МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных

технологий»

Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

WEB-приложение «Авто центр VW»

Выполнил студент Болвако Кирилл Сергеевич

(Ф.И.О.)

Руководитель проекта асс. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты асс. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер асс. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2021

Содержание

[Введение 3](#_Toc90275077)

[1 Постановка задачи 4](#_Toc90275078)

[1.1 Актуальность задачи 4](#_Toc90275079)

[1.2 Обзор аналогов 4](#_Toc90275080)

[2 Проектирование программного средства 7](#_Toc90275081)

[2.1 Диаграмма вариантов использования 7](#_Toc90275082)

[2.2 Описание используемых технологий 7](#_Toc90275083)

[2.3 Проектирование базы данных 8](#_Toc90275084)

[2.4 Проектирование серверной части приложения 9](#_Toc90275085)

[2.5 Проектирование клиентской части приложения 10](#_Toc90275086)

[2.6 Docker 11](#_Toc90275087)

[3 Разработка программного средства 13](#_Toc90275088)

[3.1 Реализация серверной части приложения 13](#_Toc90275089)

[3.2 Реализация клиентской части приложения 14](#_Toc90275090)

[3.3 Развёртывание приложения в docker 16](#_Toc90275091)

[4 Тестирование программного средства 19](#_Toc90275092)

[5 Руководство пользователя 23](#_Toc90275093)

[Заключение 27](#_Toc90275094)

[Список используемых источников 28](#_Toc90275095)

[ПРИЛОЖЕНИЕ А 29](#_Toc90275096)

[ПРИЛОЖЕНИЕ Б 30](#_Toc90275097)

[ПРИЛОЖЕНИЕ В 31](#_Toc90275098)

[ПРИЛОЖЕНИЕ Г 32](#_Toc90275099)

[ПРИЛОЖЕНИЕ Д 34](#_Toc90275100)

Введение

В наше время невозможно жить без автомобиля. Многие, покупают автомобили, чтобы быстро и комфортно добираться до работы. Также автомобили покупают для работы например в такси. Автомобиль в наше время является предметом, который экономит наше время.

Целью курсового проекта является разработка WEB-приложения, которое сможет предоставлять возможность пользователям для управления своими автомобилями, продажей и просмотра новостей. При использовании приложения может потребоваться смена роли, для этого предусмотрен администратор, который может изменять роли пользователей в системе.

Пояснительная записка курсового проекта состоит из 30 страниц, 30 рисунков, 3 приложений, 7 источников литературы.

В первом разделе рассматриваются основные технологии, которые использовались в разработке данного приложения, а также его прототипы и актуальность задачи.

Во втором разделе описана архитектура курсового проекта.

В третьем разделе предоставлена информация о разработанных объектах базы данных.

Четвертый раздел содержит руководство пользователя для разработанного клиентского приложения.

В пятом разделе представлены результаты тестирования приложения.

В заключении описывается результат курсового проектирования и задачи, которые были решены в ходе разработки приложения.

1. **Постановка задачи**

Главной задачей курсового проекта является разработка веб-приложения, которое предоставляет пользователям возможность покупать автомобили, просматривать новости и продавать автомобили. Также есть возможность просматривать свои автомобили и редактировать информацию о автомобилях. В данном курсовом проекте приложение должно удовлетворять следующим требованиям:

* регистрация и авторизация пользователей;
* обеспечивать управление новостями, акциями о автомобилях;
* обеспечивать регистрацию пользователей и преподавателей;
* обеспечивать подачу заявки на тест-драйв;
* обеспечивать поиск автомобиля по различным полям;
* обеспечивать просмотра детальной информации о автомобиле;
* обеспечивать создание комплектации автомобиля;
* обеспечивать просмотр новостей;
* обеспечивать сдачу автомобиля в trade-in.

**1.1 Актуальность задачи**

Многие люди в мире хотят экономить своё время и поэтому покупают автомобиль, чтобы быстрее добираться до места. Автомобили являются затратными ресурсами в плане денег и поэтому они не всем доступны. Некоторым людям не нравится общественный транспорт из-за того, что утром слишком много людей и им не комфортно и это является второй причиной покупки автомобиля. Также если у вас есть автомобиль, вам не нужно думать, как съездить отдохнуть и не надо подстраиваться под расписание автобусов.

1. **Обзор аналогов**

В качестве прототипа был выбран сайт «Официальный дилер VW», предоставляющий большой выбор новых автомобилей и старых, есть функционал по созданию собственных конфигураций автомобиля, есть возможность записаться на тест-драйв. Начальная страница представлена на рисунке 1.1.

|  |
| --- |
|  |

Рисунок 1.1 – Стартовая страница сайта «Официальный дилер VW»

На странице с записью на тест-драйв у нас появляется пошаговая форма с различными пунктами выбора типа тест-драйва. Страница с формой для записи на тест-драйв представлена на рисунке 1.2.

|  |
| --- |
|  |

Рисунок 1.2 – Страница, с формой для записи на тест-драйв, сайта «Официальный дилер VW»

В качестве второго прототипа был выбран сайт «Официальный дилер Audi». Отличительной особенностью этого сайта является интерфейс он понятен при первом входе на сайт и это является плюсом. Так же этот сайт содержит запись на сервисное обслуживание. В большинстве случаев сайты на подобную тему имеют одинаковый функционал. Начальная страница представлена на рисунке 1.3.

|  |
| --- |
|  |

Рисунок 1.3 – Стартовая страница сайта «Официальный дилер Audi»

Данный сайт также содержит страницу с продажей автомобилей с пробегом, что позволит пользователям автомобилей купить у данного дилера не только новые автомобили. Страница с автомобилями с пробегом представлена на странице 1.4.

|  |
| --- |
|  |

Рисунок 1.4 – Список автомобилей с пробегом «Официальный дилер Audi»

Данный сайт так же позволяет посмотреть детальное описание каждого автомобиля, который у их есть в наличии. Также есть уникальная функция, как 3d тур, после просмотра сайтов на данную тему курсового проекта, данная функция была выделена, как уникальная.

# **Проектирование программного средства**

Главной задачей данного курсового проекта является создание веб-приложения, которое предоставляет пользователям возможность совершать покупку автомобиля. Исходя из изученных, существующих аналогов, были поставлены задачи по созданию программных компонентов для работы администратора, пользователя и сотрудника, а также по созданию интуитивно понятного интерфейса.

**2.1 Диаграмма вариантов использования**

Диаграмма вариантов использования является исходным концептуальным представлением системы в процессе ее проектирования и разработки. Данная диаграмма состоит из актеров, вариантов использования и отношений между ними.

Суть данной диаграммы состоит в следующем: проектируемая система предоставляется в виде множества актеров, взаимодействующих с системой с помощью, так называемых вариантов использования. При этом актером называется любой объект, субъект или система, взаимодействующая с моделируемой системой извне. В свою очередь вариант использования — это спецификация сервисов или функций, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемых системой при взаимодействии с актером.

Диаграмму вариантов использования проекта можно увидеть в приложении А. На данной диаграмме есть 3 актера: пользователь, сотрудник и администратор.

Администратор после авторизации имеет возможность управлять пользователями, сотрудниками, автомобилями, новостями, акциями и комплектациями.

Сотрудник после авторизации может управлять автомобилями, новостями, акциями и комплектациями.

Пользователь может управлять своими автомобилями, покупать автомобили, записываться на тест-драйв, просматривать новости и комплектации.

## 2.2 Описание используемых технологий

ASP.NET Core — это платформа представляет технологию от компании Microsoft, предназначенную для создания различного рода веб-приложений. Данная платформа является полностью открытой и находится в открытом доступе. ASP.NET Core характеризуется расширяемостью. Фреймворк построен из набора относительно независимых компонентов. И мы можем либо использовать встроенную реализацию этих компонентов, либо расширить их с помощью механизма наследования, либо вовсе создать и применять свои компоненты со своим функционалом.

EF Core — представляет собой объектно-ориентированную, легковесную и расширяемую технологию от компании Microsoft для доступа к данным. Он представляет универсальный API для работы с данным. И если, к примеру, мы решим сменить целевую СУБД, то основное изменение в проекте будут касаться прежде всего конфигурации и настройки подключения к соответствующим провайдерам.

HTML – это язык разметки гипертекста. Данный язык применяется для создания веб-страниц. Он интерпретируется (обрабатывается) браузером и отображается в виде документа в удобной для человека форме.

CSS – это язык описания внешнего вида документа, написанного с использованием языка разметки.

React — это библиотека JavaScript, которая используется для создания пользовательского интерфейса. React первоначально предназначался для веба, для создания веб-сайтов. Отличительной чертой библиотеки является концепция на компонентах – мы можем создать отдельные компоненты и затем их легко переносить из проекта в проект.

## 2.3 Проектирование базы данных

В курсовом проекте была использована реляционная СУБД MSSql и не реляционная MongoDB.

При проектировании реляционной базы данных необходимо исследовать предметную область с целью определения объектов, нормализовать данные и установить связи между ними.

MSSql является одной из наиболее популярных СУБД в мире. Данная СУБД подходит для самых разных проектов. Для организации баз данных используется реляционная модель. Эта модель баз данных, которая предполагает хранение данных в виде таблиц, каждая из которых состоит из строк и столбцов. Каждая строка хранит отдельный объект, а в столбцах размещаются атрибуты этого объекта.

MongoDB является документоориентированным системой управления базами данных, не требующая описания схемы таблиц. Считается одним из классических примеров NoSQL- систем, использует JSON- подобные документы и схему база данных.

Для базы данных данного приложения было разработано 10 таблиц. В приложении Б представлена структура связей внешних ключей и первичных ключей. Если реляционные базы данных хранят строки, то MongoDB хранит документы. В отличие от строк документы могут иметь сложную по структуре информацию. Документ можно представить как хранилище ключей и значений. Структура MongoDB представлена в приложении.

Таблица «Roles», включает поля:

- id – первичный ключ, идентификатор роли;

- RoleName – название роли.

Таблица «Users», включает поля:

- id – первичный ключ, идентификатор пользователя;

- FirstName – имя пользователя;

- LastName – фамилия о пользователе;

- Surname – отчество пользователя;

- Status – статус аккаунта пользователя;

- Password – пароль пользователя;

- Email – почта пользователя;

- PhoneNumber – номер телефона;

- RoleId – внешний ключ, идентификатор роли.

Таблица «News», включает поля:

- id – первичный ключ, идентификатор новости;

- Title – название новости;

- Description – описание новости;

- EmployeeId – внешний ключ, идентификатор сотрудника.

Таблица «Employees», включает поля:

- id – первичный ключ, идентификатор сотрудника;

- StartWorkDate – дату, когда взяли на работу;

- IsActive – статус аккаунта сотрудника;

- Address – адрес сотрудника;

- UserId – внешний ключ, идентификатор пользователя.

Таблица «Imgs», включает поля:

- id – первичный ключ, идентификатор изображения;

- Urk – ссылка на изображение;

- NewId – вторичный ключ, идентификатор новости.

Таблица «ClientsCars», включает поля:

- id – первичный ключ, идентификатор клиентских машин;

- RegisterNumber – регистрационный номер автомобиля пользователя;

- UserId – вторичный ключ, идентификатор пользователя;

- CarId – вторичный ключ, идентификатор автомобиля.

Таблица «TestDrives», включает поля:

- id – первичный ключ, идентификатор тест-драйва;

- Date – дата, когда будет тест-драйв;

- Time – время в какое будет тест-драйва;

- CarId – внешний ключ, идентификатор автомобиля;

- IsActive – статус тест-драйва.

Таблица «Orders», включает поля:

- id – первичный ключ, идентификатор покупки автомобиля;

- TotalCost – общая стоимость автомобиля;

- DateOfBuyCar – дата покупки автомобиля;

- CarId – внешний ключ, идентификатор автомобиля.

Таблица «Cars», включает поля:

- id – первичный ключ, идентификатор автомобиля;

- NameCarEquipment – наименование комплектации автомобиля;

- VIN – вин код автомобиля;

- Cost – стоимость автомобиля;

- CarMileage – пробег автомобиля;

- DateOfRealeseCar – дата выпуска автомобиля;

- IsActive – состояние автомобиля;

- ActionCarId – вторичный ключ, идентификатор акции.

Таблица «ActionCars», включает поля:

- id – первичный ключ, идентификатор акции;

- SharePercentage – хранит целое число и представляет процент акции.

## 2.4 Проектирование серверной части приложения

Важным этапом проектирования любого приложения, является этап проектирования структуры проекта. Структура серверной части приложения представлена на рисунке 2.1.

|  |  |
| --- | --- |
| |  | | --- | |  | |

Рисунок 2.1 – Структура серверной части приложения

В билиотеке «MicroServiceApp.InfrastructureLayer» находится конфигурация аутентификации, конфигурация мапера, конфигурация консула, кастомные атрибуты валидации, dto требуется для сопоставление с моделями, функция хэширования и верификации пароля и модели. В библиотеке «MicroServiceApp.HttpClientLayer» находится HttpClient для генерации запросов на маршрутизатор. В web-api приложении «MicroServiceApp.ApiGateway» содержится конфигурация маршрутизатора и выполняет роль маршрутизации запросов. В web –api приложении «MicroServiceApp.ServiceAuth» находится логика аутентификации пользователя и генерации JWT-токена. В «MicroServiceApp.ServiceCar» лежит логика работы с автомобилями. В web-api приложении «MicroServiceApp.ServiceNew» лежит логика для работы с новостями. В «MicroServiceApp.ServiceRepository» находится контекст базы данных и предназначен для работы с базой данных. В «MicroServiceApp.ServiceUser» лежит логика для работы с пользователями.

Так же на стороне сервера есть файл «consul.exe» он запускается при помощи командной строки и он является service discovery, который опрашивает другие сервисы, кроме маршрутизатора, он проверяет доступен ли сервис. Когда приходит запрос на маршрутизатор , маршрутизатор спрашивает у service discovery доступен ли сервис, на который пришел запрос, если доступен, то маршрутизирует запрос.

## 2.5 Проектирование клиентской части приложения

Структура клиентской части приложения показана на рисунке 2.2.

|  |
| --- |
|  |

Рисунок 2.2 – Структура клиентской части приложения

Файл «launch.json» содержит информацию запуска приложение. Директория «node\_modules» хранит необходимые библиотеки для работы приложения. Директория «Components» содержит компоненты, которые отображаются пользователю. Директория «Services» хранит функции генерации запроса на маршрутизатор. Директория «SetTable» содержит файлы для отображения данных. Файл «App.js» содержит корневой тег, который вставляется в «index.js». Файл «context.js» создает контекст. Файл «index.css» содержит информацию о стилях в приложение. Файл «package.json» содержит скрипт для запуска приложения.

## 2.6 Docker

В основе работы Docker лежит стандартизированный способ исполнения кода. Docker – это операционная система для контейнеров. Подобно тому как виртуальная машина создает виртуальное представление аппаратного обеспечения сервера (то есть устраняет необходимость непосредственно управлять таковым), контейнеры создают виртуальное представление серверной операционной системы. После установки на каждый сервер Docker предоставляет доступ к простым командам, необходимым для сборки, запуска или остановки контейнеров. Схема основных компонентов Docker показана на рисунке.

|  |
| --- |
|  |

Рисунок 2.3 – Схема основных элементов Docker

Использование Docker позволяет быстрее и эффективнее доставлять или перемещать код, стандартизирует выполняемые приложениями операции и в целом экономит средства, оптимизируя использование ресурсов. Благодаря Docker пользователи получают объект, который с высокой надежностью можно запускать на любой платформе. Простой и понятный синтаксис Docker обеспечивает полный контроль над выполняемыми операциями. Повсеместное внедрение контейнеров подразумевает доступ к разнообразным инструментам и готовым приложениям, которые можно использовать с Docker.

Контейнеры Docker можно использовать в качестве основных компонентов для создания современных платформ и приложений. Docker упрощает сборку и запуск распределенных микросервисных архитектур, развертывание кода с помощью стандартизированных конвейеров непрерывной интеграции и доставки, создание высокомасштабируемых систем обработки данных и полностью управляемых платформ для разработчиков.

1. **Разработка программного средства**

## 3.1 Реализация серверной части приложения

В ходе реализации серверной части приложения было разработано микросервисное приложение. Каждый сервис представляет собой web-api приложение.

Листинг 3.1 демонстрирует контроллер, разработанный для управления ролями.

|  |  |
| --- | --- |
| |  | | --- | | [Route("api/[controller]")]  [ApiController]  public class RoleController : Controller  {  private readonly IAsyncServiceRole<Role> asyncService;  public RoleController(  IAsyncServiceRole<Role> asyncService  )  {  this.asyncService = asyncService;  }  [Authorize(Roles = " ADMIN")]  [HttpGet]  public async Task<IEnumerable<Role>> GetAll()  {  return await asyncService.GetAll();  }  } | |

Листинг 3.1 – Реализация контроллера

Авторизация в данном приложении реализуется при помощи JWT-токена, который выдаётся ответом пользователю при аутентификации. Используется стандартная аутентификации только со своей конфигурацией. Конфигурация JWT-токена показана в листинге 3.2.

|  |  |
| --- | --- |
| |  | | --- | | public class AuthOptions  {  public const string ISSUER = "MyAuthServer"; // издатель токена  public const string AUDIENCE = "MyAuthClient"; // потребитель токена  const string KEY = "mysupersecret\_secretkey!123"; // ключ для шифрации  public const int LIFETIME = 60; // время жизни токена - 1 минута  public static SymmetricSecurityKey GetSymmetricSecurityKey()  {  return new SymmetricSecurityKey(Encoding.ASCII.GetBytes(KEY));  }  }  } | |

Листинг 3.2 – Конфигурация JWT-токена

Сервисный слой — это шаблон проектирования, который инкапсулирует бизнес логику вашего приложения и определяет границу и набор допустимых операций с точки зрения взаимодействующих с ним клиентов. Листинг сервиса, разработанного для управления новостями представлен в приложении В.

Для доступа к данным используются репозитории. Это несколько интерфейсов которые используют Entity для взаимодействия с базой данных. Они обеспечивает основные операции по сохранения, удалению данных. Метод репозитория, разработанный для доступа к таблице с пользователями, показан в листинге 3.3.

|  |  |
| --- | --- |
| |  | | --- | | public override async Task<IEnumerable<User>> Get()  {  return await \_context.Users.Include(i => i.Role).ToListAsync();  } | |

Листинг 3.3 – Метод репозитория для получения всех пользователей

С помощью атрибутов валидации, можно отлавливать ошибки. При помощи кастомных атрибутов, можно самостоятельно писать логику обработки свойств. Листинг кастомного атрибута валидации в приложении Г.

## 3.2 Реализация клиентской части приложения

Для реализации клиентской части приложения используется React.

Для взаимодействия с сервером и отправки запросов по протоколу http применяется пакет Axios. Этот пакет определяет ряд методов для отправки различного рода запросов: GET, POST, PUT, DELETE. Сервис, разработанный для получения данных о пользователях представлен в листинге 3.4.

|  |  |
| --- | --- |
| |  | | --- | | import axios from "axios";  import GetJwtToken from "../GetJwtToken";  const URI = "http://localhost:37766/ServiceUser/api/User";  async function GetUsers() {  try {  console.log({ headers: GetJwtToken() });  const response = await axios.get(URI, { headers: GetJwtToken() });  return response;  } catch (error) {  return error.response;  }  }  export default GetUsers; | |

Листинг 3.4 – Сервис для получения данных о пользователях

React Router это библиотека маршрутизации в React. Оно хранит интерфейс приложения синхронизированным с URL на браузере. Он позволяет вам маршрутизировать поток данных в вашем приложении понятным способом. Листинг React Router представлен в листинге 3.5.

|  |  |
| --- | --- |
| |  | | --- | | <Router>  <Header />  <Routes>  <Route exact path="/" element={<Registration />} />  <Route path="/login" element={<Authorization />} />  <Route path="/home/ADMIN" element={<HomeAdmin />} />  <Route path="/home/EMPLOYEE" element={<HomeEmployee />} />  <Route path="/home/USER" element={<HomeUser />} />  <Route exact path="/home/Employee" element={<HomeAdmin />} />  <Route path="/home/Employee/add" element={<PostEmployee />} />  <Route path="/home/Employee/put" element={<PutEmployee />} />  <Route path="/home/put/user" element={<PutUser />} />  <Route path="/home/Car" element={<ListCar />} />  <Route path="/home/car/details" element={<DetailsCar />} />  <Route path="/home/Car/add" element={<PostCar />} />  <Route path="/home/Car/put" element={<PutCar />} />  <Route path="/home/ClientCar/add" element={<PostClientCat />} />  <Route path="/home/ClientCar" element={<ListClientCar />} />  <Route path="/home/ClientCar/put" element={<PutClientCar />} />  <Route path="/home/New/add" element={<PostNew />} />  <Route path="/home/New/put" element={<PutNew />} />  <Route path="/home/New" element={<ListNew />} />  <Route path="/home/Order/add" element={<PostOrder />} />  <Route path="/home/Order" element={<ListOrder />} />  <Route path="/home/TestDrive" element={<ListTestDrive />} />  <Route path="/home/TestDrive/add" element={<PostTestDrive />} />  <Route path="/home/CarEquipment" element={<ListCarEquipment />} />  <Route path="/home/CarEquipment/add" element={<PostCarEquipment />} />  <Route path="/home/CarEquipment/put" element={<PutCarEquipment />} />  <Route path="/home/Car/ListBuy" element={<CarUser />} />  <Route path="/home/Car/DetailsBuy" element={<CarUserDetail />} />  <Route path="/home/Car/Buy" element={<BuyCar />} />  <Route path="/home/Car/User" element={<UserCarList />} />  <Route path="/home/Car/User/put" element={<UserCarPut />} />  <Route path="/home/NewUser" element={<NewListUser />} />  </Routes>  </Router> | |

Листинг 3.5 – React Router

Одним из ключевых элементов приложения являются компоненты. Компонент управляет отображением представления на экране. Листинг компонент для авторизации представлен в приложении Д.

Также на клиентской части генерируются запросы на облачное хранилище, которое хранит изображение новостей и комплектаций. Функция, которая генерирует запрос на хранилище представлена рисунке 2.3.

|  |
| --- |
| import axios from "axios";  const API\_URL = "https://api.cloudinary.com/v1\_1/CourseAspCore/image/upload";  class ImageService {  async uploadImage(image) {  const formData = new FormData();  formData.append("file", image);  formData.append("upload\_preset", "kker0vet");  try {  const response = await axios.post(API\_URL, formData);  return response.data.url;  } catch (error) {  console.log(error.response);}}}  export default new ImageService(); |

Листинг 3.1 – Функция генерации запроса на облачный сервис

После отправки запроса, возвращается url изображения, который добавляется в базу данных.

## 3.3 Развёртывание приложения в docker

Docker может автоматически создавать образы читая инструкции из Dockerfile. Dockerfile представляет из себя текстовый документ содержащий все команды для сборки образа.

Dockerfile разработанный для серверной части приложения показан в листинге 3.6.

|  |
| --- |
| FROM openjdk:17-ea-jdk-oracle COPY build/libs/courses-0.0.1-SNAPSHOT.jar courses-0.0.1-SNAPSHOT.jar EXPOSE 8085 ENTRYPOINT ["java","-jar","/courses-0.0.1-SNAPSHOT.jar"] |

Листинг 3.6 – Dockerfile серверной части приложения

Для докерезации клиентского приложения нужно добавить конфигурацию для nginx. NGINX, является известным программным обеспечением с открытым исходным кодом для веб-сервера. В своём первоначальном выпуске оно функционировало для веб-обслуживания HTTP. Однако сегодня оно также служит обратным прокси-сервером, балансировщиком нагрузки HTTP и почтовым прокси-сервером.

Официальный выпуск NGINX состоялся в октябре 2004 года. Создатель программного обеспечения, начал свой проект в 2002 году, пытаясь ответить на проблему C10k. C10k — это задача одновременного управления десятью тысячами соединений. Сегодня существует ещё больше подключений, которые должны обрабатывать веб-серверы. Конфигурация для nginx показана на листинге 3.7.

|  |
| --- |
| events{}  http {  include /etc/nginx/mime.types;  server {  listen 80;  server\_name localhost;  root /usr/share/nginx/html;  index index.html;  location / {  try\_files $uri $uri/ /index.html;  }  } } |

Листинг 3.7 – Конфигурация nginx

Dockerfile, разработанный для клиентской части приложения, описан в листинге 3.8.

|  |
| --- |
| FROM node:12.7-alpine AS build WORKDIR /src/app COPY package.json package-lock.json ./ RUN npm install COPY . . RUN npm run build  FROM nginx:alpine COPY nginx.conf /etc/nginx/nginx.conf COPY —from=build /src/app/dist/angular-jwt-authentication-example /usr/share/nginx/html |

Листинг 3.8 – Dockerfile клиентской части приложения

Далее нужно создать image клиентского и серверного приложений для этого в консоли нужно прописать команду, приведённую в листинге 3.9, находясь в папке с определённым проектом.

|  |
| --- |
| docker build -t container\_name path\_to\_docker\_file |

Листинг 3.9 – Команда для сборки image

Docker Compose инструмент для создания и запуска многоконтейнерных Docker приложений. В Compose, вы используете специальный файл для конфигурирования ваших сервисов приложения. Конфигурация Docker Compose расположена в файле docker-compose.yml. Она представлена в листинге 3.10.

|  |
| --- |
| version: '3.7'  services:  db:  container\_name: db  image: postgres:13.3  restart: always  environment:  POSTGRES\_DB: stage  POSTGRES\_PASSWORD: secret  POSTGRES\_USER: postgres  PGDATA: "/var/lib/postgresql/data/pgdata"  volumes:  - .:/var/lib/postgresql/data  ports:  - '5432:5432'  backend:  container\_name: backend  image: backend  restart: always  links:  - db  ports:  - '8085:8080'  front:  links:  - backend  container\_name: front  image: front  ports:  - '4200:80' |

Листинг 3.10 – Docker-compose.yml файл

Команда запуска контейнеров представлена в листинге 3.11.

|  |
| --- |
| Docker-compose up |

Листинг 3.11 – Команда запуска контейнеров

После этого приложения станут доступны для использования.

1. **Тестирование программного средства**

Для обеспечения корректной работы программы обрабатываются различные виды ошибок, возникающие в процессе работы. Приложение работает с базой данных, поэтому допускать ввод некорректных данных недопустимо, т. к. это может привести к неработоспособности приложения.

При регистрации пользователю необходимо заполнить все поля, если ввести некоректные данные будет выведено соответствующие сообщение, показанное на рисунке 4.1.

|  |
| --- |
|  |

Рисунок 4.1 ­– Регистрация с некоректными данными

При регистрации проверяется правильность даты рождения, если пользователю меньше 18 лет, то он не сможет зарегистрирроваться 4.2.

|  |
| --- |
|  |

Рисунок 4.2 ­– Регистрация с неверной датой рождения

Если пользователь пытается зарегистрироваться с длиной именем больше 50 символов и меньше 3 ему выдаст ошибку, которая представлена на рисунке 4.3.

|  |
| --- |
|  |

Рисунок 4.3 ­– Регистрация с неверной длиной имени

Если администратор попытается добавить новость с неверным форматом файла, то ему выдаст ошибку, которая представлена на рисунке 4.4.

|  |
| --- |
|  |

Рисунок 4.4 ­– Добавление новости с неверным форматом изображения

Если пользователь при авторизации вводит неверный пароль, то ему будет выдано сообщение, показанное на рисунке 4.5.

|  |
| --- |
|  |

Рисунок 4.5 ­– Вход в систему с неверным форматом пароля

Если пользователь попытается добавить автомобиль со скидкой больше 100% ему выдаст ошибку, которая изображена на рисунке 4.6.

|  |
| --- |
|  |

Рисунок 4.6 ­– Добавление курса с невалидным процентом акции

Если пользователь попытается ввести вин код существуещего автомобиля, то выведится ошибка с сответствующим сообщение, которое показано на рисунке 4.7.

|  |
| --- |
|  |

Рисунок 4.7 ­– Добавления авто с существующим вин кодом

В ходе тестирования была проведена проверка соответствия ПО предъявляемым требованиям.

1. **Руководство пользователя**

Для использования программного средства необходимо перейти в браузер по адресу <http://localhost:3000/>.

Сначала нужно зарегистрироваться на сайте. Страница регистрации представлена на рисунке 5.1.

|  |
| --- |
|  |

Рисунок 5.1 ­– Страница регистрации

Далее нужно, чтобы администратор активировал ваш аккаунт и вы будете пользователем или добавил вас в сотрудники с соотвествующией ролью и ваш аккаунт станет активным. После чего вам требуется перейти на страницу с авторизацией и ввести данные, которые вы использовали при регистрации. Страница входа представлена на рисунке 5.2.

|  |
| --- |
|  |

Рисунок 5.2 ­– Страница входа

Рассмотрим функционал администратора. Он может управлять пользователями, сотрудниками, автомобилями, тест-драйвами, комплектациями к автомобилю. Страница пользователями представлена на рисунке 5.3.

|  |
| --- |
|  |

Рисунок 5.3 ­– Страница для управления пользователями

Также администратор имеет возможность управлять новостями. Страница управления новостями представлена на рисунке 5.4.

|  |
| --- |
|  |

Рисунок 5.4 ­– Страница для управления курсами

Рассмотрим функционал сотрудника. Он может управлять атвомобилями, новостями, заказами, тест-драйвами. Страница, в которой сотрудник может создать комплектацию атомобиля, показана на рисунке 5.5.

|  |
| --- |
|  |

Рисунок 5.5 ­– Страница для комплектации автомобиля

На странице, показанной на рисунке 5.6, сотрудник может добавлять машины пользователям, после добавления автомобиля пользователю, информации о покупке будет хранится в таблице с заказами.

|  |
| --- |
|  |

Рисунок 5.6 ­– Страница для добавления автомобиля

На странице, показанной на рисунке 5.7, сотрудник изменяет новость и он может добавить к существующей новости ещё изображения либо изменить добавленные изображения.

|  |
| --- |
|  |

Рисунок 5.7 ­– Страница для обновления новости

Рассмотрим функционал пользователя. На странице, показанной на рисунке 5.8, пользователь может записаться на тест-драйв для этого ему надо выбрать время, дату и вин код автомобиля и если данные окажутся валидными, то он запишется на тест-драйв.

|  |
| --- |
|  |

Рисунок 5.8 ­– Страница для записи на тест-драйв

Также пользователь, может добавить автомобиль в продажу и добавить автомобиль к себе в лист его автомобилей.

**Заключение**

В ходе выполнения курсового проекта мной были рассмотрены основные аналоги разрабатываемого программного средства. Затем был сформирован набор функциональных требований, покрывающих необходимый функционал.

Серверная часть приложения была разработана с помощью платформы ASP.NET Core. Также была разработана реляционная база данных для веб-приложения, использующая экземпляр MSSql, имеющая в своем наборе 10 таблицы. С помощью библиотеки React была разработана клиентская часть приложения.

Было настроено развёртывание всего приложения, как клиентской части, так и серверной вместе с базой данных, в Docker-контейнерах.

Приложение было протестировано и был составлено руководство пользователя.

Пользователями приложения являются клиенты, сотрудники и администраторы. В рамках разработанного приложения для прохождения курсов были реализованы следующие требования:

* регистрация и авторизация пользователей;
* обеспечивать реализацию 3 ролей: пользователь, сотрудник, администратор;
* обеспечивать авторизацию пользователей, сотрудников и администраторов;
* обеспечивать создание, редактирование и удаление сотдруников;
* обеспечивать поиск по комплектации;
* обеспечивать возможность записи на тест-драйв;
* обеспечивать создание, редактирование и удаление автомобилей;
* обеспечивать активацию и деактивацию пользователей;
* обеспечивать возможность просмотра новостей, детального описания автомобиля;
* обеспечивать возможность просматривать свои автомобили пользовотелям.

**Список используемых источников**

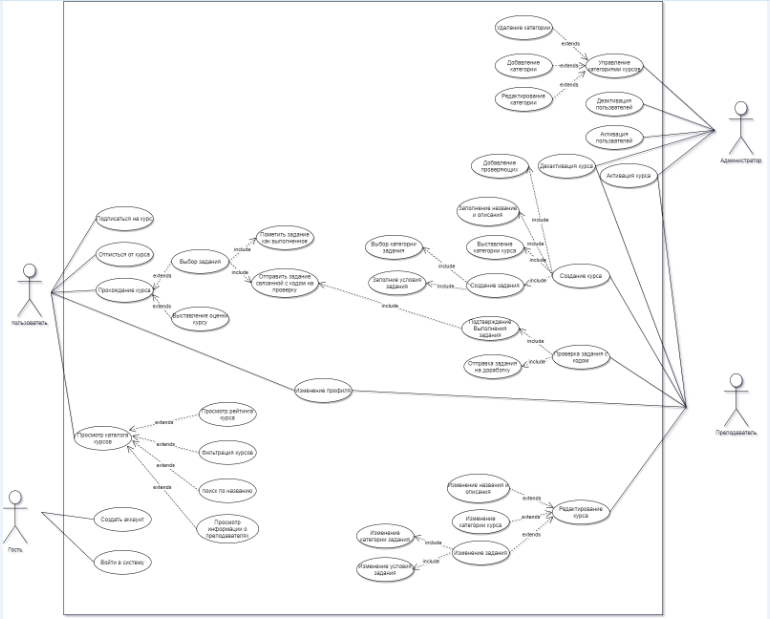
1. Spring boot // Официальная документация [Электронный ресурс]. – Режим доступа: https://spring.io/projects/spring-boot. – Дата доступа: 23.11.2021.

2. Angular // Официальная документация [Электронный ресурс]. – Режим доступа https://angular.io/. – Дата доступа: 12.11.2021.

3. Docker // Официальная документация [Электронный ресурс]. – Режим доступа https://docs.docker.com/. – Дата доступа: 22.11.2021.

4. Hibernate// [Электронный ресурс]. – Режим доступа https://hibernate.org/. – Дата доступа: 29.11.2021.

**ПРИЛОЖЕНИЕ А**



**ПРИЛОЖЕНИЕ Б**

|  |
| --- |
|  |

**ПРИЛОЖЕНИЕ В**

|  |
| --- |
| import com.belstu.course.dto.courseType.CourseTypeDto;  import com.belstu.course.mapper.CourseTypeMapper;  import com.belstu.course.model.CourseType;  import com.belstu.course.repository.CourseTypeRepository;  import lombok.RequiredArgsConstructor;  import lombok.extern.slf4j.Slf4j;  import org.springframework.stereotype.Service;  import java.util.List;  @Service  @Slf4j  @RequiredArgsConstructor  public class CourseTypeService {  private final CourseTypeMapper courseTypeMapper;  private final CourseTypeRepository courseTypeRepository;  public void create(CourseTypeDto courseType) {  courseTypeRepository.save(  new CourseType(courseType.getName())  );  }  public void edit(Long id, CourseTypeDto courseTypeDto) throws Exception {  var courseType = courseTypeRepository.findById(id).orElseThrow(  () -> new Exception(""));  courseType.setName(courseTypeDto.getName());  courseTypeRepository.save(courseType);  }  public List<CourseTypeDto> findAll() {  return courseTypeMapper.toDto(courseTypeRepository.findAll());  }  public CourseTypeDto findById(Long id) throws Exception {  return courseTypeMapper.toDto(courseTypeRepository.findById(id).orElseThrow(  () -> new Exception("типа курса с таким id нет в системе")));  }  } |

**ПРИЛОЖЕНИЕ Г**

|  |
| --- |
| import com.belstu.course.dto.ErrorDto;  import com.belstu.course.dto.ValidationErrorDto;  import com.belstu.course.exception.JwtAuthenticationException;  import com.belstu.course.exception.RegistrationException;  import lombok.extern.slf4j.Slf4j;  import org.springframework.http.HttpStatus;  import org.springframework.security.access.AccessDeniedException;  import org.springframework.security.authentication.BadCredentialsException;  import org.springframework.security.core.userdetails.UsernameNotFoundException;  import org.springframework.validation.FieldError;  import org.springframework.web.bind.MethodArgumentNotValidException;  import org.springframework.web.bind.annotation.ControllerAdvice;  import org.springframework.web.bind.annotation.ExceptionHandler;  import org.springframework.web.bind.annotation.ResponseBody;  import org.springframework.web.bind.annotation.ResponseStatus;  import java.sql.SQLException;  @Slf4j  @ControllerAdvice  public class ExceptionAdvice {  @ResponseStatus(HttpStatus.UNAUTHORIZED)  @ExceptionHandler({AccessDeniedException.class})  @ResponseBody  public ErrorDto handleAccessDeniedExceptions(Exception ex) {  ErrorDto error = new ErrorDto("Access is denied");  log.error(String.valueOf(error));  return error;  }  @ResponseStatus(HttpStatus.BAD\_REQUEST)  @ExceptionHandler(MethodArgumentNotValidException.class)  @ResponseBody  public ValidationErrorDto handleValidationExceptions(MethodArgumentNotValidException ex) {  return ex.getBindingResult()  .getAllErrors()  .stream()  .findFirst()  .map(objectError -> new ValidationErrorDto(  ((FieldError) objectError).getField(),  objectError.getDefaultMessage()  ))  .orElseGet(() -> new ValidationErrorDto("", ""));  }  @ResponseStatus(HttpStatus.BAD\_REQUEST)  @ExceptionHandler({  RegistrationException.class,  UsernameNotFoundException.class,  JwtAuthenticationException.class,  Exception.class})  @ResponseBody  public ErrorDto handleExceptions(Exception ex) {  ErrorDto error = new ErrorDto(ex.getMessage());  log.error(String.valueOf(error));  return error;  }  @ResponseStatus(HttpStatus.BAD\_REQUEST)  @ExceptionHandler({SQLException.class})  @ResponseBody  public ErrorDto handleSqlExceptions(Exception ex) {  ErrorDto error = new ErrorDto("duplicate");  log.error(String.valueOf(error));  return error;  }  @ResponseStatus(HttpStatus.BAD\_REQUEST)  @ExceptionHandler({BadCredentialsException.class})  @ResponseBody  public ErrorDto handleBadCredentialsException() {  log.error("Неверный логин или пароль");  return new ErrorDto("Неверный логин или пароль");  }  } |

**ПРИЛОЖЕНИЕ Д**

|  |
| --- |
| import { Component, OnInit } from '@angular/core';  import { FormBuilder, FormGroup, Validators } from '@angular/forms';  import { Router } from '@angular/router';  import {  ChangeCourseLinksService,  CourseService,  CourseTypeService,  TaskService,  } from '@app/\_services';  import { MatDialog } from '@angular/material/dialog';  import { CourseType } from '@app/\_models/courseType';  import \* as BehaviorSubject from 'rxjs/internal/BehaviorSubject';  import { User } from '@app/\_models';  import { Course, Task } from '@app/\_models/course';  import { first } from 'rxjs/operators';  @Component({  selector: 'edit-course',  templateUrl: './edit.course.component.html',  styleUrls: ['./edit.course.component.css'],  })  export class EditCourseComponent implements OnInit {  editCourseSubmitted = false;  editedCourse: Course;  deleteCourseForm: FormGroup;  editCourseForm: FormGroup;  isEditorShow: Boolean = false;  courseTypeNames: CourseType[];  courses: Course[];  activeStatuses: string[] = ['Активный', 'Неактивный'];  private currentUserSubject: BehaviorSubject.BehaviorSubject<User>;  constructor(  private formBuilder: FormBuilder,  private courseTypeService: CourseTypeService,  private courseService: CourseService,  private router: Router,  public dialog: MatDialog  ) {  this.currentUserSubject = new BehaviorSubject.BehaviorSubject<User>(  JSON.parse(localStorage.getItem('currentUser'))  );  }  ngOnInit(): void {  this.courseTypeService.getAll().subscribe({  next: (courseTypeNames) => {  this.courseTypeNames = courseTypeNames as unknown as CourseType[];  },  });  }  deleteCourse() {  var course = this.deleteCourseForm.get('courseNames').value as Course  this.courseService  .deleteById(course.id)  .subscribe({  next: () => {  this.router  .navigateByUrl('/', { skipLocationChange: true })  .then(() => {  this.router.navigate(['/teacher/course/edit']);  });  },  error: (error) => {  },  });  }  onEdit() {  this.editCourseSubmitted = true;  if (this.editCourseForm.invalid) {  return;  }  var courseType = new CourseType();  courseType.id = this.editCourseForm.get('editCourseType').value;  var course = new Course();  course.id = this.editedCourse.id  course.name = this.editCourseForm.get('editCourseName').value;  course.description = this.editCourseForm.get('editCourseDescription').value;  course.type = courseType  if (this.editCourseForm.get('editCourseStatus').value == 'Активный')  course.status = 'ACTIVE'  else if (this.editCourseForm.get('editCourseStatus').value == 'Неактивный')  course.status = 'IN\_ACTIVE'  this.courseService  .edit(course)  .pipe(first())  .subscribe({  next: () => {  this.router  .navigateByUrl('/', { skipLocationChange: true })  .then(() => {  this.router.navigate(['/teacher/course/edit']);  });  },  error: (error) => {  this.editCourseForm  .get('editCourseName')  .setErrors({ dublicate: 'a' });  },  });  }  showEditForm() {  var course = this.deleteCourseForm.get('courseNames').value as Course;  this.editedCourse = course  var status = ''  if (course.status == 'ACTIVE')  status = 'Активный'  else if (course.status == 'IN\_ACTIVE')  status = 'Неактивный'  this.editCourseForm = this.formBuilder.group({  editCourseName: [course.name, Validators.required],  editCourseDescription: [course.description, Validators.required],  editCourseType: [course.type.id, Validators.required],  editCourseStatus: [status, Validators.required],  });  this.isEditorShow = true;  }  get getDeleteCourseForm() {  return this.deleteCourseForm.controls;  }  } |