

**МИНОБРНАУКИ РОССИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Факультет компьютерных наук  
Кафедра технологий обработки и защиты информации

Разработка приложения фриланс-сайта «YouFree»

Курсовой проект

09.03.02 Информационные системы и технологии  
Технологии обработки и защиты информации

Обучающийся \_\_\_\_\_ *А.Н.Воробьёв, 3 курс, д/о*

Руководитель \_\_\_\_\_ *В.С.Тарасов, ст. преподаватель.*

Воронеж 2022

## Содержание

Введение	2
1. Постановка задачи	3
2. Средства реализации	5
3. Описание разработки приложения	8
3.1 Описание структуры данных	8
3.2 Сервер	9
3.3 Клиент	11
4. Описание интерфейса пользователя	13
5. Заключение	23
Список литературы	24
Приложения	25

## **Введение**

В настоящее время интернет становится все более развитой средой для осуществления коммуникаций. В связи с глобальным развитием сети Интернет, в программировании все более резко начала выделяться отдельная его отрасль web-программирование. Сейчас, чтобы привлечь внимание клиентов, покупателей или партнёров, просто необходимо заявить о себе в интернете, путём создания web-сайта. Для этих целей как раз и служит web-сайт, содержащий основную информацию об организации, частном лице, компании, товарах или услугах, прайс-листы, контактные данные. Сайты позволяют хранить, передавать, продавать различные типы информации, не отходя от экрана компьютера. Wide web - глобальная компьютерная сеть, на сегодняшний день содержит миллионы сайтов, на которых размещена всевозможная информация. Люди получают доступ к этой информации посредством использования технологий Internet. Для поиска по интернету используют специальные программы - Web-браузеры, которые существенно облегчают путешествие по бескрайним просторам интернета

### **1. Постановка задачи**

Цель: разработать приложение для осуществления услуг просмотра и выставления заказов

Для достижения поставленной цели необходимо решить следующие задачи:

1. Организовать хранение и модификацию данных о:
  - Пользователе (логин,пароль,емайл,тип пользователя);
  - Заказах(тема,заголовок,описание,стоимость,длительность выполнения).
2. Обеспечить возможность просмотра информации о заказах.
3. Обеспечить защищенность сессии пользователя.

Таким образом, требуется разработать приложение, обеспечивающее:

1. Организацию создания, хранения и модификации информации, включающей в себя:

- Данные о пользователях (данные авторизации, тип пользователя);
- Данные о заказах (название, описание, стоимость).

2. Предоставление информации о:

- Заказах;

3. Авторизацию, основанную на JWT токенах.

## **2. Средства реализации**

### **Microsoft SQL Server**

Microsoft SQL Server - система управления реляционными базами данных, разработанная корпорацией Microsoft. Основным используемый язык запросов – Transact-SQL. Сильными сторонами Microsoft SQL Server традиционно считаются:

- Интеграция структурированных и неструктурированных данных;
- Высокая производительность;
- Безопасность и соответствие требованиям;
- SQL Server упрощает развертывание, передачу и интеграцию больших данных;
- Поддержка постоянной памяти.

### **React**

React – это фреймворк для создания реактивных веб-приложений. Среди аналогичных фреймворков обладает рядом преимуществ:

- Расширение синтаксиса JavaScript – JSX, которое позволяет использовать HTML;

- подобный синтаксис для написания компонентов, транслируется в чистый JavaScript. При этом React-компоненты могут создаваться без использования JSX, но данное расширение делает код значительно более читабельным и простым;

- Однонаправленная передача данных – данные передаются от родительских компонентов к дочерним, при этом данные являются неизменяемыми – поэтому компонент не может напрямую изменять полученные от родителя данные, но для этого могут быть использованы callback-функции;

- Виртуальный DOM – для увеличения производительности React использует кэшированную в памяти структуру для вычисления разницы между предыдущим и текущим состоянием интерфейса, это позволяет эффективно обновлять DOM браузера, избегая перерисовок неизменившихся компонентов. Таким образом с разработчика снимается часть работы над оптимизацией страницы, но при этом сохраняются инструменты, позволяющие принудительно перерисовать компонент при необходимости;

- Методы жизненного цикла позволяют на разных стадиях жизни компонента запускать необходимый код;

- Помимо создания компонентов в виде классов, в React 16.8 была добавлена возможность создавать компоненты в виде функций. В этом случае вместо методов жизненного цикла используются специальные функции – хуки. RTK Query - это мощный инструмент выборки и кэширования данных. Он предназначен для упрощения распространенных случаев загрузки данных в веб-приложение, устраняя необходимость самостоятельного написания логики выборки и кэширования данных. RTK Query является дополнительным дополнением, включенным в пакет Redux Toolkit, и его функциональность построена поверх других API в Redux Toolkit.

## **Язык TypeScript**

Для реализации клиентской стороны был выбран язык TypeScript. Преимущества и особенности выбранного языка:

- Явная типизация и проверка согласования типов на этапе компиляции;
- Обратная совместимость с JS;
- Явная типизация позволяет IDE анализировать код, облегчая его поддержку и уменьшая вероятность сделать ошибку;
- Поддержка таких JS-конструкций, как spread и rest операторов, деструктуризации;
- Поддержка конструкций статически типизированных языков: интерфейсов, перечисляемых типов, обобщенного программирования и т.д.;
- Гибкая настройка компилятора.

## **Язык C#**

Для создания серверной части был выбран язык C#.

C# изначально был придуман компанией Microsoft для собственных целей и служб. Он предусматривает следующие преимущества:

- строгую типизацию;
- функциональность;
- достаточно мощный инструментарий;

Так же использовалась платформа .NET Core. Платформа .NET Core представляет технологию от компании Microsoft, предназначенную для создания различного рода веб-приложений: от небольших веб-сайтов до крупных веб-порталов и веб-сервисов. .NET Core может работать поверх кросс-платформенной среды .NET Core, которая может быть развернута на основных популярных операционных системах: Windows, Mac OS, LINUX. И таким образом, с помощью .NET Core мы можем создавать кросс-платформенные приложения. .NET Core характеризуется расширяемостью. Фреймворк построен из набора относительно независимых компонентов.

Можно либо использовать встроенную реализацию этих компонентов, либо расширить их с помощью механизма наследования, либо вовсе создать и применять свои компоненты со своим функционалом.

## **2. Описание разработки приложения**

### ***3.1 Описание структуры данных***

Для хранения информации о заказах и пользователях была создана база данных

Структура БД (рис. 1) состоит из 3 таблиц:

Users – сущность «направление». Хранит информацию о направлениях – код, полное и краткое название направления обучения;

Order– сущность «группа». Хранит информацию о группах – номер, степень образования, направление;

Categories – сущность «студент». Хранит информацию о студентах – ФИО студента, количество публикаций, группа;

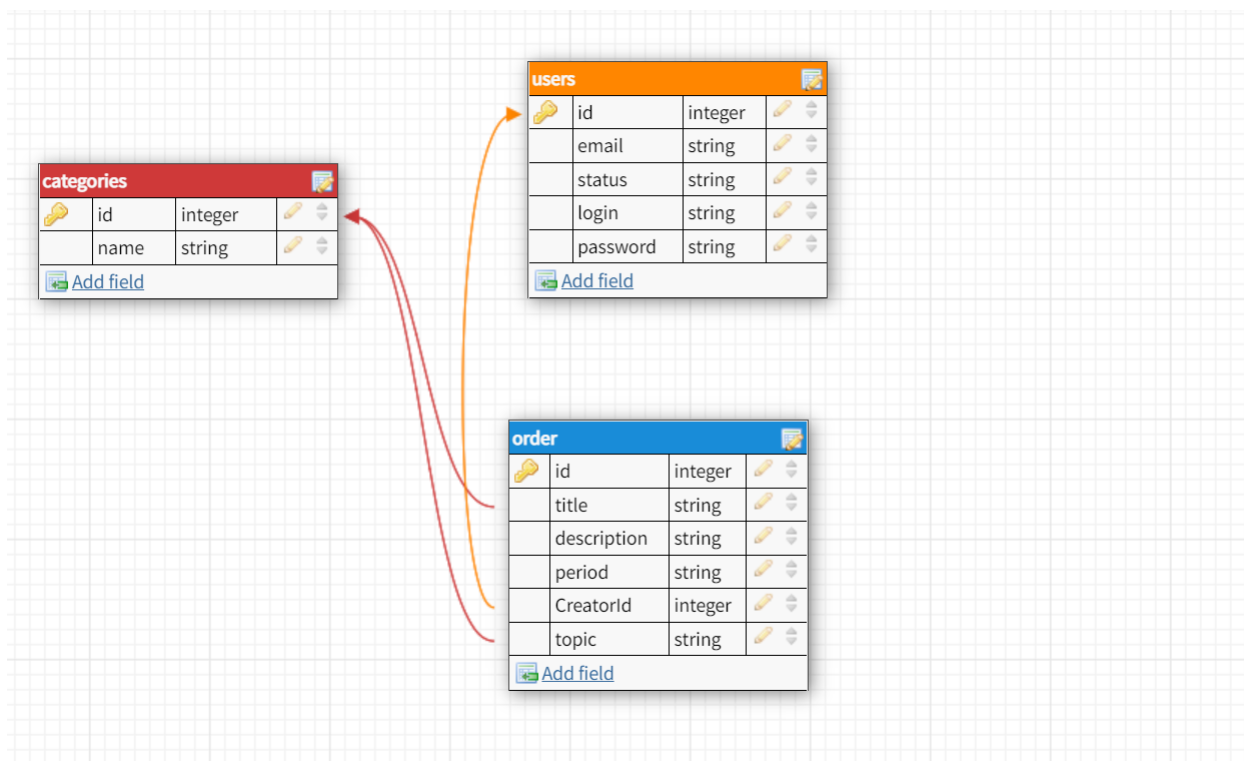


Рис. 1. Схема базы данных

### 3.2 Сервер

На серверной стороне сборка проекта обеспечивается фреймворком ASP.NET Core и платформой .NET.

Сервер вместе с базой данных развернут в Docker.(приложение 1) Docker — это платформа контейнеризации с открытым исходным кодом, с помощью которой можно автоматизировать создание приложений, их доставку и управление. Платформа позволяет быстрее тестировать и выкладывать приложения, запускать на одной машине требуемое количество контейнеров. Благодаря контейнеризации и использованию Docker разработчики больше не задумываются о том, в какой среде будет функционировать их приложение и будут ли в этой в среде необходимые для тестирования опции и зависимости. Достаточно упаковать приложение со всеми зависимостями и процессами в контейнер, чтобы запускать в любых системах: Linux, Windows и MacOS.



Платформа Docker позволила разделить приложения от инфраструктуры. Контейнеры не зависят от базовой инфраструктуры, их можно легко перемещать между облачной и локальной инфраструктурами. Приложение построено по многослойной архитектуре (рис. 2).



*Рис. 2. Многоуровневая архитектура*

Нижний уровень многослойной архитектуры отвечает за взаимодействие с базой данных с помощью SQL-запросов и отображение результатов в C#-объекты. Такое отображение возможно благодаря Entity Framework Core. Entity Framework Core (EF Core) представляет собой объектно-ориентированную, легковесную и расширяемую технологию от компании Microsoft для доступа к данным. EF Core является ORM-инструментом (object-relational mapping - отображения данных на реальные объекты). То есть EF Core позволяет работать базами данных, но представляет собой более высокий уровень абстракции: EF Core позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами.

Для удобства работы с объектами используется AutoMapper. AutoMapper - это объектно-ориентированный редактор. Объектно-объектное сопоставление работает путем преобразования входного объекта одного типа в выходной объект другого типа. Что делает AutoMapper удобным, так это то, что он предоставляет несколько соглашений, позволяющих избавиться от рутинной работы по выяснению того, как сопоставить тип А с типом В. Пока тип В соответствует установленному соглашению AutoMapper, для

сопоставления двух типов требуется почти нулевая конфигурация (приложение 2).

DTO - это объект, который используется для инкапсуляции данных и отправки их из одной подсистемы приложения в другую(приложение 3). DTO чаще всего используются уровнем служб в N-уровневом приложении для передачи данных между собой и уровнем пользовательского интерфейса. Основное преимущество здесь заключается в том, что это уменьшает объем данных, которые необходимо передавать по уровням в приложениях. Они также создают отличные модели в шаблоне MVC.

Совокупность всех моделей и DTO-объектов составляет уровень хранения данных (Data Access Layer).

Уровнем выше располагается слой бизнес-логики. Здесь выполняются все манипуляции с данными и реализуется значительная часть функциональности приложения.

Слой бизнес-логики знает все о нижестоящем слое хранения данных, а значит может использовать его для работы.

Классы, реализующие операции на уровне бизнес-логики называются сервисными классами, и, как минимум, реализуют базовые операции с данными: добавление, получение, обновление, удаление. Также, при необходимости, сервис нагружается дополнительной функциональностью, который можно отнести к бизнес-логике (пример сервиса в приложении 4).

Уровнем выше располагается слой представления данных. Он получает данные от клиента, при необходимости преобразует их в DTO-объект и передает их на дальнейшую обработку сервису слоя бизнес-логики, а также отправляет данные в понятном клиенту формате. В приложении слой представления реализован с помощью контроллеров – классов, которые ответственны за управление входящими запросами и отправкой ответов на них. Каждый метод контроллера, помеченный декоратором соответствующего HTTP-метода ([HttpGet],[HttpPost] и т.д.), становится ответственным за обработку конкретного маршрута с конкретным HTTP-методом. Также контроллер знает о нижестоящем слое, что позволяет ему пользоваться методами сервисов (пример контроллера в приложении 5).

Для защиты сессии пользователя на сервере используется выдача JWT токенов. JWT состоит из трех основных частей: заголовка (header), нагрузки (payload) и подписи (signature). Заголовок и нагрузка формируются отдельно в формате JSON, кодируются в base64, а затем на их основе вычисляется подпись. Закодированные части соединяются друг с другом, и на их основе вычисляется подпись, которая также становится частью токена (рис. 3).

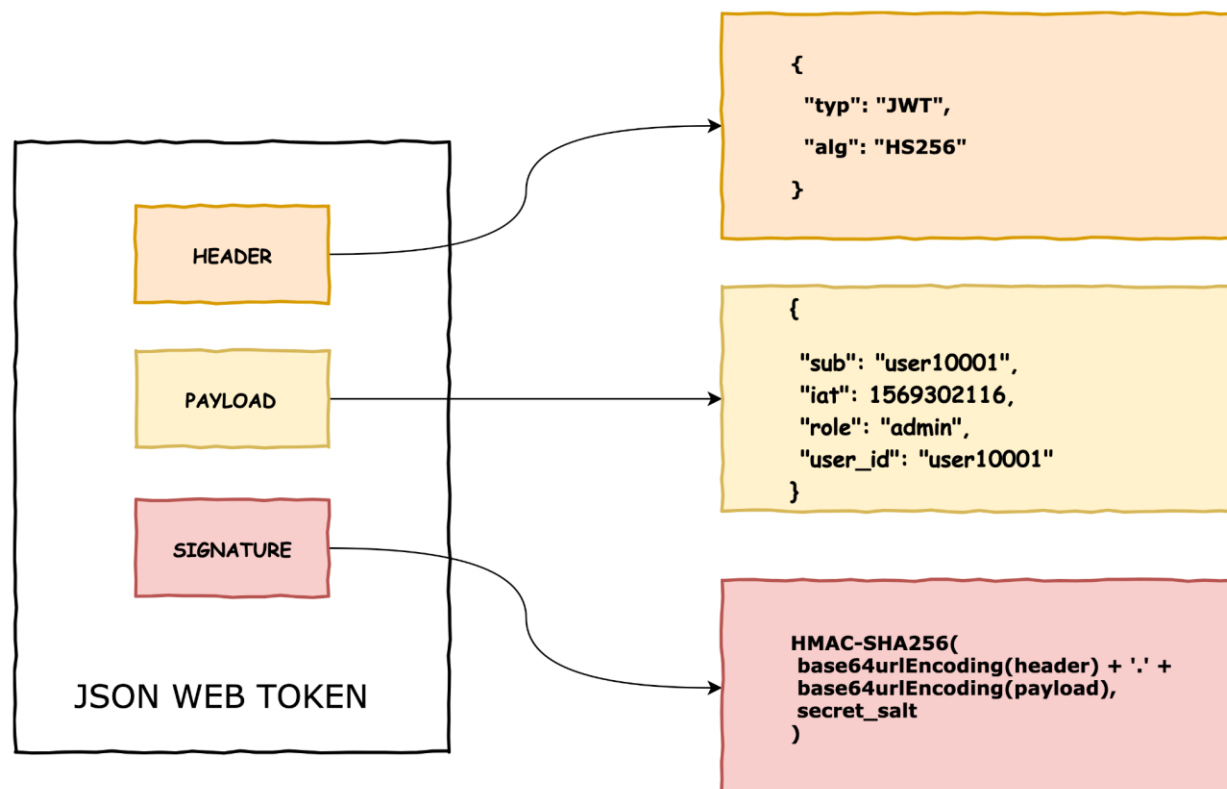


Рис.3.Составляющие JWT токена

Схема аутентификации с использованием JWT предельно проста. Пользователь вводит свои учетные данные в приложении или доверенном сервисе аутентификации. При успешной аутентификации сервис предоставляет пользователю токен, содержащий сведения об этом пользователе.

При последующих обращениях токен передается приложению в запросах от пользователя: заголовках запроса, POST или GET параметрах и т. д. Получив токен, приложение сперва проверяет его подпись. Убедившись, что подпись

действительна, приложение извлекает из части полезной нагрузки сведения о пользователе и на их основе авторизует его.(рис 4.)

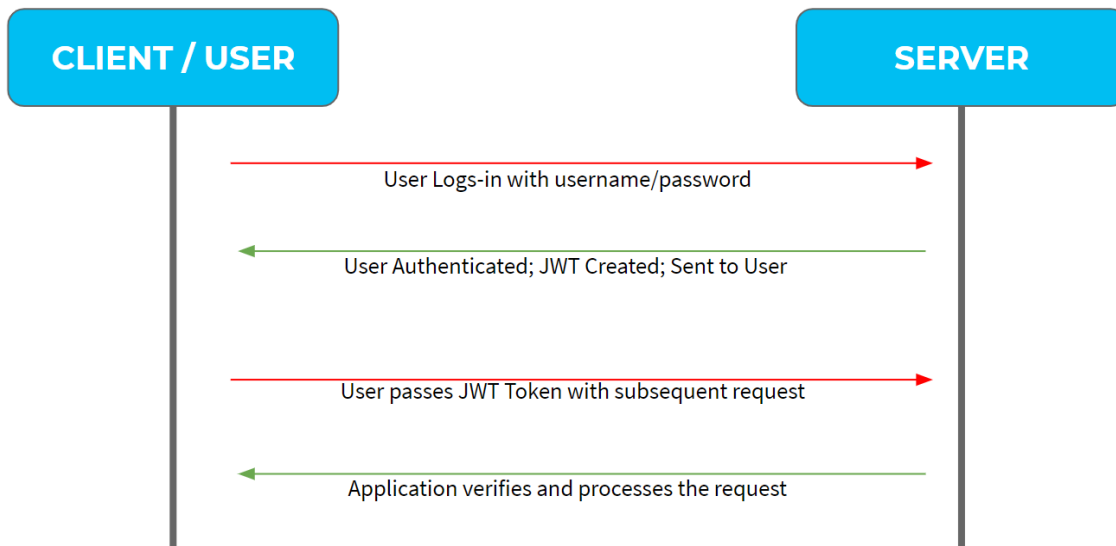


Рис.4.Жизненный цикл JWT токена

### 3.3 Клиент

Веб-клиент написан на React с использованием Redux ToolKit. Отладка такого проекта с использованием расширений React Developer Tools и Redux DevTools становится значительно проще – можно просматривать как и состояние отдельного компонента, так и глобального хранилища в любой момент.

Все страницы разделены на переиспользуемые компоненты (пример React-компонента в приложении 6). На клиентской стороне для управления сборкой проекта используется Webpack, для управления зависимостями и их версиями – пакетный менеджер Npm.

#### 4. Описание интерфейса пользователя

Для рассмотрения интерфейса можно пройти все этапы использования приложения: от просмотра заказов неавторизованным пользователем до создания заказа пользователем с правами создателя. По умолчанию открывается список заказов.



Рис.5.Верхняя часть сайта

Страница отображения заказов со списком категорий(рис. 6)

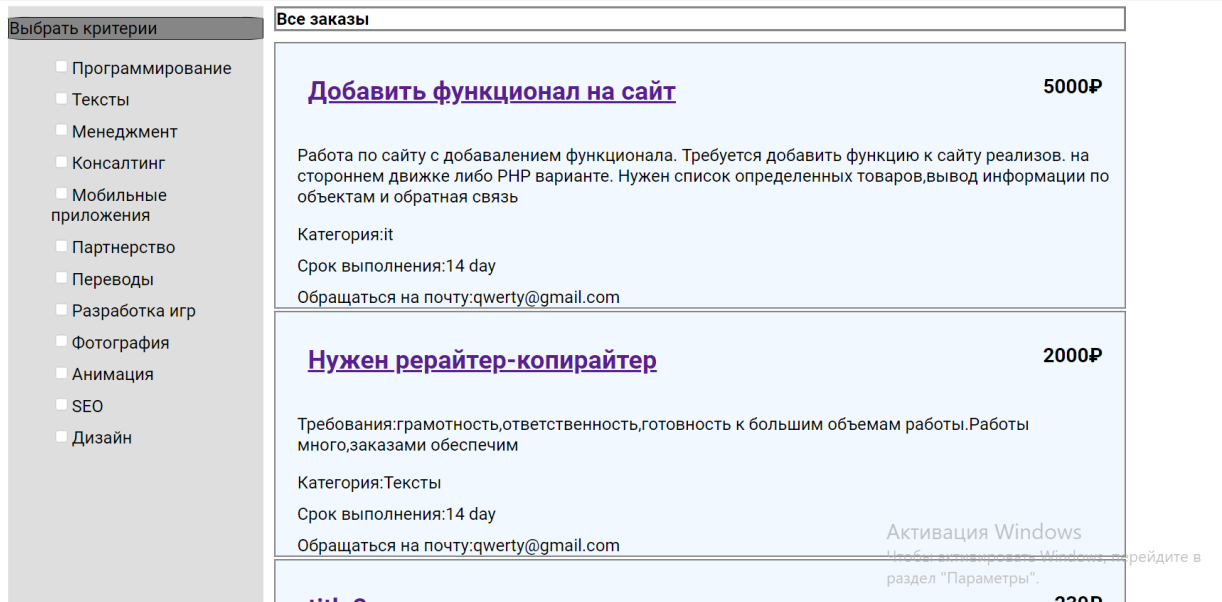


Рис 6.Список заказов

Для решения проблем с пагинацией создан компонент,который отображает доступное количество страниц с элементами (рис.7)



Рис 7.Пагинация

В итоге отобразятся лишь те заказы, которые подходят под выбранные пользователем критерии(рис.9)

Выбрать критерии

☒ Программирование
 ☐ Тексты
 ☐ Менеджмент
 ☐ Консалтинг
 ☐ Мобильные приложения
 ☐ Партнерство
 ☐ Переводы
 ☐ Разработка игр
 ☐ Фотография
 ☐ Анимация
 ☐ SEO
 ☐ Дизайн

Все заказы

Добавить функционал на сайт

5000₽

Работа по сайту с добавалением функционала. Требуется добавить функцию к сайту реализов. на стороннем движке либо PHP варианте. Нужен список определенных товаров,вывод информации по объектам и обратная связь

Категория:Программирование

Срок выполнения:14 day

Обращаться на почту:qwerty@gmail.com

Браузерная онлайн-игра на спортивную тематику

100000₽

Требуется front-end программист, работающий с интерфейсами и обладающий навыками в технологиях: Backbone.js (обязательно), HTML5, CSS3, Javascript, , JSON, JQuery, AJAX, socket.io.Базовая алгоритмическая подготовка, умение писать понятный, самодокументированный, реюзабельный и расширяемый код

Категория:Программирование

Срок выполнения:14 day

Обращаться на почту:qwerty@gmail.com

Рис 9.Список заказов на основе фильтров

Если выбрать заказ,то можно перейти на его страницу(рис. 10)

rewriter

Home All Orders

wol2

.....

Войти

Register

Проект

Добавить функционал на сайт

5000₽

Работа по сайту с добавалением функционала. Требуется добавить функцию к сайту реализов. на стороннем движке либо PHP варианте. Нужен список определенных товаров,вывод информации по объектам и обратная связь

Категория:Программирование

Срок выполнения:14 day

Обращаться на почту:qwerty@gmail.com

*Рис 10.Страница заказа*

Для того чтобы войти в аккаунт пользователю нужно ввести данные от своего аккаунта в форме авторизации(рис. 11)

A horizontal form with a light gray background. It contains a white input field with the text 'wol', a yellow input field with masked characters '.....', and a gray button labeled 'Войти'. Below the input fields is a purple link labeled 'Register'.

Рис.11.Форма авторизации

В случае ввода правильных данных пользователь успешно авторизируется и форма авторизации поменяется на личный кабинет (рис. 12)

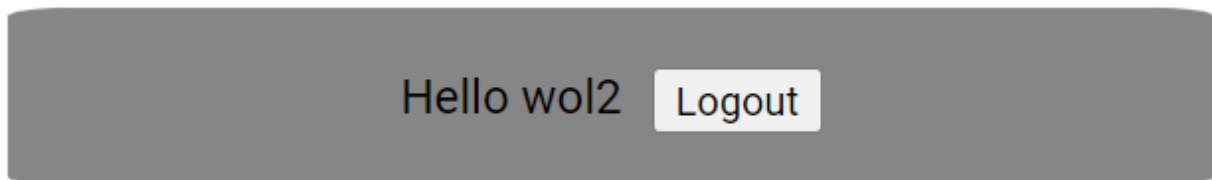
A horizontal bar with a dark gray background. It displays the text 'Hello wol2' in white, followed by a gray button labeled 'Logout'.

Рис.12.Личный кабинет

Если же у пользователя нет аккаунта,то он может нажать на кнопку регистрации аккаунта и тогда он перейдет на страницу регистрации (рис.13)



Рис.13.Страница регистрации

В форме регистрации пользователь должен ввести свои данные и выбрать тип аккаунта для регистрации:исполнитель заказов-это тот кто выполняет заказы,или же создатель заказов,тот,кто может создавать заказы.В случае правильного ввода данных,пользователь успешно зарегистрирует аккаунт и опять попадет в личный кабинет.В случае если пользователь успешно авторизируется как создатель,то в его верхней части сайта появится дополнительная страничка личных заказов-в ней он может создавать,просматривать и удалять личные заказы(рис.14)

Рис.14.Отображение личных заказов пользователя

Страница личных заказов позволяет пользователю информацию о его заказах и возможность удаления и их редактирования(рис.15)

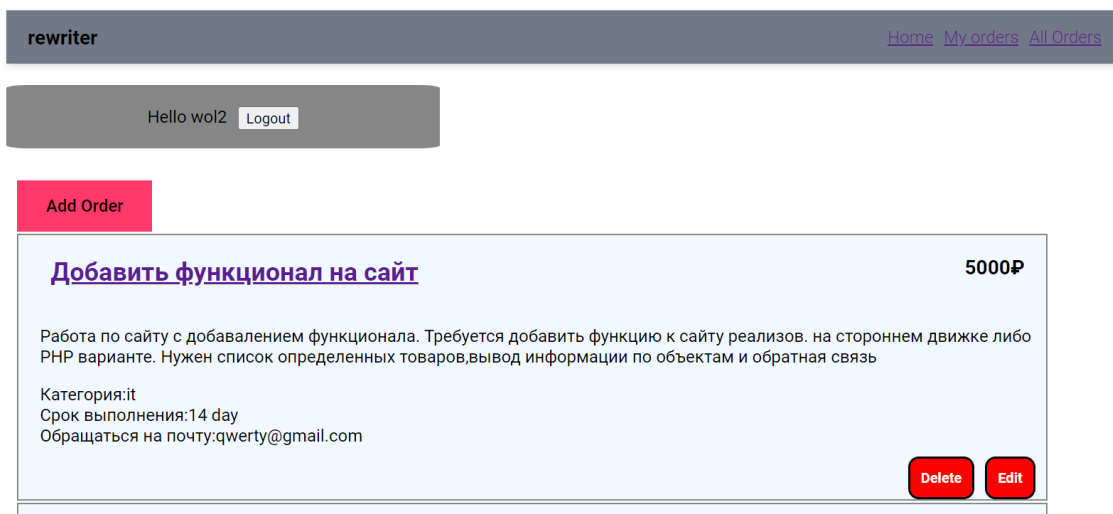


Рис.15.Личные заказы пользователя

Если пользователь хочет отредактировать заказ,он может нажать кнопку Edit и откроется модальная форма редактирования заказа(рис.16)

**Edit** [X]

Title

Браузерная онлайн-игра на спортивную тематику

Description

Требуется front-end программист, работающий с интерфейсами и обладающий навыками в технологиях: Backbone.js (обязательно), HTML5, CSS3, Javascript, ,

Cost

100000

Topic

Программирование

Close Save Changes

Рис.16.Форма редактирования заказа

При нажатии на кнопку удаления выскакивает модальное окно с подтверждением(рис. 17)

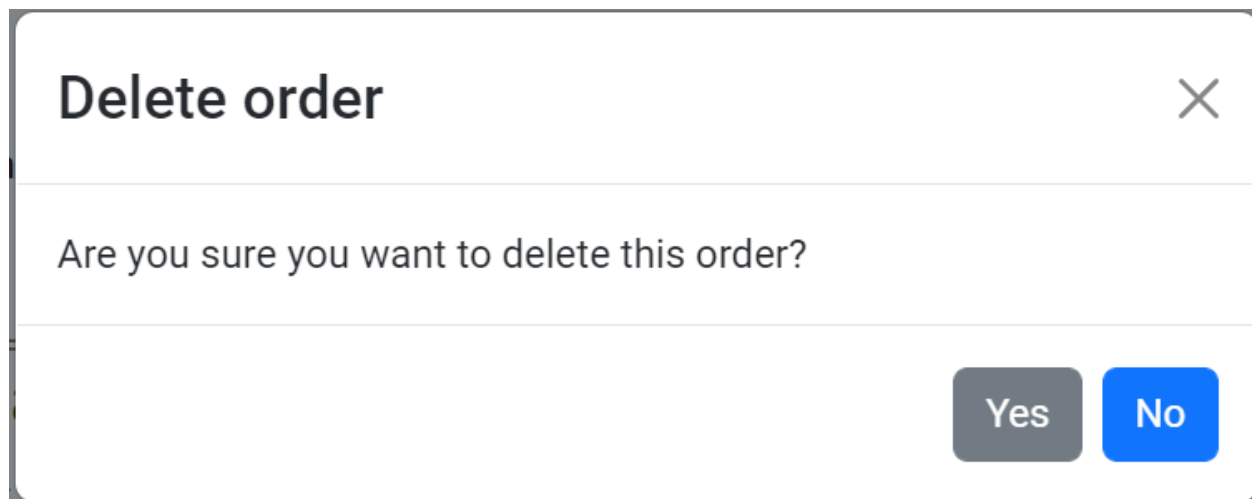


Рис.17.Окно подтверждения удаления заказа

Если пользователь хочет создать заказ,то он должен нажать кнопку Add Order и тогда появится модальное окно добавления заказа(рис. 18)

## Add Order



Title

Description

Cost

Topic

Period

Close

Save Changes

Рис.18.Форма добавления заказа

## **5. Заключение**

В результате проделанной работы было создано приложение для просмотра и создания фриланс заказов. Была выбрана клиент-серверная архитектура, клиент реализован в виде веб-клиента. Для хранения необходимой информации была разработана база данных, реализован механизм ввода, сохранения, просмотра, редактирования и удаления введенной информации: данные о категориях, пользователях и заказах. Внедрена JWT аутентификация и авторизация на основе ролей.

В дальнейшем планируется расширить функциональность приложения добавлением возможности становиться ответственным за выполнение заказа пользователем-исполнителем. Планируется добавление просмотра пользователей и их рейтинга на сайте. Планируется добавление возможности проведения оплаты через сайт

## Список литературы

1. Троелсен, Э. Язык программирования C# 5.0 и платформа .NET 4.5 / Э. Троелсен; Пер. с англ. Ю.Н. Артеменко. – М.: Вильямс, 2016. – 1312 с.
2. Рихтер,Джеффри,(1964).CLR via C# /Джеффри Рихтер ; 4-е изд.- Москва [и др.] : Питер, 2019. – 895 с.
3. Грабер,Мартин.Введение в SQL / Мартин Грабер; [Пер. В. А. Ястребов]. – М. : Лори, 1996. – XVII, 375.
4. Розенталс,Натан.Изучаем TypeScript 3:создавайте промышленные веб-приложения корпоративного класса с использованием TypeScript 3 и современных фреймворков / Натан Розенталс. – Москва : ДМК Пресс, 2019. – 623 с.
5. Чиннатамби,Кирупа.Изучаем React : практическое руководство по созданию веб-приложений при помощи React и Redux / Кируп Чиннатамби ; пер. с англ. М. А. Райтмана. - 2-е изд. - Москва : Эксмо, 2019. - 365 с.
6. Учебник. Контейнеризация приложения .NET – Режим доступа: свободный.<https://learn.microsoft.com/ru-ru/dotnet/core/docker/build-container?tabs=windows> – (Дата обращения: 12.08.2022).
7. Решаем проблемы REST с помощью Redux Toolkit Query. – Режим доступа:свободный. <https://habr.com/ru/company/netcracker/blog/646163/>. – (Дата обращения: 15.08.2022).
8. Building an ASP.NET Web API with ASP.NET Core. – Режим доступа: свободный. <https://www.toptal.com/asp-dot-net/asp-net-web-api-tutorial> – (Дата обращения 20.08.2022).
9. JWT validation and authorization in ASP.NET Core . – Режим доступа: свободный.<https://devblogs.microsoft.com/dotnet/jwt-validation-and-authorization-in-asp-net-core/>. – (Дата обращения 29.08.2022).
- 10.Redux Toolkit как средство эффективной Redux-разработки. – Режим доступа: свободный. <https://habr.com/ru/company/inobitec/blog/481288/>. – (Дата обращения: 27.08.2022)





## Приложения

### Приложение 1

#### *Пример Docker файла для контейнеризации сервера*

```
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src
COPY ["rewriter/rewriter.csproj", "rewriter/"]
RUN dotnet restore "rewriter/rewriter.csproj"
COPY . .
WORKDIR "/src/rewriter"
RUN dotnet build "rewriter.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "rewriter.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "rewriter.dll"]

version: '3.4'

services:
  rewriter_api:
    image: ${DOCKER_REGISTRY-}rewriterapi
    build:
      context: .
      dockerfile: rewriter/Dockerfile
    depends_on:
      - rewriter_sqlserver
    env_file:
      - env.api

  rewriter_sqlserver:
```

```
image: mcr.microsoft.com/mssql/server:2019-latest
env_file:
  - env.api
ports:
  - "1433:1433"
```

*Пример описания работы AutoMapper*

```
using AutoMapper;
using Db.Entities;
using FluentValidation;

namespace rewriter.OrderService.Models
{
    public class AddOrderModel
    {
        public string title { get; set; }
        public string description { get; set; }
        public decimal cost { get; set; }
        public string period { get; set; }
        public string topic { get; set; }
        public int CreatorId { get; set; }
    }
    public class AddBookModelValidator : AbstractValidator<AddOrderModel>
    {
        public AddBookModelValidator()
        {
            RuleFor(x => x.title)
                .NotEmpty().WithMessage("Empty Title")
                .MaxLength(200).WithMessage("Not allowed size of title");
            RuleFor(x => x.description)
                .MaxLength(300).WithMessage("Not allowed size of description");
            RuleFor(x => x.cost)
                .NotEmpty().WithMessage("Set the cost");
            RuleFor(x => x.period)
                .NotEmpty().WithMessage("Set the period");
        }
    }
    public class AddOrderModelProfile : Profile {
        public AddOrderModelProfile()
        {
            CreateMap<AddOrderModel, Order>();
        }
    }
}
```

*Пример DTO-объекта*

```
using AutoMapper;
using Db.Entities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace rewriter.Services.Models
{
    public class UserModel
    {
        public int id { get; set; }
        public string email { get; set; }
        public string status { get; set; }
        public string password { get; set; }
        public string login { get; set; }
    }
    public class UserModelProfile : Profile
    {
        public UserModelProfile()
        {
            CreateMap<User, UserModel>();
        }
    }
}
```

*Пример сервиса*

```

    using AutoMapper;
using Db.Context.Context;
using Db.Entities;
using FluentValidation;
using Microsoft.EntityFrameworkCore;
using rewriter.OrderService.Models;
using rewriter.Services.Models;
using rewriter.Shared.Common.Enums;
using rewriter.Shared.Common.Exceptions;
using rewriter.Shared.Common.Validator;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace rewriter.OrderService
{
    public class OrderService : IOrderService
    {
        private readonly IDbContextFactory<MainDbContext> contextFactory;
        private readonly IMapper mapper;
        private readonly IModelValidator<AddOrderModel> addordermodelValidator;
        private readonly IModelValidator<UpdateOrderModel> updateordermodelValidator;

        public OrderService(IDbContextFactory<MainDbContext> contextFactory, IMapper mapper, IModelValidator<AddOrderModel> addordermodelValidator, IModelValidator<UpdateOrderModel> updateordermodelValidator)
        {
            this.contextFactory = contextFactory;
            this.mapper = mapper;
            this.addordermodelValidator = addordermodelValidator;
            this.updateordermodelValidator = updateordermodelValidator;
        }

        public async Task<OrderModel> AddOrder(AddOrderModel model)
        {
            addordermodelValidator.Check(model);
            using var context = await contextFactory.CreateDbContextAsync();
            var order = mapper.Map<Order>(model);
            order.status = StatusOrderEnum.InProgress;
            await context.Orders.AddAsync(order);
            context.SaveChanges();
            return mapper.Map<OrderModel>(order);
        }

        public async Task DeleteOrder(int id)
        {
            using var context = await contextFactory.CreateDbContextAsync();

```

```

        var order = await context.Orders.FirstOrDefaultAsync(x=>x.id.Equals(id))
        ?? throw new ProcessException($"The order id:{id} was not found");
        context.Remove(order);
        context.SaveChanges();
    }

    public async Task<OrderModel> GetOrder(int id)
    {
        using var context = await contextFactory.CreateDbContextAsync();
        var order = await context.Orders.FirstOrDefaultAsync(x=>x.id.Equals(id));
        var data=mapper.Map<OrderModel>(order);
        return data;
    }

    public async Task<OrdersResponse> GetOrderForPage(string page,string[] topics,int offset)
    {
        int pageInt = int.Parse(page);
        using var context = await contextFactory.CreateDbContextAsync();
        var splitTopics = (topics.Length>0 && topics[0]!=null)?topics[0].Split(','):Array.Empty<string>();
        var orders = context.Orders.AsEnumerable();
        if (splitTopics.Length>0)
        {
            orders = orders.Where(x => splitTopics.Contains(x.topic));
        }
        OrdersResponse response = new OrdersResponse();
        response.TotalCountItems = orders.Count();
        response.ItemsPerPage = offset;
        var filterOrders = orders.Where((x, i) => ((i >= offset * (pageInt - 1) && i < offset * pageInt)));
        //var data = (await orders.ToListAsync()).Select(order => mapper.Map<OrderModel[]>(order));
        response.orders=mapper.Map<IEnumerable<OrderResponseModel>>(filterOrders);

        return response;
    }

    public async Task<IEnumerable<OrderModel>> GetOrders()
    {
        using var context = await contextFactory.CreateDbContextAsync();
        var orders = context.Orders.AsQueryable();
        var data = (await orders.ToListAsync()).Select(order => mapper.Map<OrderModel>(order));
        return data;
    }

    public async Task UpdateOrder(int id, UpdateOrderModel model)
    {
        updateordermodelValidator.Check(model);
        using var context = await contextFactory.CreateDbContextAsync();
        var order = await context.Orders.FirstOrDefaultAsync(x => x.id.Equals(id))
        ?? throw new ProcessException($"The order id:{id} was not found");
        order = mapper.Map(model, order);
        context.Orders.Update(order);
    }

```

```

        context.SaveChanges();
    }

    public async Task<OrdersResponse> GetProductsByUserId(int page,int userId,int offset)
    {
        using var context = await contextFactory.CreateDbContextAsync();
        var orders = context.Orders.AsEnumerable();
        var userOrders = orders.Where(x => x.CreatorId == userId);
        OrdersResponse response = new OrdersResponse();
        response.TotalCountItems = userOrders.Count();
        response.ItemsPerPage = offset;
        var paginationOrders=userOrders.Where((x, i) => ((i >= offset * (page - 1) && i < offset * page)));
        response.orders = mapper.Map<IEnumerable<OrderResponseModel>>(paginationOrders);
        return response;
    }
}

```

*Пример контроллера*

```

using AutoMapper;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using rewriter.OrderService;
using rewriter.OrderService.Models;
using rewriter.Services.Models;
using System.Web.Http.Cors;

namespace rewriter.Controllers
{
    [ApiController]
    [Route("api/orders")]
    public class OrdersController : ControllerBase
    {
        //private readonly ILogger<OrderController> logger;
        private readonly IOrderService orderService;
        private readonly IMapper mapper;
        public OrdersController(IOrderService orderService, IMapper mapper)
        {
            this.orderService = orderService;
            this.mapper = mapper;
        }
        [HttpPost("")]
        public async Task<OrderModel> AddOrder(AddOrderModel model)
        {
            var order = await orderService.AddOrder(model);
            return order;
        }
        [Authorize]
        [Route("all")]
        public async Task<IEnumerable<OrderModel>> GetAllOrders()
        {
            var orderds=await orderService.GetOrders();
            return orderds;
        }
        [Authorize(Roles = "Creator")]
        [Route("string")]
        public async Task<string> GetAllOrdersstring()
        {
            var orderds = await orderService.GetOrders();
            return "working";
        }
        [HttpGet]
        public async Task<OrdersResponse> GetOrderForPage([FromQuery] string page,[FromQuery(Name = "topics")]
string[] topics, int offset=7)
        {
            var orders = await orderService.GetOrderForPage(page,topics,offset);

```



```

        return orders;
    }
    [HttpGet]
    [Authorize(Roles = "Creator")]
    [Route("userId")]
    public async Task<OrdersResponse> GetProductsByUserId([FromQuery] int page,[FromQuery] int userId,int off-
set=7)
    {
        var orders = await orderService.GetProductsByUserId(page, userId,offset);
        return orders;
    }

    [HttpDelete]
    public async Task DeleteOrder([FromQuery] int id)
    {
        await orderService.DeleteOrder(id);
    }

    [HttpPut]
    public async Task UpdateOrder([FromQuery] int id,UpdateOrderModel model)
    {
        await orderService.UpdateOrder(id,model);
    }
}

```

## Пример React-компонента

```

import React, { useEffect, useState } from "react";
import { useGetProductsQuery, useGetStringQuery } from
"../store/order/order.api";
import OrderItem from "../components/OrderItem"
import './AllOrders.css'
import isNaN from "lodash/isNaN";
import { IOrder, orderType } from "../store/order/order.type";
import { useActions } from "../hooks/useActions";
import { useTypedSelector } from "../hooks/useTypedSelector";
import { Pagination } from "../components/Pagination";
import { useLocation, useNavigate } from "react-router-dom";
import { Categories } from "../components/Categories";

const initState: IOrder[] = []
export function AllOrders() {
  const initialState: string[] = []
  const [categories, setCategories] = useState(initialState)
  const { addOrders } = useActions();
  const navigate = useNavigate();
  const location = useLocation();
  const pageNumber = !isNaN(Number(location.search.split("=")[1]))
    ? Number(location.search.split("=")[1])
    : 1;
  const [ordersData, SetOrders] = useState(initstate)
  const [pagesCount, setPagesCount] = useState(0);

```

```

    const [currentPage, setCurrentPage] = useState(pageNumber);

    const { data, isLoading, error } = useGetProductsQuery({page:currentPage,topics:categories})

    const handlePageChange = ({ selected }: { selected: number }) => {
        setCurrentPage(selected + 1);
        navigate(`/allOrders/?page=${selected + 1}`)
    };

    const categoriesHandler = (event: any) => {
        if (!categories.includes(event.target.name))
            setCategories([...categories, event.target.name])
        else {
            setCategories((prev) => {
                return prev.filter(e => e !== `${event.target.name}`)
            })
        }
        setCurrentPage(1)
        navigate("/allOrders/?page=1")
    }

    useEffect(() => {
        if (!isLoading) {
            var total =0;
            var perPage=0;
            if(data?.totalCountItems)
                total+=data.totalCountItems
            if(data?.itemsPerPage)

```

```

        perPage+=data.itemsPerPage

        const pagesQuantity = Math.max(Math.ceil(total /
perPage), 1)

        if (data) {
            addOrders(data.orders as IOrder[])
        }

        const fetchPr = async () => {
            await SetOrders(data?.orders as IOrder[])
        }

        fetchPr();

        if (pagesQuantity) {
            setPagesCount(pagesQuantity);
        }
    }

}, [currentPage, categories, isLoading, data])

return (
    <div className="all-container">
        <div className="left-container">

            <div className="filter-container">

                <Categories categoriesHandler={categorie-
sHandler} submitHandler={submitHadhler} />

            </div>

        </div>

        <div className="orders-container">
            <h4 className="orders-header">Все заказы</h4>
            {isLoading ? (

```

```

        'Loading...'
    ) : error ? (
        <div>Error</div>
    ) : (
        <div className='flex flex-wrap justify-be-
tween'>
            {data?.orders && data.orders.map(order
=> (
                <OrderItem key={order.id} order={or-
der} />
            ))}
        </div>
    )}
    <div className="paginations-allorders"><Pagina-
tion
        pagesCount={pagesCount}
        onChange={handlePageChange} /></div>
    </div>
</div>
)
}

```