**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное ГОСУДАРСТВЕННОЕ бюджетное ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

# Факультет информатики

# и вычислительной техники

Кафедра информационно-

вычислительных систем

## **РАЗРАБОТКА БАЗЫ ДАННЫХ И ПРИЛОЖЕНИЯ**

## **«Фирма по продаже запчастей»**

пояснительная записка

к курсовому проекту по дисциплине

“Базы данных”

Вариант **№ 8**

# Выполнил: студент ИВТ-31 \_\_\_\_\_\_\_ Корнилов А.С. \_\_\_\_\_\_\_\_\_

подпись дата

# Проверил: к.т.н., доцент \_\_\_\_\_\_\_\_\_\_Малашкевич В.Б. \_\_\_\_\_\_\_

подпись дата

### Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_

Йошкар-Ола

2021 г.

**Аннотация**

Лист

2

В данной курсовой работе реализовано веб приложение, которое осуществляет связь с базой данных посредством запросов на сервер.

Основная задача данной работы – научиться строить реляционные СУБД, научиться взаимодействовать с базой данных с помощью приложения.

Результатом данной работы является законченное веб-приложение, осуществляющее доступ к базе данных различным группам пользователей. Интерфейс пользователя реализован браузерным приложением, непосредственно запросы к базе данных осуществляются с помощью сервера.

В данной курсовой использовались следующие инструменты разработки:

1. Для написания интерфейсной (клиентской) части приложения использовалась программная платформа Node.js, с помощью которой был загружен и установлен фреймворк AngularJs 12 версии, а также набор библиотек Angular Material для диалоговых окон.
2. Серверная часть работает на программной платформе .Net 5.0, использующая язык C#. Так же использовались следующие фреймворки: ASP.NET Core для использования приложения, как сервера, и Enity Framework Core для взаимодействия с базой данных.
3. База данных Microsoft SQL Server и средство доступа к базе данных Microsoft SQL Server Management Studio.
4. Visual Studio Code для удобства написания и отладки клиентской части веб приложения.
5. Visual Studio для удобства написания и отладки клиентской части веб приложения.
6. Система контроля версий Git, а так же сайт GitHub, для фиксаций изменений проекта.
7. Notepad++ для редактирования конфигурационных файлов.

Лист

3

**Содержание**

Техническое задание.……………………………………………………………….4

Введение……………………………………………………………………………..5

1. Теоретическая часть……………………………………………………………...6

2. Аналитическая часть……………………………………………………………10

2.1 Схема и состав полей………………………………………………………….10

2.2 Выбор СУБД…………………………………………………………………...14

3. Программная часть………………………………………………………………5

3.1 Создание базы данных на языке SQL………………………………………..15

3.2 Серверная часть приложения (BackEnd)…………………………………….19

3.2.1 Настройка связи с базой данных………………………………………..….19

3.2.2 Репозитории, Сервисы, Контроллеры……………………………………...33

3.3 Клиентская часть приложения (FrontEnd)…………………………………...41

3.3.1 Предварительная настройка серверной части………………………….….41

3.3.2 Веб интерфейс приложения…………………………………………….…..42

3.4 Авторизация и регистрация……………………………………………….….51

4. Интерфейс приложения…………………………………………………….….55

5. Система контроля версий Git……………………………………………….....59

Заключение………………………………………………………………………..62

Список литературы…………………………………………………………….…63

**Техническое задание**

Лист

4

**Фирма по продаже запчастей**

Вы работаете в фирме, занимающейся продажей запасных частей для автомобилей. Вашей задачей является отслеживание финансовой стороны работы компании. Основная часть деятельности, находящейся в Вашем ведении, связана с работой с поставщиками. Фирма имеет определенный набор поставщиков, по каждому из которых известны название, адрес и телефон. У этих поставщиков Вы приобретаете детали. Каждая деталь наряду с названием характеризуется артикулом и ценой (считаем цену постоянной). Некоторые из поставщиков могут поставлять одинаковые детали (один и тот же артикул). Каждый факт покупки запчастей у поставщика фиксируется в базе данных, причем обязательными для запоминания являются дата покупки и количество приобретенных деталей.

Список обязательных полей:

Поставщики (Код поставщика, Название, Адрес, Телефон).

Детали (Код детали, Название, Артикул, Цена, Примечание).

Поставки (Код поставщика, Код детали, Количество, Дата).

Развитие постановки задачи.

Теперь ситуация изменилась. Выяснилось, что цена детали может меняться от поставки к поставке. Поставщики заранее ставят Вас в известность о дате изменения цены и о его новом значении. Нужно хранить не только текущее значение цены, но и всю историю изменения цен. Внести в структуру таблиц изменения, учитывающие этот факт.

**Введение**

Изм.

Разработ.

Проверил

Н. контр.

Утв.

Лист

№ докум.

Подп.

Дата

КНФУ.401000.001П3

Разработка базы данных

и web приложения

«Фирма по продаже запчастей».

Пояснительная записка

Лит.

Лист

Листов

5

63

ИВТ-31 ПГТУ

Корнилов А.С.

Малашкевич В.Б.

Использование баз данных и информационных систем становится неотъемлемой составляющей деловой деятельности современного человека и функционирования шагающих в ногу со временем организаций. В связи с этим большую актуальность приобретает освоение принципов построения и эффективного применения соответствующих технологий и программных продуктов.

Перерабатывать большой объем информации в заданные сроки без специальных средств обработки практически невозможно. К сожалению, большая часть информации еще находится вне ЭВМ, что объясняется отсутствием достаточного количества и номенклатуры технических средств обработки. Но если учесть, что стоимость ЭВМ снижается, то можно предположить, что в перспективе машинная обработка информации будет основной повсеместно. В ЭВМ могут храниться и обрабатываться не только печатные тексты, но и чертежи, фотографии, запись голоса и т.д. Таковы возможности безбумажной технологии обработки информации. Методы организации процессов обработки информации, реализуемые в концепции банков данных и знаний, позволили принципиально по-новому подойти к их реализации в автоматизированных системах.

Банки данных и знаний являются одним из основных компонентов автоматизированных систем различных уровней и типов. Их создают для многих отраслей и сфер народного хозяйства: планирования, учета, управления предприятиями, статистики, здравоохранения и др. Концепция банков данных стала определяющим фактором при создании систем автоматизированной обработки информации.

**1. Теоретическая часть**

Лист

6

База данных (БД) — это поименованная совокупность взаимосвязанных данных, управляемых специальной системой, называемой СУБД.

СУБД представляет собой совокупность специальных языковых и программных средств, облегчающих пользователям выполнение всех операций, связанных с организацией хранения данных, их корректировкой и доступом к ним. СУБД служит, по существу, посредником между пользователем и БД.

БД и СУБД являются составными частями более сложной системы, именуемой банком данных. Банк данных (БнД) — это система, состоящая из баз данных, программных, технических, языковых, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных. Термин «банк данных» схож с понятием «система баз данных». Система баз данных представляет собой совокупность программного обеспечения, данных и аппаратного обеспечения компьютеров, которая реализует набор приложений и моделей данных, и использует СУБД и прикладное программное обеспечение для создания конкретной информационной системы.

БнД является сложной человеко-машинной системой, состоящей из взаимосвязанных и взаимозависимых компонентов (рис. 1.1).

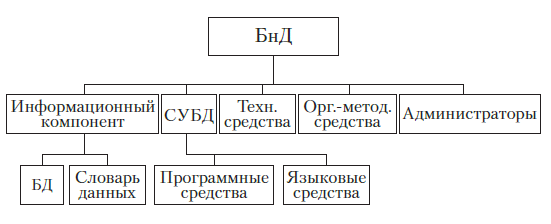


Рис. 1.1. Компоненты БнД

СУБД предоставляет пользователю программные и языковые средства, обеспечивающие взаимодействие всех частей информационной системы при ее функционировании.

Лист

7

Языковые средства СУБД относятся к языкам четвертого поколения. Языковые средства предназначаются для пользователей разных категорий: конечных пользователей, системных аналитиков, профессиональных программистов. По функциональным возможностям выделяются 10 категорий языков. По форме представления различают аналитические, табличные и графические языковые средства.

Наиболее распространенным пока являются базы данных, содержащих обычные символьные данные. Эти БД подразделяются на неструктурированные, частично структурированные и структурированные.

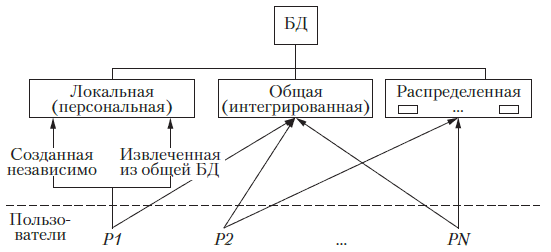
К неструктурированным БД относятся базы, организованные в виде семантических сетей. Частично структурированными можно считать БД, в которых информация представлена в виде обычного текста или гипертекста.

Структурированные БД по типу используемой модели делятся на иерархические, сетевые, реляционные, объектно- ориентированные, смешанные и мультимодельные. Наибольшее коммерческое использование в настоящее время имеют реляционные БД.

По типу хранимой информации БД делятся на документальные и фактографические. Документальные БД содержат сведения о документах на естественном языке — монографиях, научных отчетах, текстах законодательных актов и т.д. Фактографические БД содержат фактические сведения, например, данные о кадровом составе предприятия.

По характеру организации хранения данных и обращения к ним различают локальные (персональные), общие (интегрированные) и распределенные БД (рис. 1.2).

По условиям предоставления услуг различают бесплатные и платные БД. Платные БД делятся на бесприбыльные и коммерческие.



Лист

8

Рис. 1.2. Классификация БД по характеру хранения данных и обращения к ним

По форме собственности БД делятся на государственные и негосударственные.

По степени доступности различают общедоступные и с ограниченным кругом пользователей.

Иерархическая, сетевая и реляционная модели отражают способ установления связей между данными. Иерархическая и сетевая модели предполагают наличие связей между данными, имеющими какой-либо общий признак.

Реляционная модель является простейшей и наиболее привычной формой представления данных в виде таблицы (рис. 1.3).

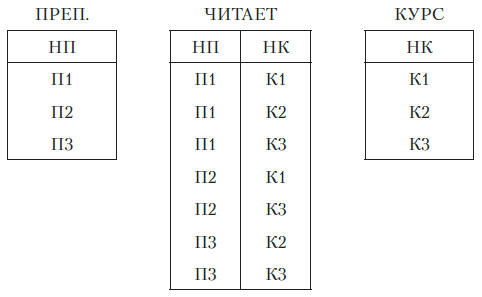


Рис. 1.3. Реляционная модель данных:

НП — номер преподавателя; НК — номер курса

В теории множеств таблице соответствует термин «отношение» (relation), который и дал название реляционной модели. Достоинством РМ является сравнительная простота инструментальных средств ее поддержки, а недостатком — жесткость структуры данных и зависимость скорости выполнения операций от размера таблиц.

Лист

9

При создании моделей данных используются такие понятия, как «сущности», «атрибуты» и «связи». Сущность — это отдельный класс объектов предметной области (сотрудники или клиенты, понятия или события), который должен быть представлен в базе данных. Атрибут — это свойство, описывающее определенный аспект объекта, значение которого следует зафиксировать в описании предметной области. Связь является ассоциативным отношением между сущностями, при котором каждый экземпляр одной сущности соединен с некоторым (в том числе нулевым) количеством экземпляров другой сущности. Объектно-ориентированная модель расширяет определение сущности с целью включения в него не только атрибутов, которые описывают состояние объекта, но и действий, которые с ним связаны, т.е. его поведение. В таком случае говорят, что объект инкапсулирует состояние и поведение.

В настоящее время наиболее распространенными являются системы управления базами данными, поддерживающие реляционную модель данных. Эти системы называются реляционными СУБД.

**2. Аналитическая часть**

Лист

10

В данной курсовой работе необходимо создать приложение для доступа к базе данных и ее управления.

Описание предметной области. Вы работаете в фирме, занимающейся продажей запасных частей для автомобилей. Вашей задачей является отслеживание финансовой стороны работы компании. Основная часть деятельности, находящейся в Вашем ведении, связана с работой с поставщиками. Фирма имеет определенный набор поставщиков, по каждому из которых известны название, адрес и телефон. У этих поставщиков Вы приобретаете детали. Каждая деталь наряду с названием характеризуется артикулом и ценой (считаем цену постоянной). Некоторые из поставщиков могут поставлять одинаковые детали (один и тот же артикул). Каждый факт покупки запчастей у поставщика фиксируется в базе данных, причем обязательными для запоминания являются дата покупки и количество приобретенных деталей.

**2.1 Схема и состав полей**

На основе технического задания была разработана упрощенная логическая схема данных, необходимая для дальнейшего создания БД (рис. 2.1).



Рис. 2.1. Упрощенная логическая схема

На следующем этапе формируется состав таблиц и их полей, определяются типы данных, выделяются первичные и внешние ключи. Основные типы значений для полей:

Лист

11

1. Целочисленный тип *int* для первичных/внешних ключей, полей, хранящих числа, таких как комиссия, требуемый объем з/п, флаг того, что должность открыта у работодателя.
2. Строковый тип *varchar* для всех остальных полей, например номер телефона, наименование работодателя, ФИО соискателя и т.д. Размер строкового типа данных взят с запасом и равен 50.

Первичный ключ у всех таблиц назван *id*, так как по названию таблицы можно определить, что он относится именно к ней;

Ниже представлены таблицы и состав их полей (табл. 1-4).

Таблица 1 Поставщики (Providers)

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип  данных | Длина | Назначение |
| **id** | целое | 10 | Код поставщика (перв. ключ) |
| name | строка | 256 | Наименование поставщика |
| address | строка | 256 | Адрес поставщика |
| phone | строка | 256 | Номер телефона поставщика |

Таблица 2 Запчасти (Details)

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип  данных | Длина | Назначение |
| **id** | целое | 10 | Код запчасти (перв. ключ) |
| name | строка | 256 | Наименование запчасти |
| article | целое | 10 | Артикль товара (уник.) |
| price | строка | 10 | Цена запчасти |
| note | строка | 256 | Краткое описание запчасти |

Лист

12

Таблица 3 Поставки (Deliveries)

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип  данных | Длина | Назначение |
| **id** | целое | 10 | Код поставки (перв. ключ) |
| providerId | целое | 10 | Код поставщика (внеш. ключ) |
| detailId | целое | 10 | Код запчасти (внеш. ключ) |
| count | целое | 10 | Количество товара |
| date | дата | 10 | Дата поставки |
| price | целое | 10 | Итоговая сумма поставки |

Таблица 4 Пользователи (Users)

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип  данных | Длина | Назначение |
| **id** | целое | 10 | Код пользователя (перв. ключ) |
| name | строка | 256 | Имя пользователя |
| login | строка | 256 | Логин пользователя (уник.) |
| password | строка | 256 | Пароль пользователя |
| status | строка | 256 | Статус пользователя |

2.2 Доказательство соответствия схемы данных 3-й нормальной форме

Перейдем к доказательству соответствия схемы данных 3-ей нормальной форме.

Третья нормальная форма предполагает, что каждый столбец, не являющийся ключом, должен зависеть только от столбца, который является ключом, то есть должна отсутствовать транзитивная функциональная зависимость (transitive functional dependency). Транзитивная функциональная зависимость выражается следующим образом: А → В и В → С. То есть атрибут С транзитивно зависит от атрибута А, если атрибут С зависит от атрибута В, а атрибут В зависит от атрибута А (при условии, что атрибут А функционально не зависит ни от атрибута В, ни от атрибута С).

Если столбец зависит не только от первичного ключа, то данный столбец находится не в той таблице, в которой он должен находиться, либо же является производным от других столбцов.

В качестве примера возьмём таблицу поставки (Deliveries).

Лист

13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id | providerId | detailId | count | date | price |
| 2 | 5 | 2 | 3 | 2021-12-22 08:27:50 | 15000 |
| 4 | 4 | 3 | 10 | 2021-12-22 08:28:47 | 12500 |

Чтобы определить, находится ли эта таблица в третьей нормальной форме, мы должны проверить все неключевые столбцы, каждый из них должен зависеть только от первичного ключа, и никаким образом к другим неключевым столбцам он не должен относиться.

Рассмотрим внешние таблицы.

Запчасти:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | name | article | price | note |
| 2 | Рейка | 24124 | 5000 | Рулевая рейка, для автомобиля |
| 3 | Рычаг стояночного тормоза | 352446 | 2500 |  |

Поставщики:

|  |  |  |  |
| --- | --- | --- | --- |
| id | name | article | price |
| 5 | Лучшие запчасти | ул. Строителей 87 | 89027423191 |
| 4 | Карбюраторы даром | ул. Эшпая 56 | 89084916059 |

Таким образом, в наших таблицах отсутствует транзитивная зависимость, и они находятся в третьей нормальной форме.

**2.2 Выбор СУБД**

Лист

14

В качестве СУБД выбрана Microsoft SQL Server 2014 Express, по следующим причинам:

1. является реляционной СУБД;
2. используется ОС Windows. Удобная и бесплатная система от Microsoft, являющая самой популярной операционной системой;
3. богатая и понятная документация;
4. большое количество книг и статей посвящено работе с ней;
5. является высокопроизводительным решением для бизнеса;
6. удобное управление с базой данных посредством входящего в комплект Microsoft SQL Server Management Studio с графическим интерфейсом;
7. возможно управление с помощью утилиты командной строки sqlcmd.

SQL Server Management Studio (SSMS) — это интегрированная среда для управления любой инфраструктурой SQL, от SQL Server до баз данных SQL Azure. SSMS предоставляет средства для настройки, наблюдения и администрирования экземпляров SQL Server и баз данных. С помощью SSMS можно развертывать, отслеживать и обновлять компоненты уровня данных, используемые вашими приложениями, а также создавать запросы и скрипты.

**3. Программная часть**

Лист

15

Программа представляет собой веб приложение, работающее на платформе Microsoft .NET 5.0 core. Пользователь взаимодействует с приложением через браузер. Все запросы пользователя к базе данным проходят через сервер, написанный на языке C#. Для хранения данных используется Microsoft SQL Server.

**3.1 Создание базы данных на языке SQL**

В качестве базы данных мной была использована база данных Microsoft SQL server. А для проверки и отладки базы, я использовал Microsoft SQL server Management Studio.

Язык с помощью которого осуществляется связь управление базой данных, называется SQL (structured query language — «язык структурированных запросов»).

SQL— декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

Является, прежде всего, информационно-логическим языком, предназначенным для описания, изменения и извлечения данных, хранимых в реляционных базах данных. В общем случае SQL (без ряда современных расширений) считается языком программирования не полным по Тьюрингу, но вместе с тем стандарт языка спецификацией SQL/PSM предусматривает возможность его процедурных расширений.

Изначально SQL был основным способом работы пользователя с базой данных и позволял выполнять следующий набор операций:

1) создание в базе данных новой таблицы;

2) добавление в таблицу новых записей;

3) изменение записей;

4) удаление записей;

Лист

16

5) выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);

6) изменение структур таблиц.

Со временем SQL усложнился — обогатился новыми конструкциями, обеспечил возможность описания и управления новыми хранимыми объектами (например, индексы, представления, триггеры и хранимые процедуры) — и стал приобретать черты, свойственные языкам программирования.

При всех своих изменениях SQL остаётся самым распространённым лингвистическим средством для взаимодействия прикладного программного обеспечения с базами данных. В то же время современные СУБД, а также информационные системы, использующие СУБД, предоставляют пользователю развитые средства визуального построения запросов.

В первую очередь необходимо создать структурную схему базы данных:

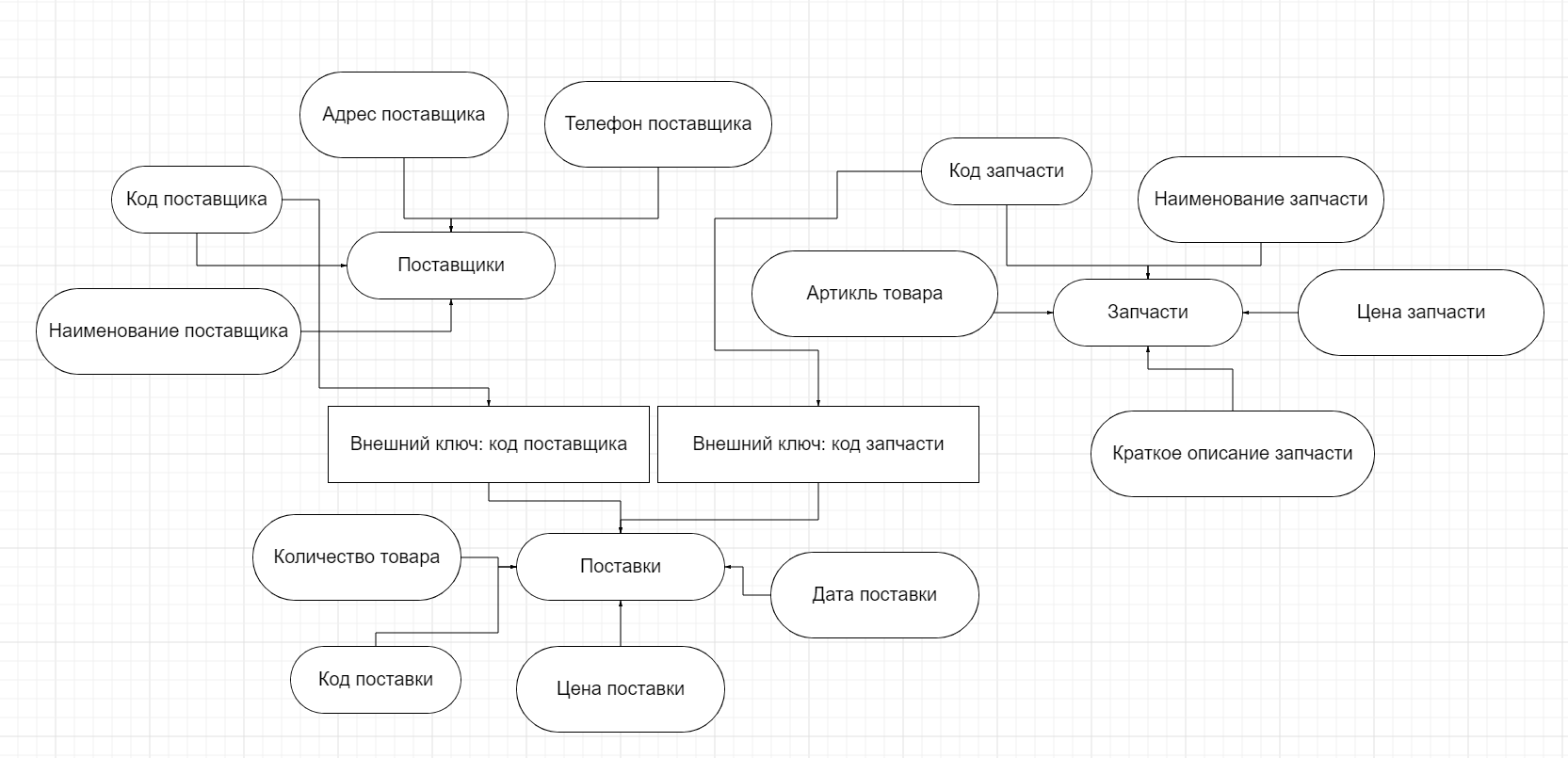


Рис. 3.1 структурная схема базы данных курсового проекта

Рассмотрим скрипты создания базы данных:

Лист

17

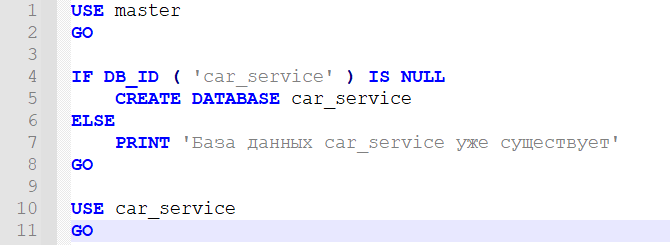


Рис. 3.2 скрипт создания базы данных

Данный скрипт создаёт пустую базу данных, проверяя на существование данной базы данных.

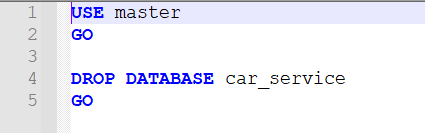
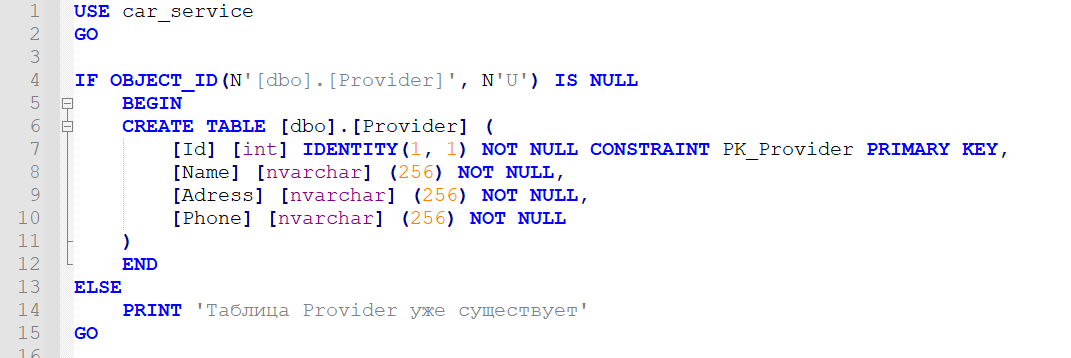


Рис. 3.3 скрипт удаления базы данных

Данный скрипт позволяет удалить базу данных.



Лист

18

Рис. 3.4 скрипт создания таблицы поставщиков

Данный скрипт позволяет создать таблицу поставщиков для данной базы данных. Исходя из задания выделяются поля с соответствующими типами данных. Также идёт проверка на наличие таблице в базе данных.

Аналогично создаются остальные таблицы:

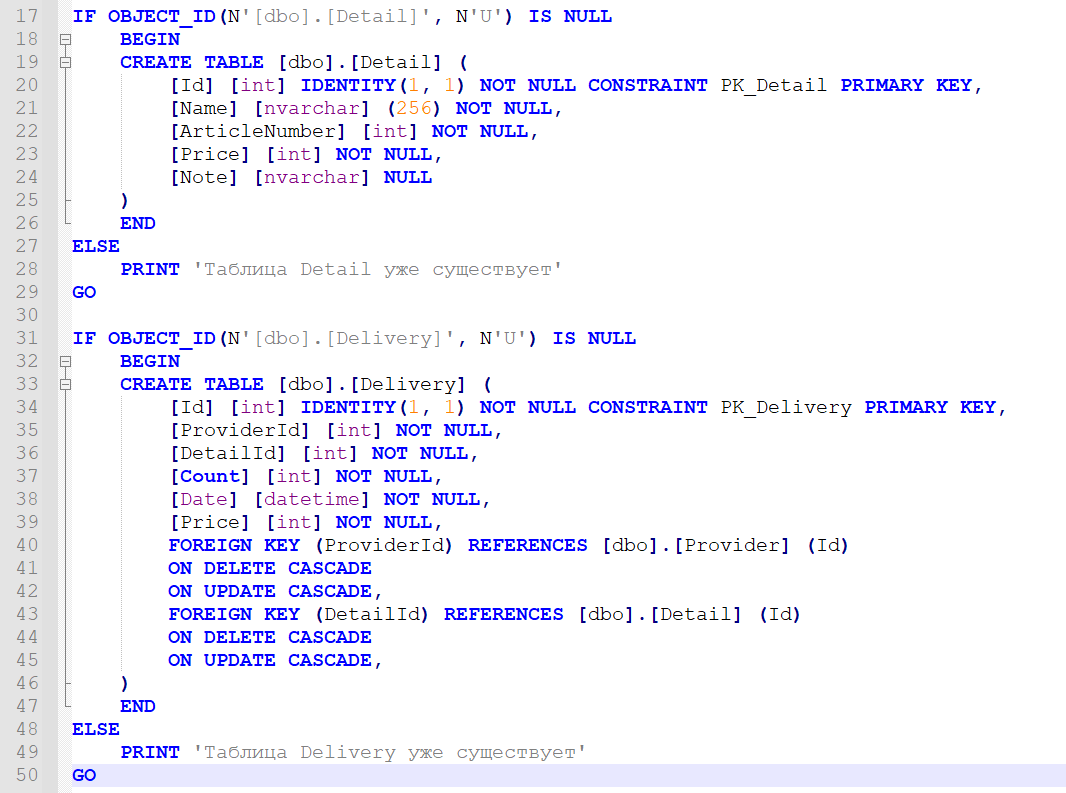


Рис. 3.5 скрипт создания таблиц запчастей и поставок

В таблице поставок указываются ссылки на ключи внешних таблиц, соответственно указываются каскады удаления и обновления. Таким образом создаются зависимости между таблицами.

Лист

19

Скрипты создания таблиц в итоге оказались не нужны, так как в C# присутствуют инструменты для взаимодействия с базой данных. Так что взаимодействие с базой данных будет рассмотрено в серверной части приложения.

**3.2 Серверная часть приложения (BackEnd)**

**3.2.1 Настройка связи с базой данных**

Для того, чтобы осуществить связь серверной части с базой данных используется Microsoft 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 оперируются таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который предлагает Entity Framework, идёт взаимодействие непосредственно с объектами.

Entity Framework Core поддерживает множество различных систем баз данных. Таким образом, мы можем через EF Core работать с любой СУБД, если для нее имеется нужный провайдер.

Для того, чтобы подключиться к базе данных, необходимо в конфигурационном файле прописать подключение к базе данных:

Лист

20

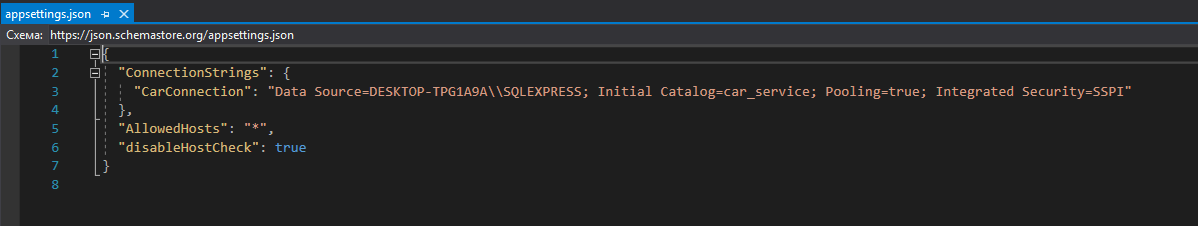


Рис. 3.6 подключение к базе данных car\_service

Рассмотрим, как происходит работа с Entity Framework Core.

Но для начала рассмотрим ещё одну платформу для разработки веб приложений ASP.NET (Active Server Pages для .NET) — платформа разработки веб-приложений, в состав которой входят: веб-сервисы, программная инфраструктура, модель программирования, от компании Майкрософт. ASP.NET входит в состав платформы .NET Framework и является развитием более старой технологии Microsoft ASP.

ASP.NET внешне во многом сохраняет схожесть с более старой технологией ASP, что позволяет разработчикам относительно легко перейти на ASP.NET. В то же время внутреннее устройство ASP.NET существенно отличается от ASP, поскольку она основана на платформе .NET и, следовательно, использует все новые возможности, предоставляемые этой платформой.

Поскольку ASP.NET основывается на Common Language Runtime (CLR), которая является основой всех приложений Microsoft .NET, разработчики могут писать код для ASP.NET, используя языки программирования, входящие в комплект .NET Framework (C#, Visual Basic.NET, J# и JScript .NET).

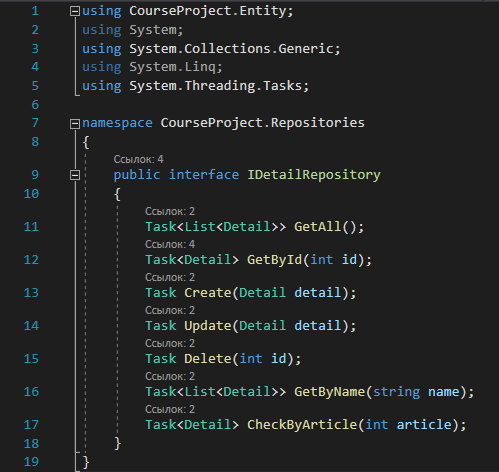
Программная модель ASP.NET основывается на протоколе HTTP и использует его правила взаимодействия между сервером и браузером. При формировании страницы заложена абстрактная программная модель Web Forms и на ней основана основная часть реализации программного кода.

Лист

21

И так, основой серверной части является платформа ASP.NET, входящая в .NET 5.0 Core. Основным классом настроек сервера является Startup, в котором прописываются все необходимые классы, которые должен включать в себя сервер.

Перед тем как рассмотреть EF Core, необходимо акцентировать внимание на использовании в своей работе интерфейсов. Интерфейсы– это еще один инструмент реализации полиморфизма в C#. Интерфейс представляет собой набор методов (свойств, событий, индексаторов), реализацию которых должен обеспечить класс, который реализует интерфейс. Интерфейс может содержать только сигнатуры (имя и типы параметров) своих членов. Интерфейс определяет контракт. Любой class или struct, реализующий этот контракт, должен предоставлять реализацию для членов, определенных в интерфейсе. Начиная с C# 8.0, интерфейс может определять реализацию по умолчанию для членов. Он также может определять члены static, чтобы обеспечить единую реализацию для общих функциональных возможностей. В курсовой работе интерфейсы используются для определения методов, которые должны реализовывать соответствующие классы.



Лист

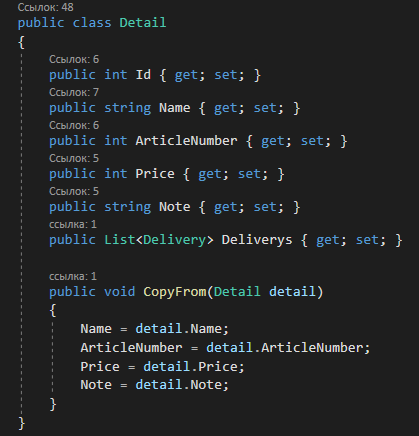
22

Рис. 3.7 пример интерфейса репозитория DetaiRepository.

Теперь можно вернуться к рассмотрению EF core.

Исходя из схемы базы данных на рисунке 3.1, необходимо описать так называемые “сущности” базы данных (Entities). Сущность представляет набор данных, ассоциированных с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их наборами.

Рассмотрим сущности, которые были использованы в курсовом проекте:



Лист

23

Рис. 3.8 сущность Detail

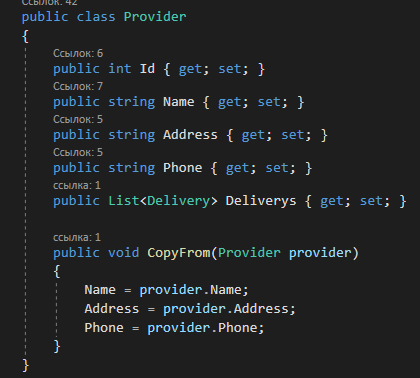
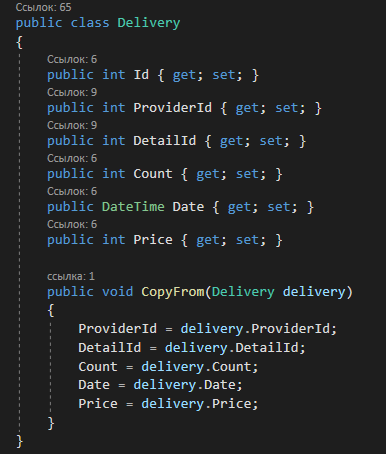


Рис. 3.9 сущность Provider



Лист

24

Рис. 3.10 сущность Delivery

Сущности представляют собой классы, с определёнными полями, отражающие таблицы в базе данных. В отличие от построения таблиц в языке SQL, описание сущностей происходит несколько иначе. Чтобы связать сущности Delivery с Detail и Provider недостаточно просто описать внешние ключи в сущности Delivery. Необходимо так же указать, связь один ко многим, указав, что одна сущность Detail или Provider может много раз использоваться в сущности Delivery. Данная связь выполняется с помощью инициализации списка сущностей Deliveries в Detail и Provider.

Чтобы использовать Entity Framework для запроса, вставки, обновления и удаления данных с помощью объектов .NET, необходимо сначала создать модель, которая сопоставляет сущности и связи, определенные в модели, с таблицами в базе данных.

Эта модель называется контекст данных - класс, унаследованный от класса Microsoft.EntityFrameworkCore.DbContext.

Лист

25

С помощью Fluent API можно задать многочисленные настройки для моделей и их свойств. Однако, если настроек очень много, то они могут утяжелять класс контекста. Однако в Entity Framework Core есть возможность вынести конфигурацию моделей в отдельные классы.

Для вынесения конфигурации во вне необходимо создать класс конфигурации, реализующий интерфейс EntityTypeConfiguration<T>.

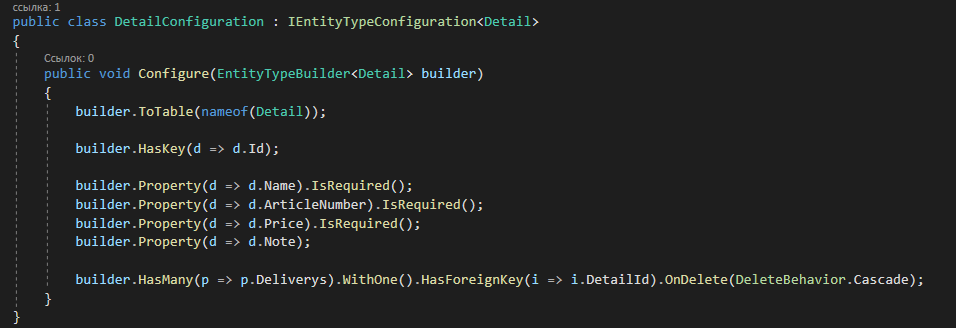


Рис. 3.11 конфигурация для сущности Detail

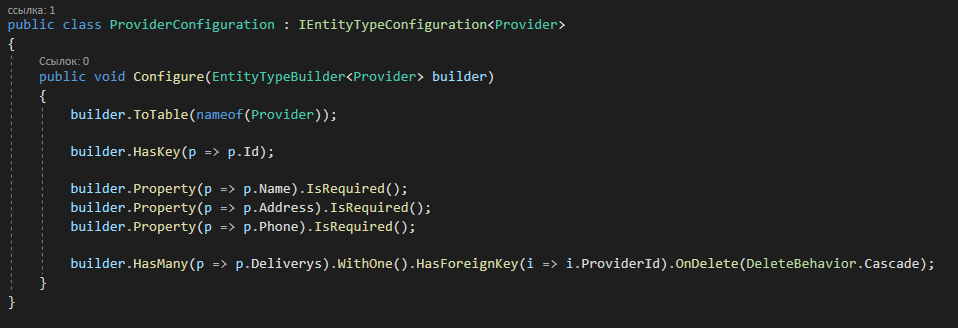
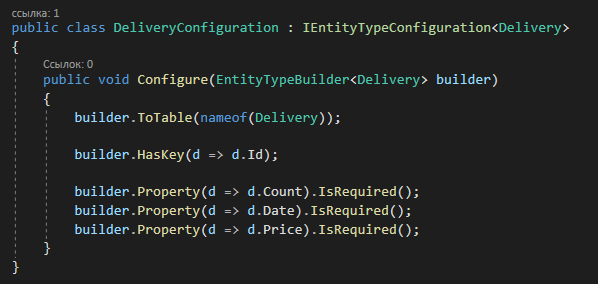


Рис. 3.12 конфигурация для сущности Provider



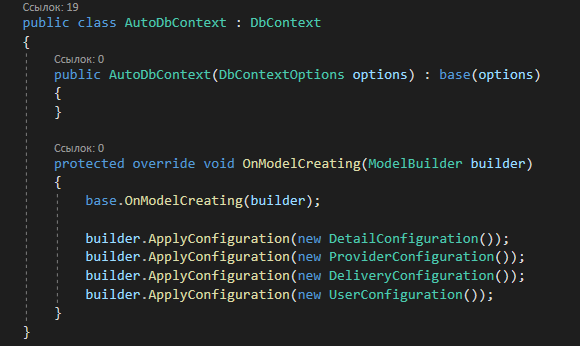
Лист

26

Рис. 3.13 конфигурация для сущности Delivery

Описание конфигураций, позволяет определить, какими свойствами будут обладать поля таблиц в базе данных. Для определения полей, в методе Configure определяется объект builder, через который идёт настройка полей таблиц. Для начала определяется наименование таблицы, исходя из наименования сущности (builder. ToTable (nameof (“Сущность”)). Затем определяется первичный ключ таблиц (HasKey). Чтобы задать остальные поля с данными, используется свойство “Property” и указывается обязательное ли поле для заполнения директивой “IsRequired”. Необходимо обратить внимание, что конфигурация внешних ключей отличается от SQL скрипта. В данном случае поведение внешних ключей определяются внутри внешних таблиц (свойство “HasMany”), а не внутри таблицы, содержащей внешние ключи.

Сущности и конфигурации описаны, теперь можно приступить к реализации контекста данных:



Лист

27

Рис. 3.14 контекст данных AutoDbContext

DbContext, связанный с моделью, можно использовать для:

1) Создание и выполнение запросов

2) Материализация результатов запросов в виде объектов сущностей

3) Отслеживание изменений, внесенных в эти объекты

4) Сохранить изменения объекта обратно в базе данных

5) Привязка объектов в памяти к элементам управления пользовательского интерфейса

При инициализации конструктора используется ключевое слово base, которое используется для доступа к членам базового из производного класса в следующих случаях:

1) Вызов метода базового класса, который был переопределен другим методом.

2) Определение конструктора базового класса, который должен вызываться при создании экземпляров производного класса.

Метод “OnModelCreating” вызывается, если модель для производного контекста была инициализирована, прежде чем модель была заблокирована и использована для инициализации контекста. Реализация этого метода по умолчанию не делает ничего, но его можно переопределить в производном классе и выполнить в нем дальнейшую настройку модели перед ее блокировкой. В методе “OnModelCreating” собираются все конфигурации, которые существуют для данной базы данных.

Лист

28

Так как метод “OnModelCreating” перегруженный метод, то к перегружаемому методу можно обратиться с помощью ключевого слова base (base.OnModelCreating(builder)).

Все необходимые модели для взаимодействия с базой данных проинициализированы, но теперь необходимо сохранить данные конфигурации в саму базу данных. В этом поможет специальный паттерн (шаблон) проектирования, который называется Unit of Work.

Что же такое паттерн проектирования? Это повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста. Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

В приложениях ASP.NET нередко используется паттерн Unit of Work для инкапсулирования логики работы с источниками данных. Паттерн Unit of Work позволяет упростить работу с различными моделями и дает уверенность, что для текущей базы данных, будет использоваться один и тот же контекст данных.

Лист

29

В текущей курсовой работе паттерн Unit Of Work необходим для сохранения изменений данных и конфигураций в базу данных.

Для начала описывается интерфейс для класса UnitOfWork. В текущем проекте он нужен для того, чтобы сохранять текущее состояние контекста в базу. Значит для класса необходимо реализовать только один метод, который необходим для сохранения контекста. Этот метод в данном случае называется “commit” (что означает “зафиксировать”).

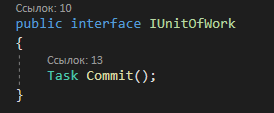


Рис. 3.15 интерфейс класса UnitOfWork

Далее реализуется данный интерфейс реализуется в классе UnitOfWork.

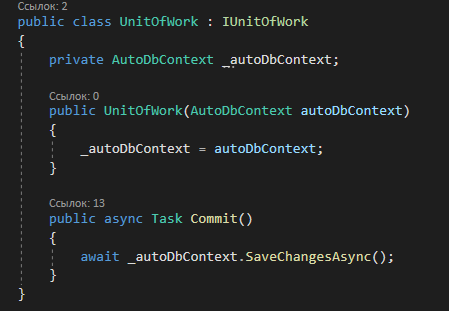


Рис. 3.16 класс UnitOfWork

Класс UnitOfWork представляет собой конструктор, в который записывается текущее состояние контекста, и метод, который сохраняет данный контекст в базу данных.

Лист

30

Чтобы произвести доступ к базе данных, и произвести нужные изменения, необходимо инициализировать DbContext и UnitOfWork в классе Startup, в методе ConfigureService.



Рис. 3.17 инициализация UnitOfWork

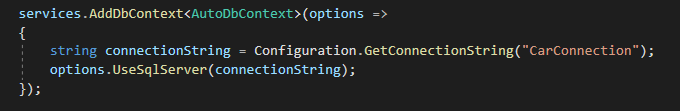


Рис. 3.18 инициализация DbContext

Так же в методе Configure необходимо разрешить DbContext создавать миграции.

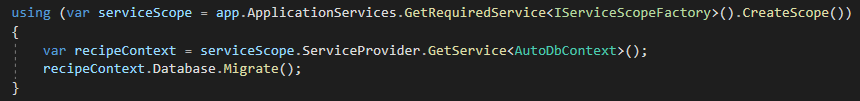
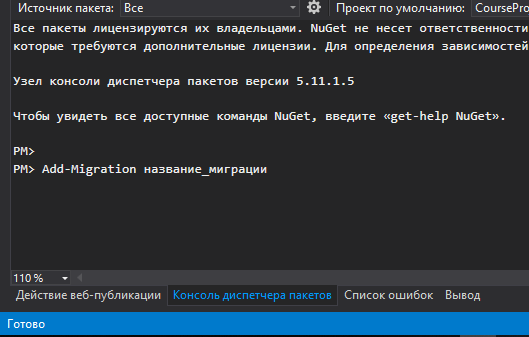


Рис. 3.19 добавление возможности создания миграций в проекте

Для того, чтобы обновить создать или обновить таблицы в базе данных, используются “миграции”. Миграция по сути предствляет план перехода базы данных от старой схемы к новой. Как использовать миграции?

Для создания миграции в окне Package Manager Console вводится следующая команда:



Лист

31

Рис. 3.20 создание миграции для базы данных

Название миграции представляет произвольное название, главное чтобы все миграции в проекте имели разные названия.

После создания миграции ее можно выполнить с помощью команды:



Рис. 3.21 обновление базы данных на основе последней миграции

После создания первой миграции в проекте появится папка под названием “Migrations”, в которой будут храниться все миграции сделанные при разработке приложения.

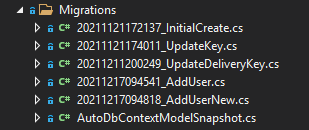
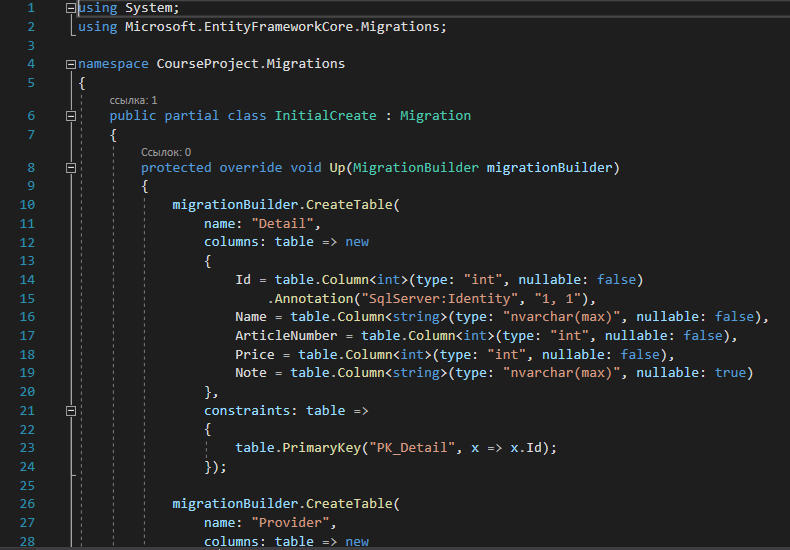


Рис. 3.22 миграции к базе данных

Каждая миграция представляет собой класс, в котором описаны текущие поля для каждой таблицы в базе данных.



Лист

32

Рис. 3.23 пример миграции InititialCreate

В базе данных так же создаётся таблица EFMigrationHistory, в которой отображается история всех миграций для текущей бд.

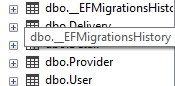


Рис. 3.24 таблица истории миграций в базе данных

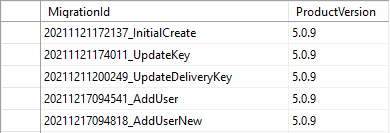


Рис. 3.25 содержимое таблицы EFMigrationHistory

Таким образом с помощью инструментов .NET Core можно производить работу с базами данных.

**3.2.2 Репозитории, Сервисы, Контроллеры**

Лист

33

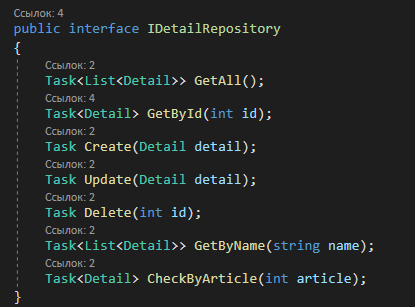
Перейдём к рассмотрению серверной логики работы веб приложения. Как осуществляются запросы к базе данных, как идёт обработка результатов запроса, как данные, полученные с базы данных, отправить пользователю? Все эти вопросы будут рассмотрены в данном разделе.

Одним из наиболее часто используемых паттернов при работе с данными является паттерн ”Репозиторий”. Репозиторий позволяет абстрагироваться от конкретных подключений к источникам данных, с которыми работает программа, и является промежуточным звеном между классами, непосредственно взаимодействующими с данными, и остальной программой.

Допустим, у нас есть одно подключение к базе данных MS SQL Server. Однако, что если в какой-то момент времени захочется сменить подключение с MS SQL на другое - например, к бд MySQL или MongoDB. При стандартном подходе даже в небольшом приложении, осуществляющем выборку, добавление, изменение и удаление данных, пришлось бы сделать большое количество изменений. Либо в процессе работы программы в зависимости от разных условий мы хотим использовать два разных подключения. А паттерн репозиторий позволяет использовать обработку данных от разных бд, что позволяет не переписывать запросы под другую базу данных.

Рассмотрим пример DetailRepository.

Для начало разрабатывается интерфейс репозитория, в котором определяются методы, реализуемые в репозитории.



Лист

34

Рис. 3.26 интерфейс репозитория DetailRepository

Теперь перейдём к реализации.

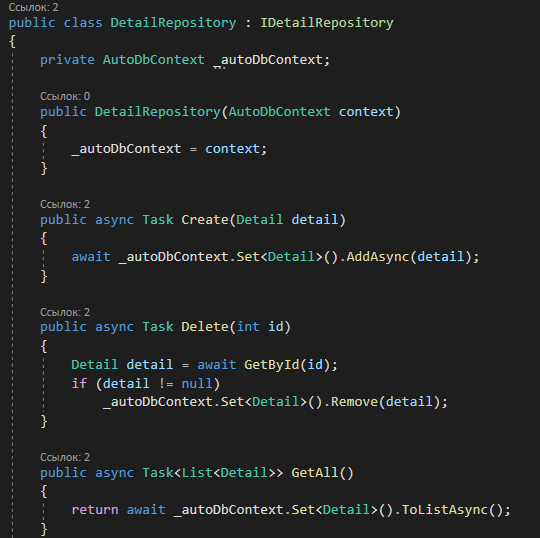


Рис. 3.27 репозиторий DetailRepository

С помощью инструментов EF Core можно выполнять запросы к базе данных посредством DbContext. Это позволяет использовать одни и те же запросы для разных типов баз данных. Сам репозиторий не преполагает никакой логики, то есть должен выполнять только запросы не более. Но для удаления записи из базы данных, мной была реализована дополнительная проверка, чтобы запрос не выдал ошибку, которая может повлечь за собой сбой работы программы.

Лист

35

Чтобы запросы не мешали работе программе, были реализованы асинхронные методы для обработки запроса, а также запросы, которые идут к базе данных, тоже являются асинхронными.

Асинхронность позволяет вынести отдельные задачи из основного потока в специальные асинхронные методы или блоки кода. Особенно это актуально в графических программах, где продолжительные задачи могу блокировать интерфейс пользователя. И чтобы этого не произошло, нужно задействовать асинхронность. Также асинхронность несет выгоды в веб-приложениях при обработке запросов от пользователей, при обращении к базам данных или сетевым ресурсам. При больших запросах к базе данных асинхронный метод просто уснет на время, пока не получит данные от БД, а основной поток сможет продолжить свою работу. В синхронном же приложении, если бы код получения данных находился в основном потоке, этот поток просто бы блокировался на время получения данных.

Ключевыми для работы с асинхронными вызовами в C# являются два ключевых слова: async и await, цель которых - упростить написание асинхронного кода. Они используются вместе для создания асинхронного метода.

Вся логика, обработка ошибок, проверка данных, полученных репозиторием, осуществляется классами, называемыми сервисами.

Лист

36

Интерфейс сервиса выглядит следующим образом.



Рис. 3.28 интерфейс сервиса DetailService

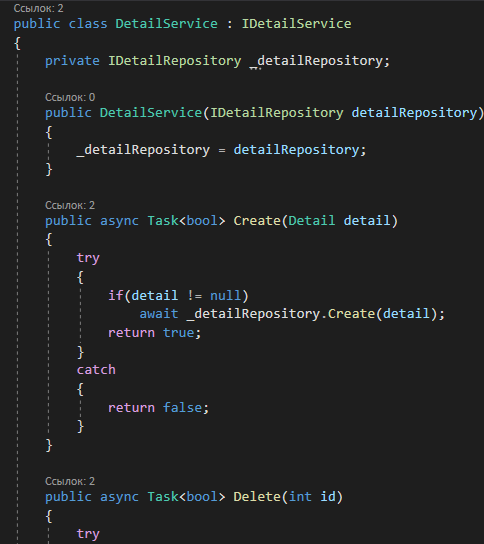
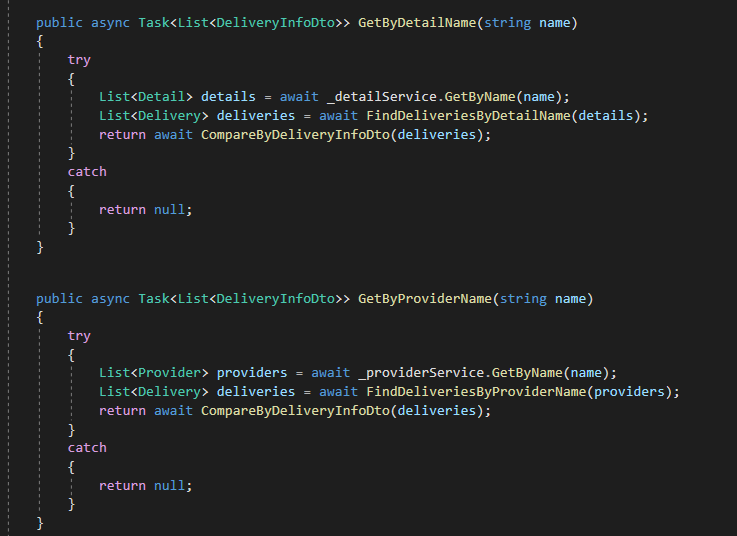


Рис. 3.29 сервис DetailService



Лист

37

Рис. 3.30 пример обработки данных с других сервисов

Сервисы, в первую очередь, нужны, чтобы данные, которые приходят или отправляются в базу данных, были корректные и не вызывали ошибок при работе программы.

Но как же осуществить отправку данных клиенту, например, в браузерное приложение?

На этот счёт Microsoft предоставляет специализированные классы для работы с http запросами под названием “Контроллеры”.

Центральным звеном в архитектуре ASP.NET Core MVC является контроллер. При получении запроса система маршрутизации выбирает для обработки запроса нужный контроллер и передает ему данные запроса. Контроллер обрабатывает эти данные и посылает обратно результат обработки. В ASP.NET Core MVC контроллер представляет обычный класс на языке C#, который наследуется от абстрактного базового класса Microsoft.AspNetCore.Mvc.Controller.

Рассмотрим для примера контроллер, занимающийся отправкой и получением информации о запчастях.

Лист

38



Рис. 3.31 контроллер DetailController

Перед объявлением класса контроллера в квадратных скобках указывается атрибут [ApiController], чтобы компилятор понял, что перед ним класс контроллера. [Route(“путь”)] указывает путь к контроллеру внешним формирователям запросов. Класс DetailController наследуется от класса Controller, в котором указаны атрибуты, которыми могут обладать методы контроллера. Атрибутам являются, например: HttpGet(запрос получения), HttpPost(Запрос создания), HttpDelete(Запрос удаления), HttpPut(Запрос обновления); то есть реализуются все основные CRUD (create, read, update, delete) операции.

Данные, приходящие от пользователя, и классы сущностей, приходящие с базы данных, зачастую могут отличаться между собой, поэтому необходимо разделять данные, которые приходят с бд, от данных, которые приходят от клиента. Для этого и вводятся специальные классы, а именно DTO (Data Transfer Object) сущности.

DTO — это класс с данными, но без логики. Он используется для передачи данных между слоями приложения и между приложениями, для сериализации и аналогичных целей. Примером DTO является любой класс, который содержит только поля и свойства.

Лист

39

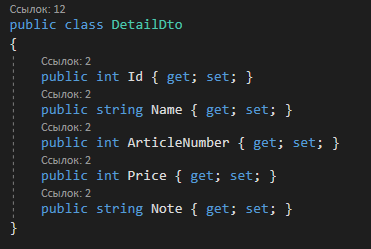


Рис. 3.32 DTO сущность DetailDto

В данном случае DTO сущность полностью дублирует сущность для базы данных, но очень часть бывают ситуации, когда необходимо отправить с сервера сущность, которая имеет дополнительные свойства, а может и вовсе не имеет каких то свойств.

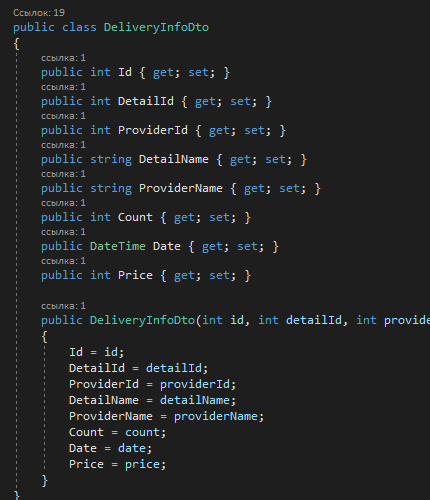


Рис. 3.33 DeliveryInfoDto, сильно отличающаяся от сущности Delivery

Для преобразования из DTO в Entity и наоборот, введены специальные классы, осуществляющие эту функцию.

Лист

40

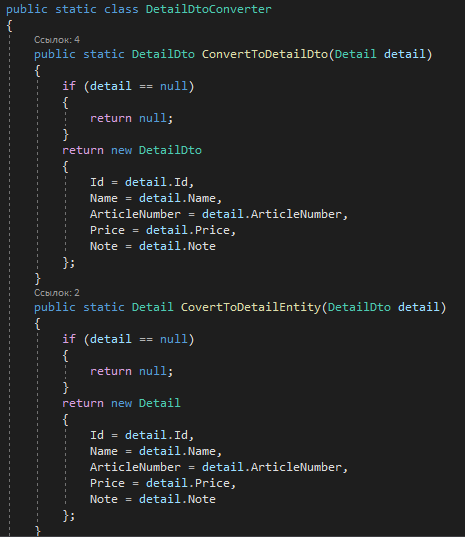


Рис. 3.34 преобразователь сущностей DetailDto и Detail

Таким образом, серверная часть представляет собой сложную структуру, осуществляющую взаимодействие пользователя с базой данных, через специализированные серверные запросы.

Контроллеры, сервисы и репозитории также, необходимо проинициализировать в классе Startup.

В метод ConfigureServices необходимо добавить следующие настройки.

Лист

41



Рис. 3.35 инициализация всех классов контроллеров

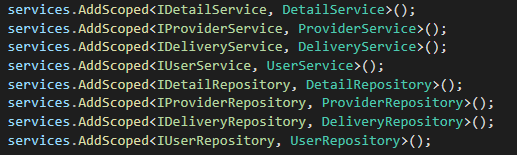


Рис. 3.36 инициализация всех репозиториев и сервисов.

**3.3 Клиентская часть приложения (FrontEnd)**

**3.3.1 Предварительная настройка серверной части**

Перед тем, как приступить к разработке веб сайта, необходимо в классе Startup настроить следущее:

В методе ConfigureService для моего курсового проекта прописывается:

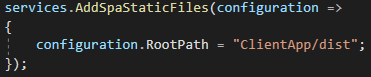


Рис. 3.37 брать файлы для отображения интерфейсной части приложения из папки “ClientApp/dist”

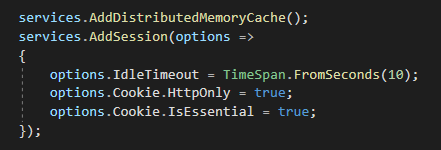


Рис. 3.38 добавить сессию

В методе Configure:

Лист

42

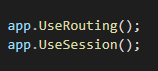


Рис. 3.39 использовать маршрутизацию и сессию

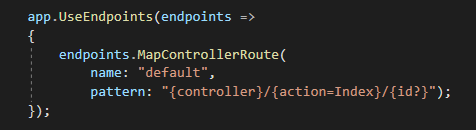


Рис. 3.40 добавить инициализацию путей для контроллеров

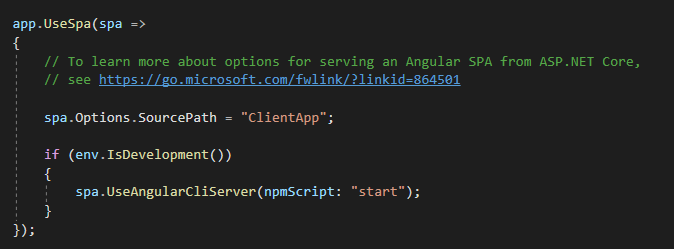


Рис. 3.41 настройка запуска клиентского приложения

Данные настройки позволяют серверу автоматически определять, где находятся необходимые файлы, для отображения веб сайта в браузере, а также позволяет вести динамическую разработку клиетской части приложения, не перезапуская сервер, при изменении веб интерфейса.

**3.3.2 Веб интерфейс приложения**

Раньше любое веб приложение работало по принципу многостраничного сайта (multi page application), т. е. для каждого раздела на сайт выгружалась отдельная страница с сервера, со всем контентом и данными. Самый главный минус данного подхода заключается в том, что для обновления контента на странице необходимо было снова выгружать всю страницу с сервера.

Во-первых, на выгрузку страницы требуется время, во-вторых по мере развития веб сайтов, количество выгружаемого контента выросло в разы, в результате страницы стали грузиться ещё дольше, в-третьих нагрузка на сервер осталась высокая, так как постоянно происходило обновление страниц. В результате был придуман другой способ представления сайта для пользователя. Такие компании как Google предложили загружать на браузер какую-то часть интерфейса или даже весь интерфейс целиком. Хоть это и занимало больше времени, но зато это открыло возможность мгновенно обновлять контент на странице, не выгружая её снова, а просто скачивать нужные данные. Это открыло возможности для модернизации пользовательского интерфейса. Теперь верстальщик сайтов, уже не дизайнер, а полноценный программист, который может прописывать свою логику для различных веб приложений. Так появился новый подход к разработке интерфейсной части, как “одностраничное приложение” (single page application). Писать на обычном JavaScript SPA приложения оказалось очень сложной задачей, поэтому была придумана специализированная программная платформа Node.js, основанная на движке V8, превращающая JavaScript из узкоспециализированного языка в язык общего назначения.

Лист

43

Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API, написанный на C++, подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. Node.js применяется преимущественно на сервере, выполняя роль веб-сервера, но есть возможность разрабатывать на Node.js и десктопные оконные приложения и даже программировать микроконтроллеры. В основе Node.js лежит событийно-ориентированное и асинхронное программирование с неблокирующим вводом/выводом. Так же это открыло возможности разработчикам писать свои фреймворки для написания SPA приложений, такие как Angular.js, React.js, Vue.js.

В данной курсовой работе мной был использован Angular.js, так как до этого был опыт использования данного фреймворка.

Angular – MVW-фреймворк для разработки качественных клиентских веб-приложений на JavaScript. Он создан и поддерживается в Google и предлагает взглянуть на будущее веба, на то, какие новые возможности и стандарты он готовит для нас.

Лист

44

MVW означает Model-View-Whatever (модель – вид – что угодно), то есть гибкость в выборе шаблонов проектирования при разработке приложений. Мы можем выбрать модели MVC (Model-View-Controller) или MVVM (Model-View-ViewModel).

Позиция AngularJS по работе с данными и другими инженерными концепциями отличается от таких фреймворков, как Backbone.js and Ember.js. Многие довольствуются уже известным нам HTML, а Angular самостоятельно его улучшает. Angular обновляет DOM при любых изменениях Модели, которая живёт себе в чистых Объектах JavaScript с целью связи с данными. Когда обновляется Модель, Angular обновляет Объекты, которые содержат актуальную информацию о состоянии приложения.

AngularJS считается одним из самых сложных фреймворков для разработки веб приложений, но соответственно и обладает гораздо большими возможностями в отличие от VueJS.

Рассмотрим структуру проекта Angular:

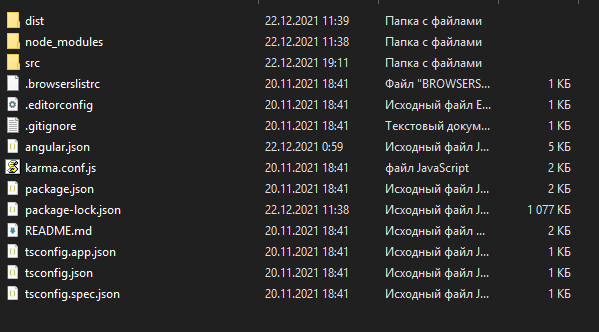


Рис. 3.42 структура проекта AngularJS

На рисунке видно, что в проекте присутствуют 3 папки и много конфигурационных файлов. В папке node\_modules хранятся библиотеки, необходимые для разработки приложения. В папке src находятся все написанные разработчиком модули для проекта. В папке dist хранятся файлы с компилированным js кодом, который понимает браузер.

Лист

45

В файле angular.json хранятся все основные настройки проекта.

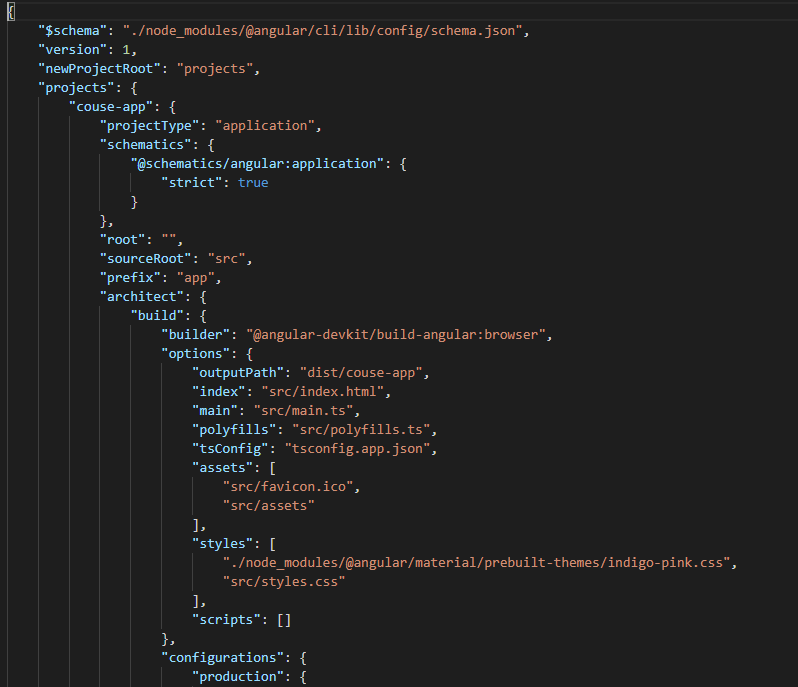


Рис. 3.43 пример angular.json

В папке src находятся написанные разработчиком файлы, а также все загружаемые ресурсы (картинки, видео, музыка) на веб сайт.

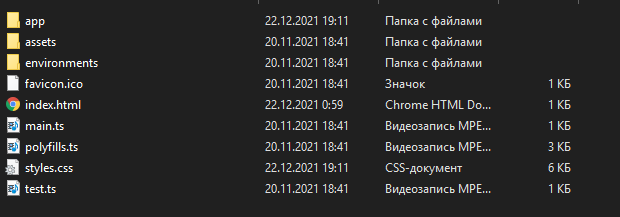


Рис. 3.44 папка src

Angular project представляет собой набор “компонентов”, с главным компонентом app.component, на котором отображаются остальные.

Лист

46

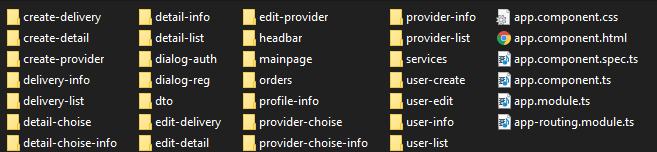


Рис. 3.45 компоненты AngularJS

Компонент (Angular component) - обособленная часть функционала со своей логикой, HTML-шаблоном и CSS-стилями.

Класс становится Angular компонентом, если его объявлению предшествует декоратор @Component() с объектом конфигурации.

Верстка компонента осуществляется на языке html и css, а вот логика разрабатывается с помощью специально разработанного компанией Google языка TypeScript, который потом компилирутся в JavaScript. TypeScript В целом очень схож с js, но отличается возможностью работы с библиотеками AngularJS.

Все компоненты прописываются в специальный файл app.module, для того, чтобы зарегистрировать компонент в проекте.

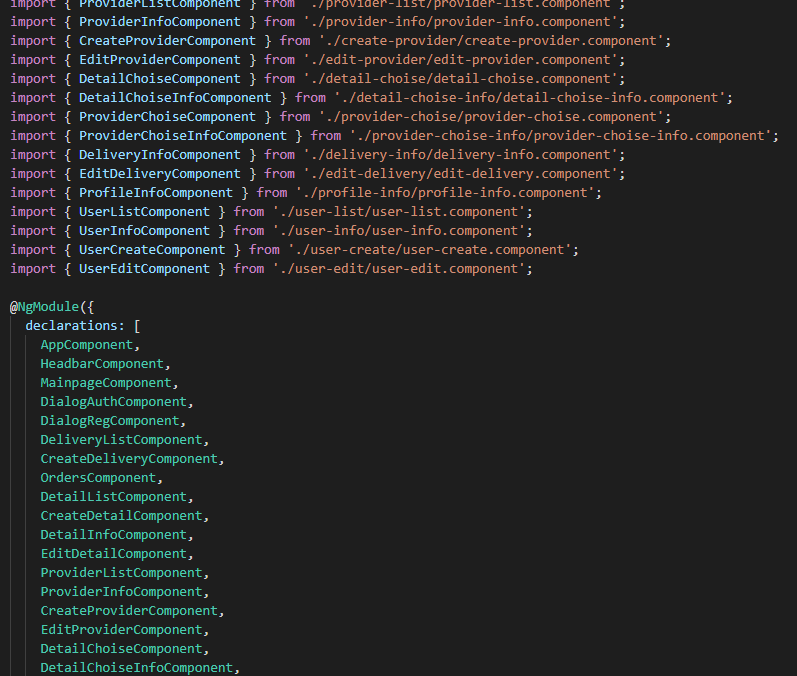


Рис. 3.46 пример app.module

Всего в проекте было использовано 24 компонентов.

Лист

47

Рассмотрим компонент по подробнее. Например, detail-list component.

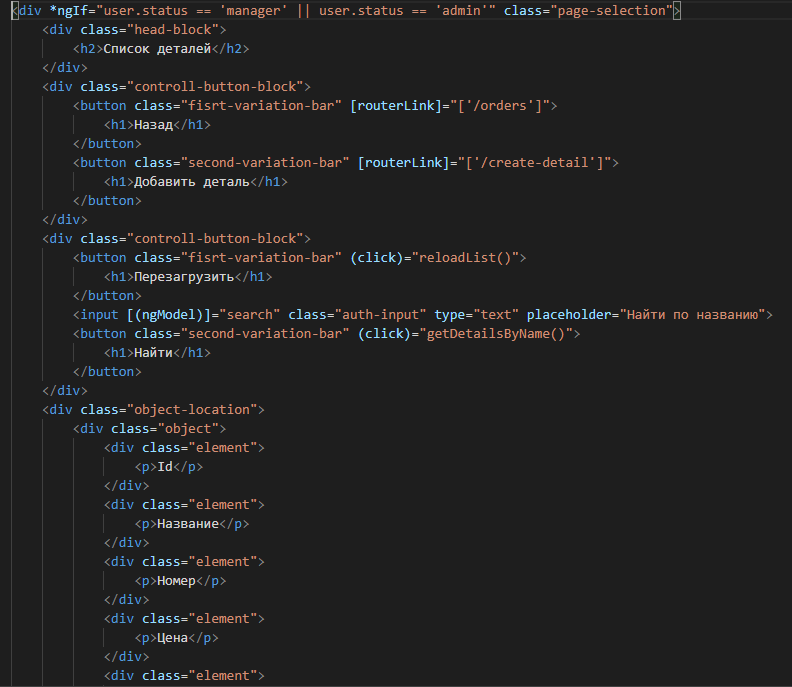


Рис. 3.47 detail-list.component.html

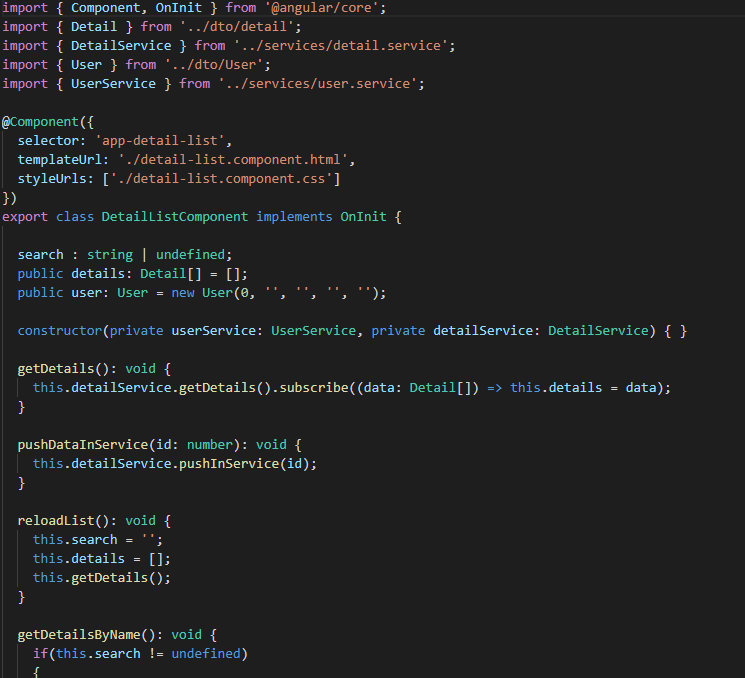


Рис. 3.48 detail-list.component.ts

Файл, представленный на рисунке 3.47 представляет собой html документ, но с некоторыми директивами, которые предоставляет AngularJS. Например, директива \*ngIf позволяет, в зависимости от значения определённой переменной, отображать нужные блоки страницы. Директива [/routerLink], позволяет переходить из одного компонента в другой (по сути смена страницы, но при этом выгружается не новая страница, а меняется содержимое текущей). Директива (click) обрабатывает событие нажатия на элемент. Хочется отметить директиву ngModel, аналог id=”название элемента” из js, она позволяет связать определённый тег с переменной.

Лист

48

Рисунок 3.48 является файлом с логикой для данного компонента. Сам компонент в системе Angular представляет собой класс, с определённым набором методов и свойств. На основе событийно-ориентированного программирования можно описать достаточно сложную работу компонента. По умолчанию любой компонент наследуется от класса OnInit, в результате чего и обладает методом ngOnInit, в котором выполняются определённые действия при вызове данного компонента.

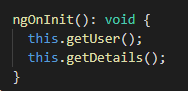


Рис. 3.49 метод ngOnInit()

Чтобы создать компонент можно воспользоваться консолью. Данная команда позволит создать компонент с определённым набором файлов и сразу зарегистрирует его в app.module.



Рис. 3.50 команда для создания компонента

Что в серверной части, что в интерфейсной части приложения необходимы классы, позволяющие перемещать определённый набор данных с клиента на сервер и обратно. Для этого созданы DTO сущности дублирующие их на сервере.

Лист

49

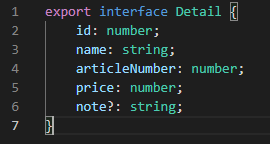


Рис. 3.51 сущность, дублирующая серверную DetailDto

Для отправки запросов на сервер, необходимы специальные сервисы.

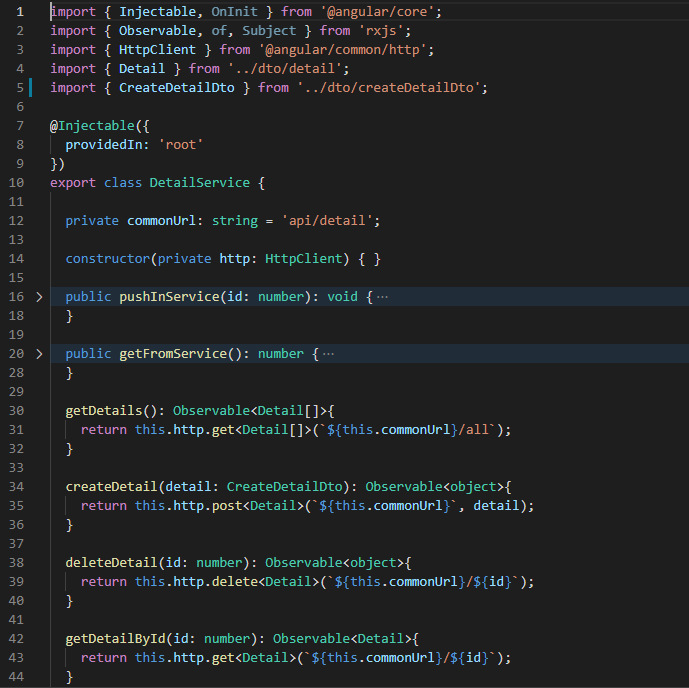


Рис. 3.52 сервис DetailService

Сервисы, обладают теми методами, которыми обладают контроллеры на сервере. С помощью специального инструмента HttpClient, позволяющего выполнять запросы на сервер, сервис получает или отправляет DTO сущности, которые потом обрабатывает сервер. Любой запрос, даже который идёт на отправку данных, всегда возвращает какое-либо значение, все эти значения фиксирует наблюдатель (Observable). За счёт данного инструмента все запросы на сервер возвращаются асинхронно, это позволяет пользователю не прерывать работу с веб приложением. Чтобы обработать результат действия наблюдателя, необходимо подписаться на него, в следующем примере рассмотрим, как работает подписка на Observable.

Лист

50



Рис. 3.53 обработка данных перед post запросом

Перед подготовкой к отправке данных на сервер, данные проверяются на корректность: идёт проверка на пустые значения и размер, введённых данных, а также проверяется, уникальный номер артикля для запчасти, и это запрос на сервер. Вызывается сервис, выбирается нужный метод, а затем идёт ключевое слово subscribe (подписка), таким способом идёт подписка на данного наблюдателя, который будет выполнять свою работу асинхронно. Поэтому обработать значения, возвращаемые наблюдателем, не получится, пока он не выполнит запрос. Для того, чтобы понять, что пришло с сервера, Angular предлагает продолжить писать функцию внутри подписки. Через лямбда-оператор можно посмотреть, какие данные вернул Observable, и уже исходя из условия, предпринять дальнейшие действия. В данном случае, если нет такого артикля в базе, вызвать post запрос прямо внутри лямбда-функции.

Лист

51

Таким образом сделаны запросы на сервер в данной курсовой работе.

Хотелось бы упомянуть ещё один инструмент AngularJS под названием Angular Material. Данное расширение содержит уже готовые решения для реализации проекта: диалоговые окна, теги, диаграммы, анимации. Мной были применены диалоговые окна регистрации и авторизации.

**3.4 Авторизация и регистрация**

Для авторизации и регистрации понадобилось создать ещё одну таблицу, пользователей, в которой будут храниться аккаунты.

В данной курсовой работе, необходимо сделать разный уровень доступа пользователей, а именно:

1) Обычный пользователь, который может только создавать поставки из имеющегося набора товаров и поставщиков.

2) Менеджер, может создавать не только поставки, но и добавлять новые товары и поставщиков.

3) Администратор, может заниматься работой предыдущих уровней, а также редактировать учётные записи пользователей (только он может менять уровни доступа).

Лист

52

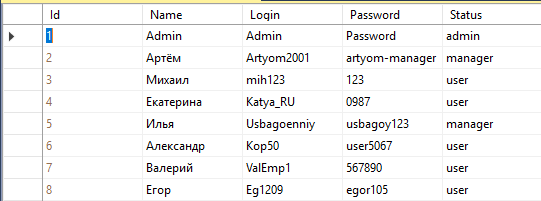


Рис. 3.54 Таблица User

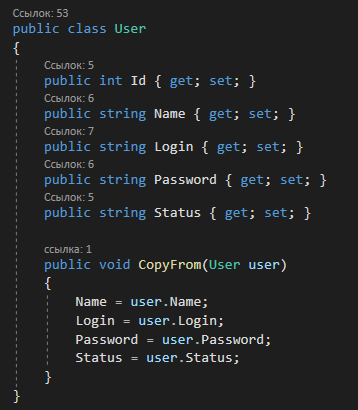
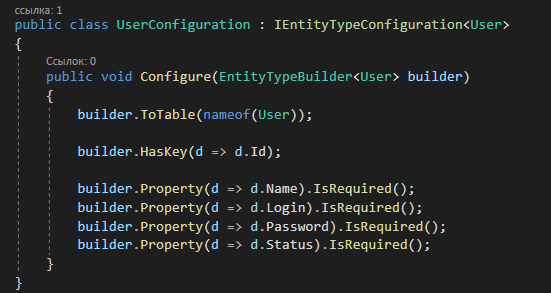


Рис. 3.55 сущность User



Лист

53

Рис. 3.56 конфигурация для таблицы User

Необходимы: репозиторий, сервис, контроллер для обработки регистрации, авторизации, а также для редактирования учётных записей других пользователей (функция администратора).

Рассмотрим, как обрабатывается авторизация на сервере:

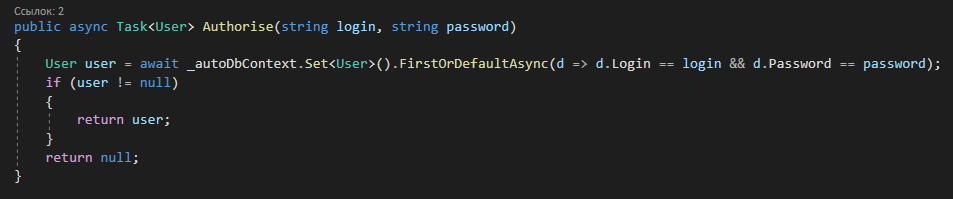


Рис. 3.57 метод авторизации в репозитории

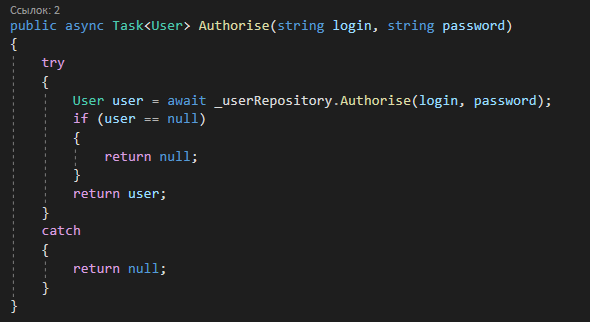
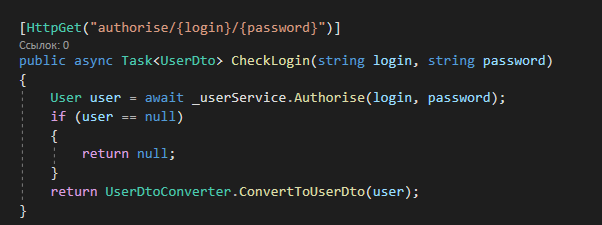


Рис. 3.58 метод авторизации в сервисе



Лист

54

Рис. 3.59 метод авторизации в контроллере

Осуществляется поиск на по логину и паролю в базе данных, результаты запроса проверяются сервисом, и после проверки отправляются контроллером в веб приложение.

Теперь рассмотрим, как происходит получение результата авторизации клиентской частью:

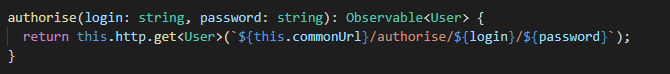


Рис. 3.60 метод авторизации в сервисе пользователей



Рис. 3.61 обработка результата авторизации компонентом

Авторизация делается по подобному принципу: отправка запроса на сервер, получение результата.

Лист

55

Регистрация выполняется по подобному принципу, но за 2 запроса. Сначала проверяется есть ли пользователь с данным логином на сервере, а потом уже идёт post запрос на создание новой записи.

**4. Интерфейс приложения**

Интерфейс приложения выполнен в строгом сером стиле, так как направлен в первую очередь, для работы в организации, как корпоративный сайт. Рассмотрим небольшой пример работы с приложением.

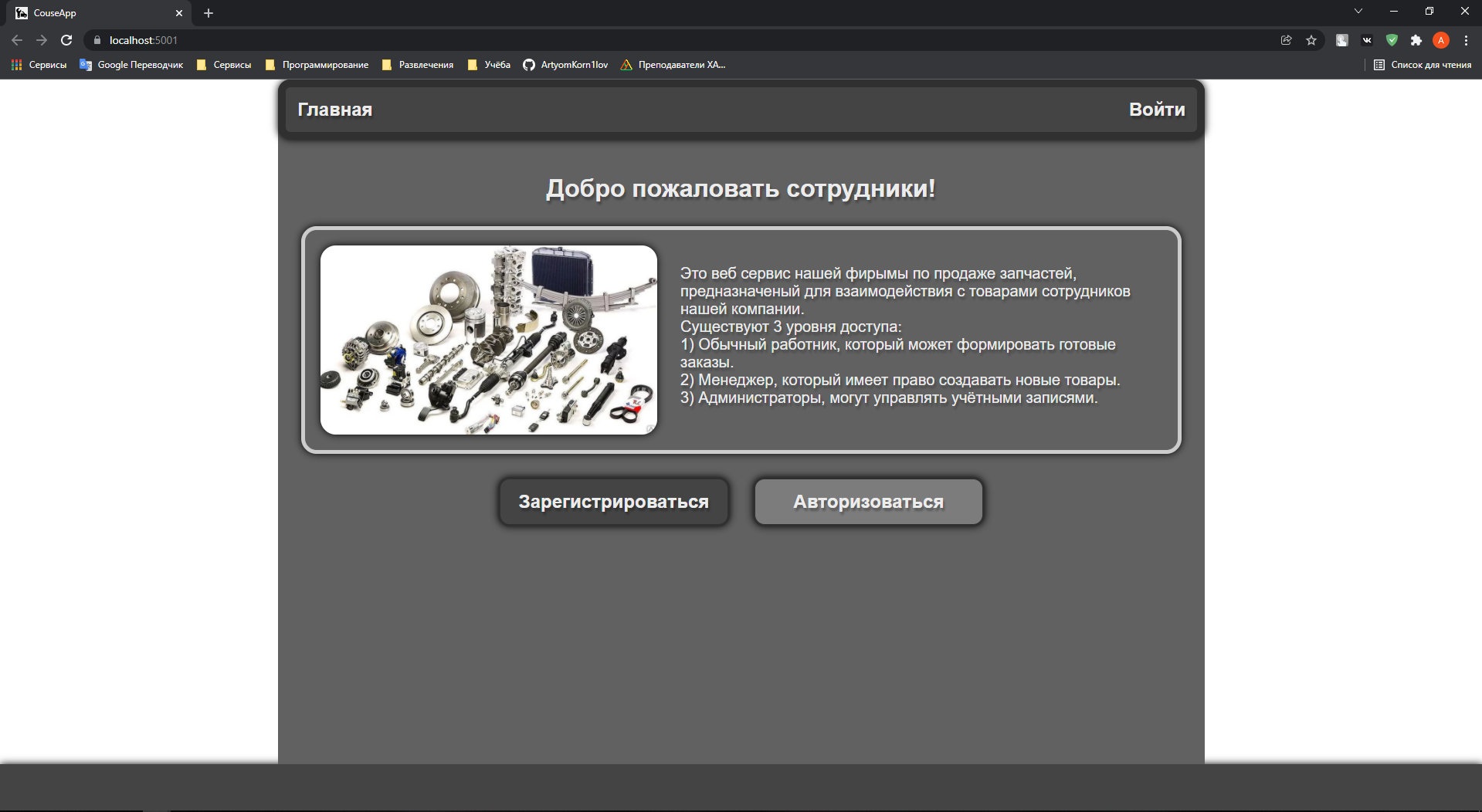
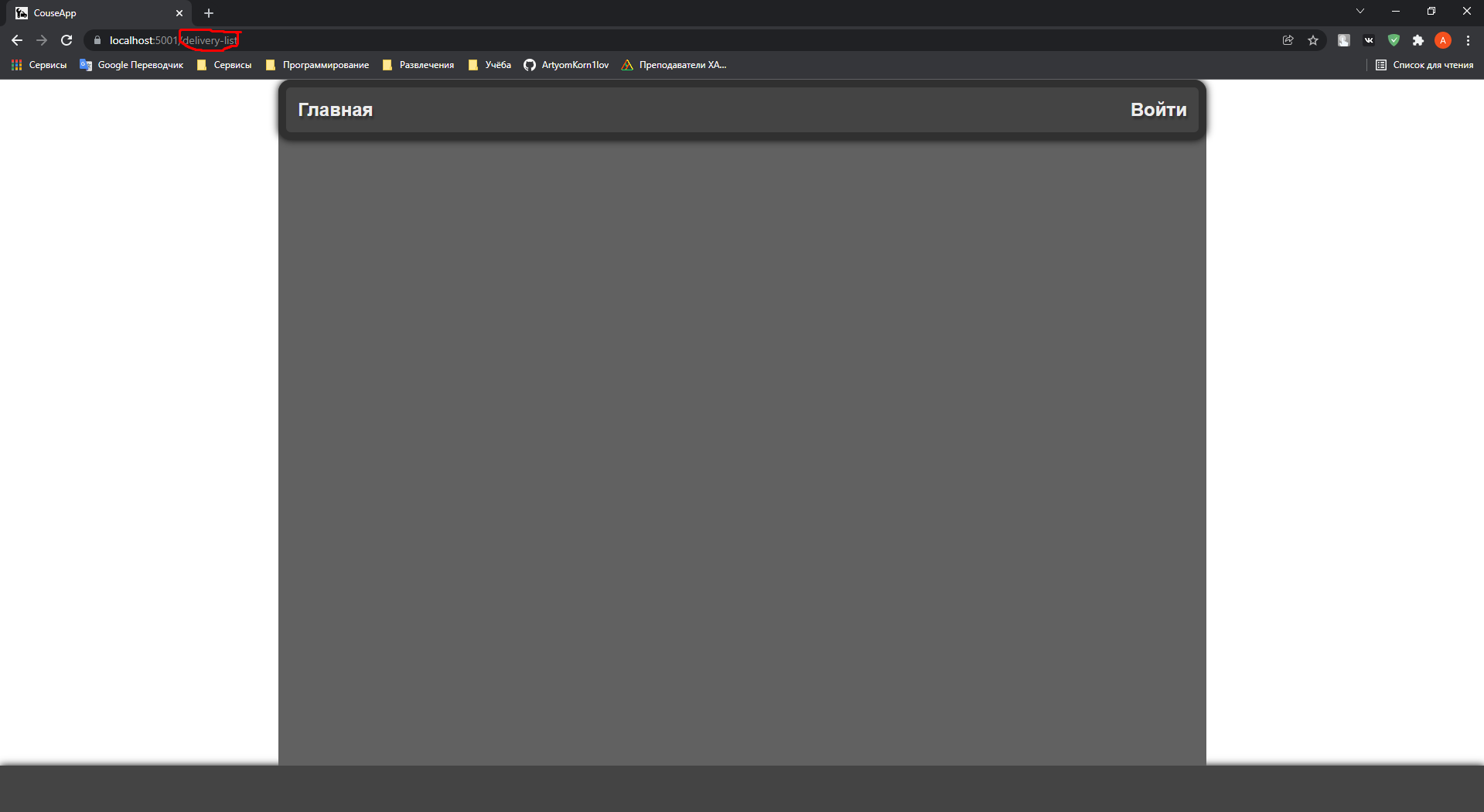


Рис. 4.1 главная страница веб сайта фирмы

Главная страница предоставляет возможность зарегистрироваться или авторизоваться, чтобы получить доступ к тем или иным ресурсам. При попытке, попасть на нужный url без нужного уровня доступа, приложение не допустит пользователя до контента.



Лист

56

Рис. 4.2 доступ к ресурсу заблокирован

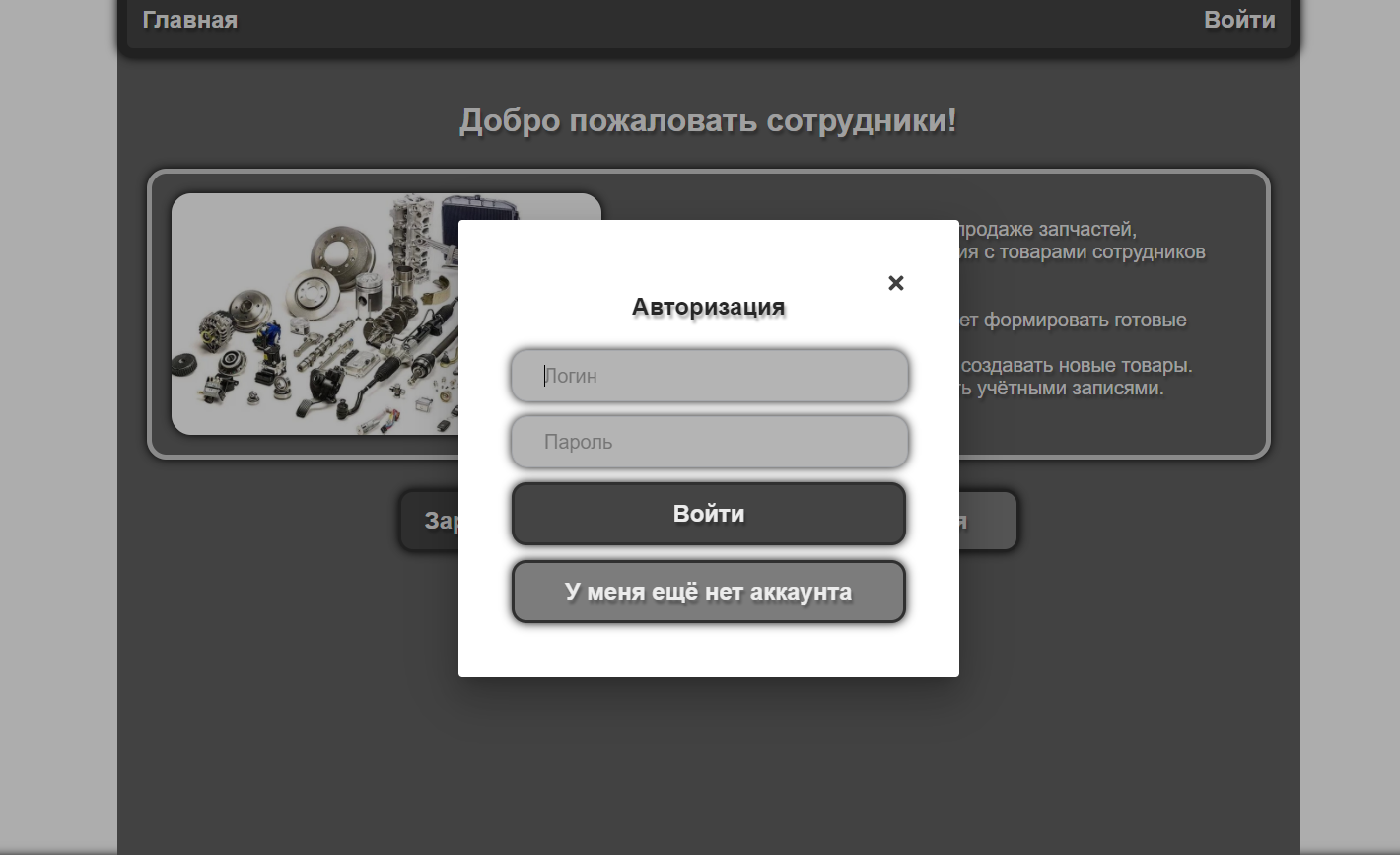


Рис. 4.3 окно авторизации

На рисунке 4.3 показана возможность авторизоваться, но для примера будет продемонстрирована регистрация под новым аккаунтом



Лист

57

Рис. 4.4 регистрация нового пользователя

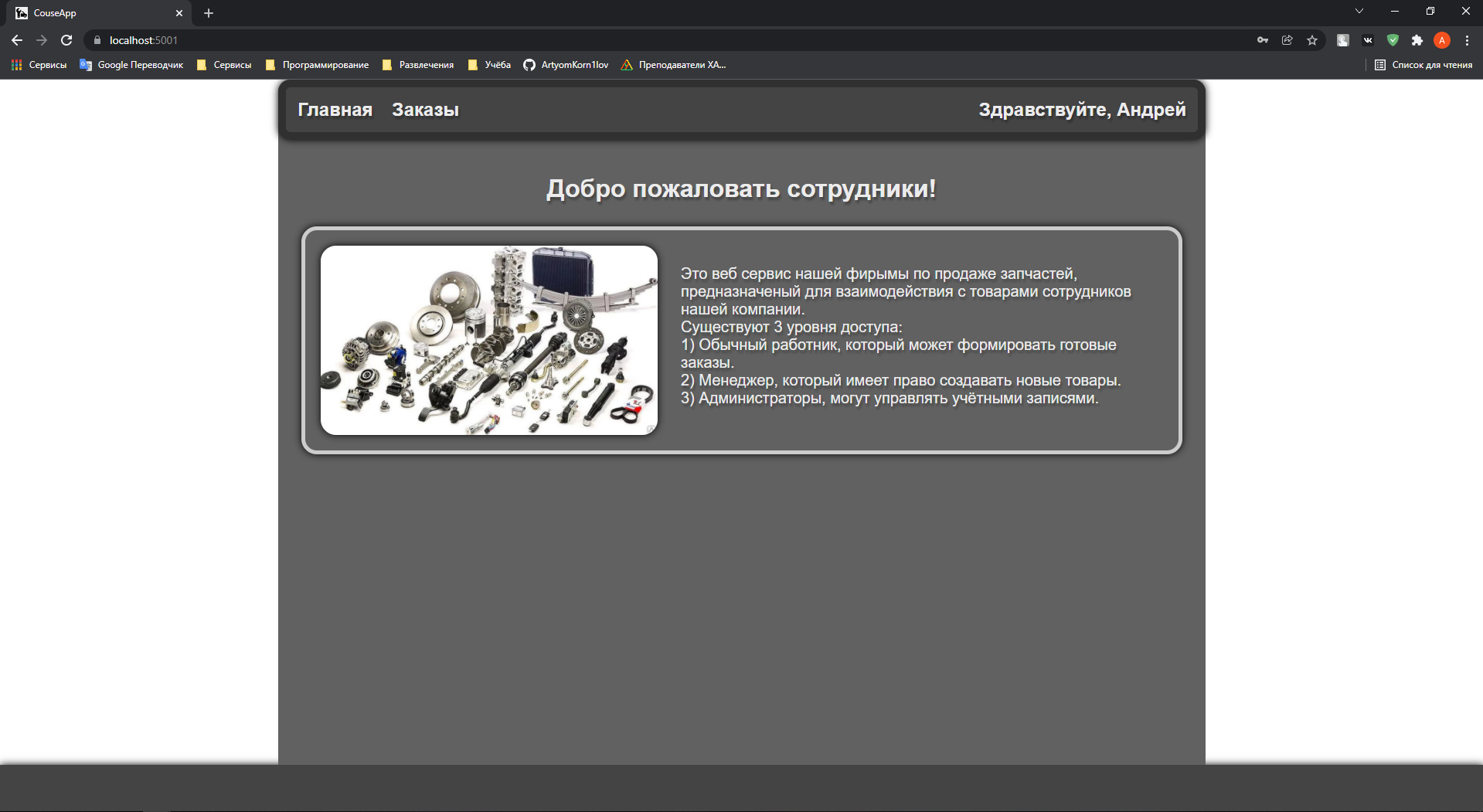


Рис. 4.5 пользователь зарегистрирован

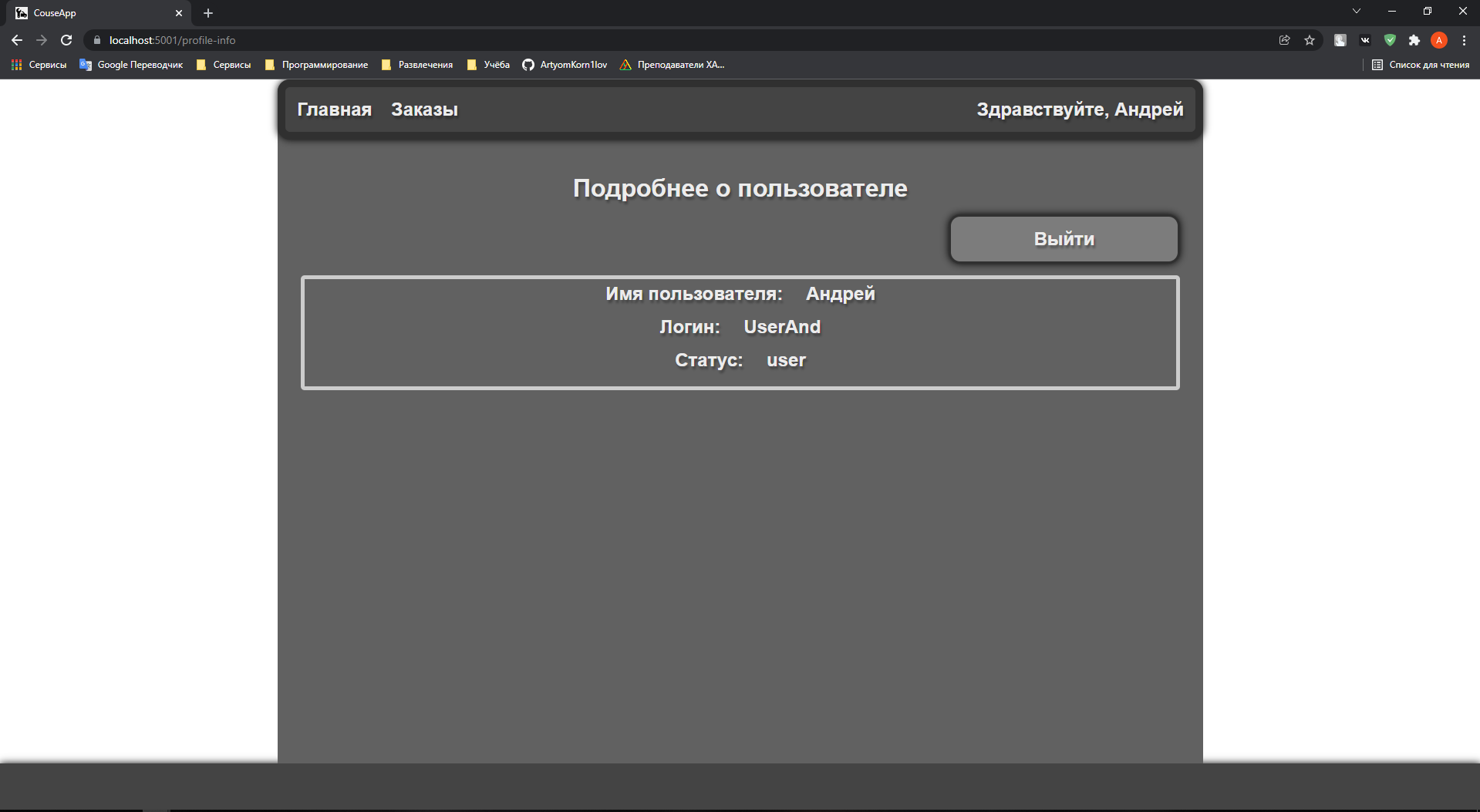


Рис. 4.6 информация о пользователе

Новый пользователь по умолчанию зарегистрирован со статусом user, это значит он должен иметь доступ только к разделу поставок, а к остальным не имеет прав доступа.

Лист

58

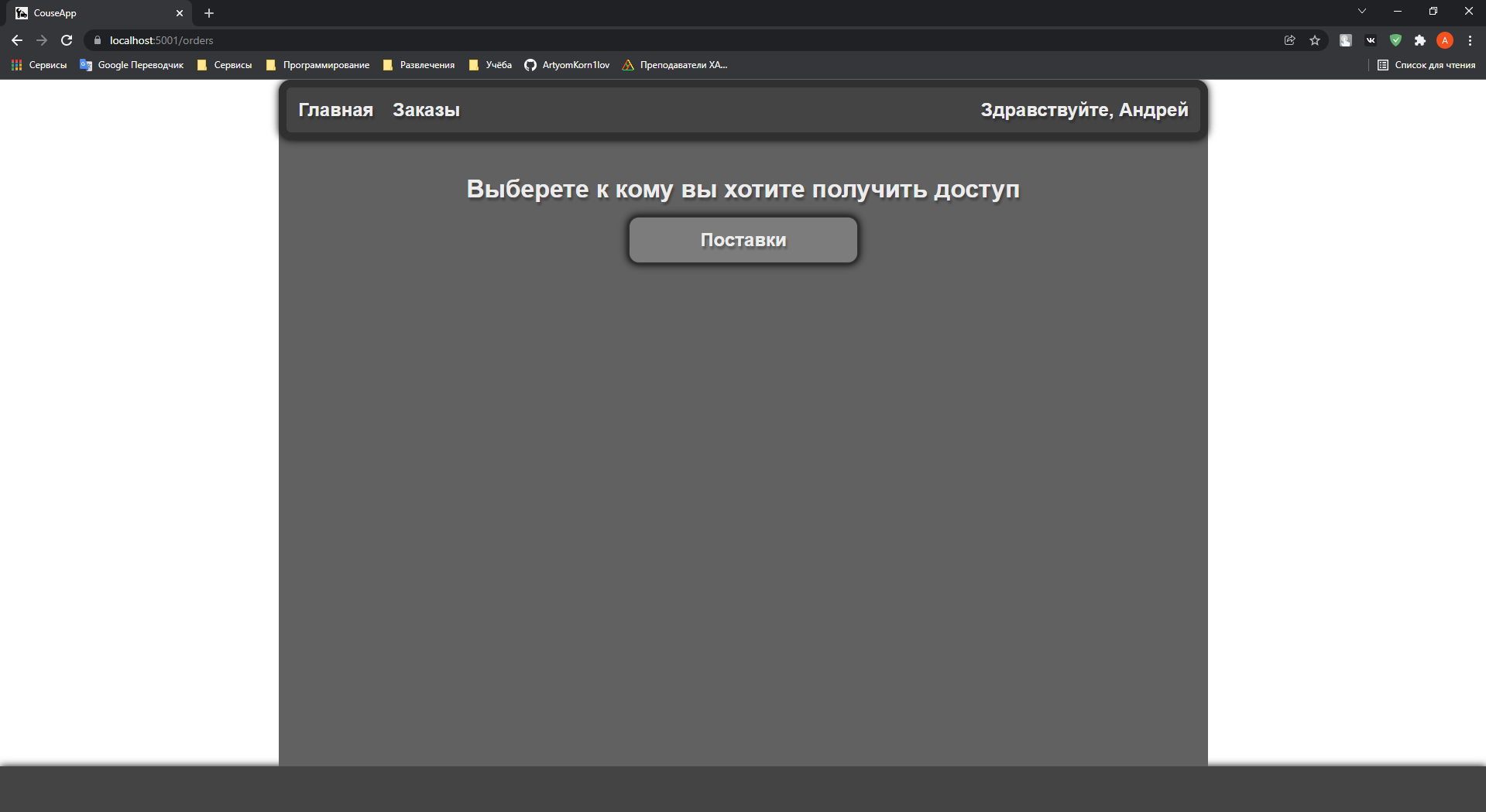


Рис. 4.7 пользователь со статусом user имеет доступ только к поставкам.

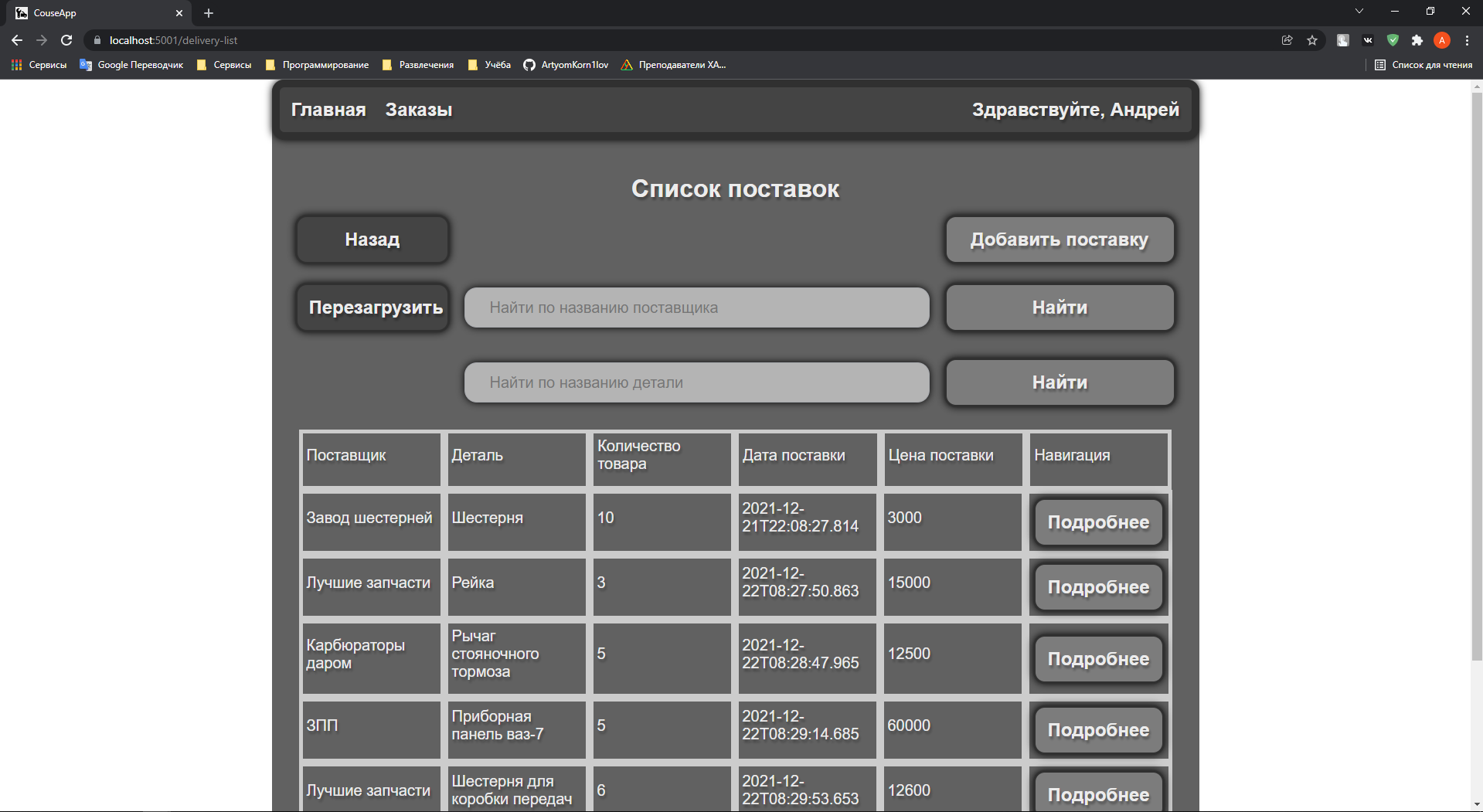


Рис. 4.8 список поставок, доступных пользователю

В данном примере был рассмотрен один из элементов работы веб приложения с пользователями, разграничивающий разные права доступа.

Лист

59

**5. Система контроля версий Git**

Что такое “система контроля версий” и почему это важно? Система контроля версий — это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии.

Git — одна из систем контроля версий. Предназначена, в основном, для работы распределенной команды разработчиков. То есть разработчики могут находиться в разных концах света и работать над одним проектом.

Нет Git используют не только разработчики, но и дизайнеры, писатели, редакторы, проектировщики, переводчики. GitHub часто используют hr специалисты и hr менеджеры для поиска успешных кандидатов на те или иные вакасии в различных областях. Для математического анализа успешности проектов.

Основная задача системы управление версий — это упрощение работы с потоками изменяющейся информации. Главной парадигмой системы управления версий является локализация данных каждого разработчика проекта. Каждый разработчик имеет на своей машине локальный репозиторий. В случае необходимости изменения отправляются из локального репозитория в удаленное хранилище в определенную ветку. И любой разработчик из распределенной команды может скачать новые изменения в проекте, чтобы продолжить совместную работу над проектом.

Почему мной используется система контроля версий? Во-первых, хранение репозитория проекта в облаке, можно очень удобно и быстро выгрузить проект на новую машину и продолжить работу. Во-вторых, мне важно было регистрировать любые изменения в курсовом проекте, так как данным способом можно очень легко понять, какая часть кода, которая была изменена, повлияла на появление программной ошибки.

Ну и в-третьих, необходимо научиться работать с репозиториями и изменениями содержимого проектов, хоть сейчас я и не работаю в команде, но в будущем знание принципов работы систем контроля версий, позволит мне легче освоиться в команде программистов.

Лист

60

Почему именно система контроля версий Git? На мой взгляд это самая удобная и широко используемая во всём мире система контроля версий.

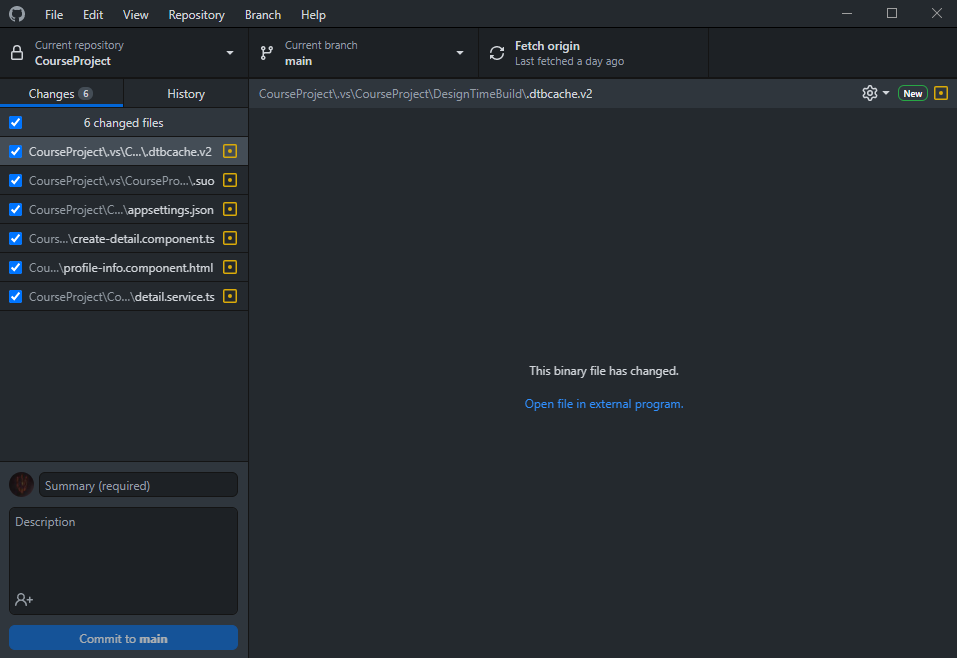
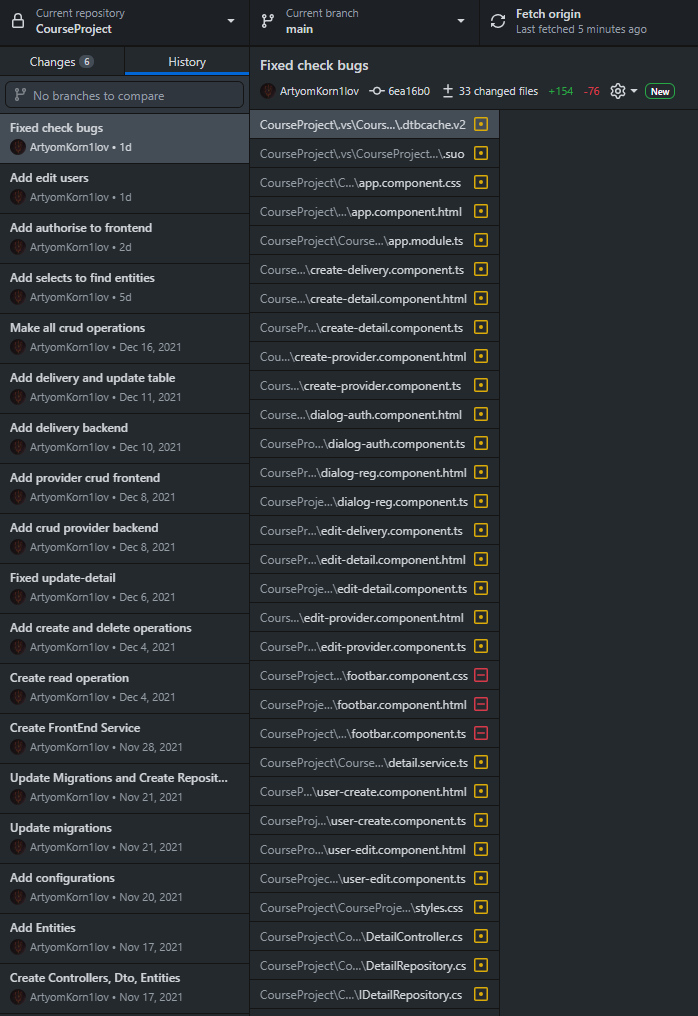


Рис. 5.1 интерфейс GitHub desktop

Для регистрации изменений проекта мной было использовано приложение GitHub decktop, так как данное приложение обладает очень удобным графическим интерфейсом и работает под windows 10.

Для того, чтобы зафиксировать изменения необходимо сделать commit (фиксацию). Коммит своим названием должен отражать, то что было сделано в проекте, быть понятным, чтобы в случае чего, можно было его проанализировать. Ниже приведёт список коммитов, созданных в результате разработки приложения для курсовой работы.



Лист

61

Рис. 5.2 коммиты, зафиксированные в рамках курсового проекта

**Заключение**

Лист

62

В процессе создания курсового проекта был создано веб приложение для управления базой данных для «Фирмы по продаже запчастей».

В данной курсовой работе разработана структура таблиц БД. Продемонстрировано, что таблицы БД находятся в 3 НФ.

Итогом работы является выпуск программного продукта, выполняющего работу с базой данных (добавление, удаление, изменение записей в таблицах), программа реализована в виде веб сайта, представляющего собой: браузерный интерфейс и веб сервер, который обрабатывает запросы с клиентской части и отправляет запросы в базу данных.

Рекомендуется размещение БД автоматизированной системы на сервере фирмы, что обеспечит совместный доступ к общим ресурсам всем сотрудникам.

**Список литературы**

Лист

63

1. <https://docs.microsoft.com/ru-ru/dotnet/csharp/>

2. <https://metanit.com/sharp/aspnet5/>

3. <https://metanit.com/sql/sqlserver/>

4. <https://metanit.com/sharp/articles/mvc/11.php>

5. <https://metanit.com/sharp/mvc5/23.3.php>

6. <https://metanit.com/sharp/aspnet5/5.1.php>

7. <https://metanit.com/web/angular2/>

8. <https://angular.io/guide/what-is-angular>

9. <https://material.angular.io/guide/getting-started>

10. <https://git-scm.com/docs>