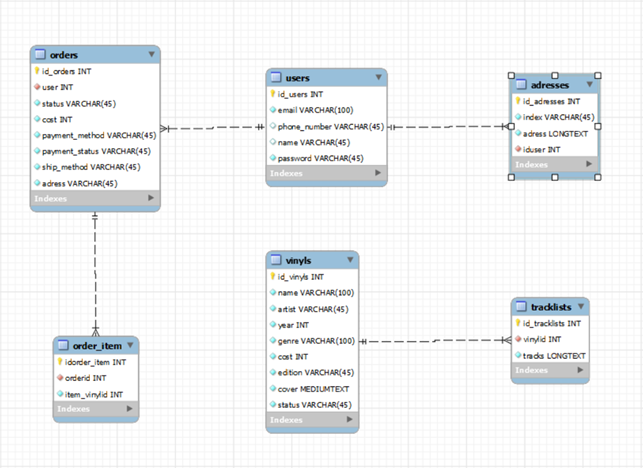
# ЛИТЕРАТУРА И ПРОЧИЕ ИСТОЧНИКИ

# Приложение А

**Тестовый сценарий по стандартам WorldSkills**

|  |  |
| --- | --- |
| **Поле** | **Описание** |
| Название проекта | Онлайн-магазин виниловых пластинок «Ящик винила» |
| Рабочая версия | «Ящик винила» 1.0 |
| Имя тестирующего | Яндулов Егор |
| Дата теста | 2024-05-01 |
| Тестовый пример | UI\_UR\_1 |
| Приоритет тестирования | Высокий |
| Название теста | Регистрация нового пользователя |
| Краткое изложение теста | При прохождении теста должна подтвердиться работоспособность формы регистрации нового пользователя в системе |
| Этапы теста | 1. Открыть интернет-магазин по ссылке (в dev версии localhost:3000)  2. Перейти в раздел «профиль» через верхнее меню приложения  3. Во второй форме (после заголовка «Ещё не зарегистрированы?» заполнить все поля: email, пароль и повтор пароля  4. Нажать кнопку «Регистрация» |
| Тестовые данные | Email: [test@ya.ru](mailto:test@ya.ru)  Пароль: test1test |
| Ожидаемый результат | После нажатия на кнопку «Регистрация» открывается страница созданного профиля |
| Фактический результат | После нажатия на кнопку «Регистрация» открывается страница созданного профиля |
| Предварительные условия | Пользователь не должен быть авторизован в приложении (пустые файлы cookie для приложения) |
| Постусловия | В файлах cookie сохраняются данные пользователя, при каждом переходе в раздел «Профиль» отображаются данные пользователя |
| Статус | Зачет |
| Примечания / комментарии | Данные в полях «Пароль» и «Повтор пароля» должны совпадать, иначе выйдет push об ошибке несовпадения введённых данных |

# Приложение Б

**ER диаграмма БД интернет-магазина**