МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет
имени К.И.Сатпаева

Институт автоматики и информационных технологий

Кафедра «Программная инженерия»

Насаналиев Самат Бекболұлы

Разработка интернет-магазина с использованием Angular Framework

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту

5В070400 – Вычислительная техника и программное обеспечение

Алматы 2022

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет имени К.И.Сатпаева

Институт автоматики и информационных технологий

Кафедра «Программная инженерия»

ДОПУЩЕН К ЗАЩИТЕ
Заведующей кафедрой ПИ
канд. физ-мат. наук, профессор
_____ Молдагулова А.Н.
«19» 05 2022 год

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту

На тему: Разработка интернет-магазина с использованием Angular Framework

по специальности 5В070400 – Вычислительная техника и программное обеспечение

Выполнил Насаналиев С.Б

Рецензент
Профессор, к.ф.-м.н, PhD
_____Акжалова А.Ж
"19" 05 2022 г.

Научный руководитель
канд. физ-мат. наук, профессор
_____ Молдагулова А.Н.
"19" 05 2022 г.

Алматы 2022

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет имени К.И.Сатпаева

Институт автоматики и информационных технологий

Кафедра "Программная инженерия"

5В070400 – Вычислительная техника и программное обеспечение

УТВЕРЖДАЮ
Заведующая кафедрой ПИ
канд. физ-мат. наук, профессор
_____ Молдагулова А.Н.
«20» 05 2022 г.

ЗАДАНИЕ
на выполнение дипломного проекта

Обучающемуся: *Насаналиеву Самату Бекболұлы*
Тема: *Разработка интернет-магазина с использованием Angular Framework*
Утверждена *приказом проректора по академической работе* №489-П/О от "24" 12 20 21 *г.*
Срок сдачи законченного проекта "24" 05 20 22 *г.*
Исходные данные к дипломному проекту: *Разработка Frontend и Backend части веб-платформы*
Перечень подлежащих разработке в дипломном проекте вопросов:

a) *создание стека технологий для компоновки;*
b) *разработка front end части веб-сайта;*
c) *разработка back end части веб-сайта;*
d) *ведение системы контроля версий.*

Перечень графического материала (с точным указанием обязательных чертежей):*предоставлены* 17 *слайда презентации.*
Рекомендуемая основная литература: *из* 27 *наименований.*

ГРАФИК

подготовки дипломного проекта

Наименование разделов, перечень разрабатываемых вопросов	**Сроки предоставления научному руководителю и консультантам**	**Примечание**
1.Анализ предметной области, разработка технического задания	15.01.2022	выполнено
2.Выбор программных пакетов	20.01.2022	выполнено
3.Разработка UML - диаграмм	25.01.2022	выполнено
4.Анализ методологий разработки	30.01.2022	выполнено
5.Анализ и изучение используемых инструментов	15.02.2022	выполнено
6. Создание, отладка связей базовой проектной части	27.02.2022	выполнено
7. Разработка MVP части проекта	04.03.2022	выполнено
8. Написание компонентов frontend	10.03.2022	выполнено
9. Формирование БД	25.03.2022	выполнено
10. Разработка backend части	01.04.2022	выполнено
11. Отладка и дебаг, оптимизация	15.04.2022	выполнено
12. Написание документальной части дипломного проекта	28.04.2022	выполнено

Подписи

Консультантов и нормоконтролера на законченный дипломный проект с указанием относящихся к ним разделов проекта

Наименование разделов	**Консультанты, И.О.Ф (уч. степень, звание)**	**Дата подписания**	**Подпись**
Нормоконтролер	Жекамбаева М.Н. Доктор Ph.D., профессор	18.05.22	
Программное обеспечение	Марғұлан Қ. Магистр техн.наук, лектор	18.05.22г	

Научный руководитель _____ Молдагулова А.Н

Задание принял к исполнению обучающийся _____ Насаналиев С.Б

Дата “17” 11 2021 г.

АННОТАЦИЯ

Данный дипломный проект посвящен разработке интернет-магазина на веб-платформе с использованием Angular Framework для Frontend части. Ориентированный для показа возможностей фреймворка и использовании его инструментария. Проект показывает удобство и ускоренную разработку frontend части сайта, что является заслугой Angular Framework. Проект состоит из двух частей это frontend и backend части, в сборке веб-платформы использовались такие инструменты как: Angular Framework, Material Design, TFA Google Authenticator, Node.js Express App, XAMPP, MySqli, Apache, PhpMyAdmin. В совокупности всех инструментов дающую полноценную веб-платформу. В дипломном проекте подробно расписан весь инструментарий Angular Framework. Обеспечивающий базовый функционал клиент-серверного сайта, компонентной реализации с использованием парадигм объектно-ориентированного программирования, предоставления пользовательского интерфейса, взаимодействия с сервером и базой данных.

АННОТАЦИЯ

Бұл дипломдық жоба frontend бөлігі үшін Angular Framework көмегімен веб-платформада интернет-дүкен құруға арналған. Жақтаудың мүмкіндіктерін көрсетуге және оның құралдарын пайдалануға бағытталған. Жоба angular Framework-тің еңбегі болып табылатын сайттың frontend бөлігінің ыңғайлылығы мен жедел дамуын көрсетеді. Жоба екі бөліктен тұрады: frontend және backend бөліктері, веб-платформаны құру кезінде бұрыштық жақтау, Материалдық дизайн, TFA Google Authenticator, Node сияқты құралдар қолданылды.js Express App, XAMPP, MySqli, Apache, PhpMyAdmin. Толық веб-платформа беретін барлық құралдардың жиынтығы. Дипломдық жобада бұрыштық шеңбердің барлық құралдары егжей-тегжейлі сипатталған. Клиент-серверлік сайттың негізгі функционалдығын, объектіге бағытталған бағдарламалау парадигмаларын қолдана отырып, компонентті іске асыруды, пайдаланушы интерфейсін, сервермен және мәліметтер базасымен өзара әрекеттесуді қамтамасыз етеді.

ANNOTATION

This graduation project is dedicated to the development of an online store on a web platform using the Angular Framework for the Frontend part. Oriented to show the capabilities of the framework and the use of its tools. The project shows the convenience and accelerated development of the frontend part of the site, which is the merit of the Angular Framework. The project consists of two parts: frontend and backend parts, tools such as Angular Framework, Material Design, TFA Google Authenticator, Node were used in the assembly of the web platform.js Express App, XAMPP, MySqli, Apache, PhpMyAdmin. Combined with all the tools, it provides a full-fledged web platform. The entire Angular Framework toolkit is described in detail in the graduation project. Providing the basic functionality of a client-server site, component implementation using object-oriented programming paradigms, providing a user interface, interacting with the server and database.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Интернет-магазин или веб-платформы используются создателями для онлайн покупки, продажи чего-либо, что значительно увеличивает доходность владельца.

С приходом web 2.0, предприниматели активно начали использовать интернет в виде инструмента для привлечения трафика, заработка на онлайн продажах. Что привело к росту спроса одностраничных сайтов, маркетплейсов, различных сервисов на веб-платформах.

Интернет стал неотъемлемой частью ведения бизнеса, сопутствуя прогрессу web технологий, сейчас практический любой бизнес имеет свою собственную платформу, решающая различного рода задачи. Исходя из этого выросла потребность в создании web сервисов с передовыми технологиями. Для этого мы разберем проект создания интернет-магазина на основе Angular Framework для frontend части, входящий в топ 3 фреймворков.

Frontend часть является главнейшей составляющей для любого сайта, так как это место взаимодействия с пользователем и для его создания должны использоваться наилучшие инструменты. На данном фреймворке написаны такие популярные сайты как: Gmail, Forbes, PayPal, YouTube, Web Telegram.

Интернет-магазины обладают многими возможностями, что подтверждается на статистике, которая прогнозирует, 22% процента всех мировых розничных продаж к 2023 году будут проходить с помощью электронной торговли, это около 740 миллионов долларов для США.

При развитии AR&VR технологий, появится возможность визуального и физического контакта с выбранным товаром на маркетплейсе, что значительно увеличит спрос на интернет-магазины, что в следствии еще сильнее ускорит развитие web технологий.

Проект в целом представляет видение B2C, фреймворк относится к SPA, то есть одностраничное приложение, где данные обновляются на стороне клиента, состоящий из отдельный компонентов, которые могут быть обновлены, удалены, заменены без перезагрузки всей страницы, что в следствии дает большую пропускную способность и не загружает внешние файлы.

1 Исследовательский раздел

1.1 Цель разработки

Целью данной дипломной работы является разработка интернет-магазина используя Angular Framework, для backend части было принято решение использовать Node.js Express App, в качестве базы данных MySqli, для веб сервера Apache, для развёртки на локальном сервере использовал XAMPP. В качестве продажных товаров использовать готовый кластер с товарами.

Проект должен показать функциональность инструментария фреймворка, способ объединения frontend части с backend и какие при этом можно использовать инструменты, преимущества и недостатки фреймворка, так же практическая демонстрация легкости разработки frontend части на angular framework.

1.2 Термины и сокращения

В ниже приведенной таблице 1.1 описаны все термины, сокращения, которые были использованы при написании.

Таблица 1.1 - Терминология

Термины и сокращения	Определение
Web 2.0	Эпоха интернет, при котором контент создается самими пользователями.
Frontend	Визуальная часть сайта, все что может видеть, взаимодействовать пользователь
Backend	Функционал, внутренняя логика и работа сайта на стороне сервера
MVP	Minimal Viable Product. Минимально жизнеспособный продукт
AR&VR	Технология дополненной реальности, позволяющая взаимодействовать с объектами виртуальной реальности, интегрирование объектов из реальности в режиме реального времени

API	Application program interfaceПрограммный интерфейс приложения
IDE	Среда разработки
ES12	Стандарт ECMAScript от 2020 года
БД	База данных
СУРБД	Система управления реляционными базами данных
SPA	Одностраничное приложение
DI	Dependency injection – зависимость
DOM	Document object model – объектная модель документа HTML
RestAPI	Общие принципы организации приложения с сервером, посредством протокола HTTP
Primary Key	Первичный ключ, уникальный идентификатор, используется для взаимосвязи таблиц

1.3 Предметная область

К базовым частям frontend относятся HTML, CSS, JavaScript.

HTML – язык гипертекстовой разметки, использующаяся для разметки структуры веб-страницы, путем получения файлов с сервера и построения каркаса на который накладываются различные стили. Каркас строится из блоков которые определяют семантику структуры, такие как расположения блока, заголовки, ссылок, текста, списков и других элементов. Элементы записываются с помощью тегов и могут включать в себя другие тэги, в качестве подэлементов.

CSS – язык таблиц стилей, применяется для стилизации и описания HTML элементов. Позволяет гибко настраивать элементы, макетирование, стилизацию по шрифту и цветам, что позволяет тонко настроить визуализацию элементов.

JavaScript – высокоуровневый, объектно-ориентированный, с динамической типизацией интерпретируемый язык программирования спецификации ECMAScript, стандарт ES12. Предназначен для взаимодействия, управления событиями на веб-странице, динамически обновлять данные, управлять мультимедиа, анимировать и многое другое.

TypeScript – расширенная версия JavaScript, обладающий статистической типизацией данных, поддержка IDE используемая для упрощения крупномасштабных проектов, имеет обратную совместимость с JavaScript что

позволяет производить преобразования. Используется в Angular Framework, основной язык проекта.

Framework – является предварительно написанным кодом который диктует программисту как он должен создавать проект и изначально включает в себя возможности архитектурного решения проекта в процессе разработки, отладки, расширяемости, улучшают работоспособность рабочего процесса опираясь на передовые технологии разработки, таким образом берет на себя большую часть автоматизации разработки проекта. Так же достоинством является что множество современных решений от экспертных программистов, включены в последние обновления, тем самым фреймворки значительно облегают создание крупномасштабного проекта.

1.4 Понятие веб-платформы

Веб-платформа – сайт, работающий в интерактивном режиме и представляющий собой стек API, позволяющий взаимодействовать с элементами страницы, такой сайт тесно взаимосвязан с сервером, так как зачастую все действия происходят с манипуляцией данных на сервере, так же относится к web 2.0 и могут содержать обширный функционал, к примеру кроссплатформенность.

Данный проект является e-commerce маркетплейсом, позволяющий использовать систему электронной торговли, в виде покупки товара на онлайн интернет-магазине. Маркетплейс является онлайн-интернет магазином где пользователи могут размещать, продавать, покупать товары. В качестве примера можно привести пример: Каспи Магазин, Wildberries, OLX, eBay. В настоящий момент с развитием web 3.0, появляются децентрализованные маркетплейсы, что показывает прогресс веб-платформ.

1.5 Современные методы разработки веб-платформ

Существует большое количество различных способов создания веб-платформ, но все способы базируются на frontend и backend, это основа для любого сайта.

Рассмотрим современные технологии для разработки frontend. Самым простым вариантом создания frontend будет API основанное на HTML, CSS, JavaScript. К более продвинутым способам можно отнести фреймворки такие как:ASP.NET, Ruby on Rails, Angular, Vue.js и библиотеку React.js. Продвинутые способы основаны на C#/Java/JavaScript и включают в себя улучшенный функционал, готовые решения.

Для стилизации так же есть Bootstrap, Semantic UI, Foundation, Pure By Yahoo, Uikit, Material Design. Они так же включают в себя готовые решения и продвинутый функционал. Использование CMS – готовых шаблонов основанных на WordPress, Drupal, Joomla или сайтов конструкторов Wix, Weebly, uKit, что значительно ускоряет создание сайта, очень полезны людям которые не сильно разбираются в программировании и создании сайтов самостоятельно.

В настоящее время очень популярны фреймворки, они позволяют структурированно создать крупномасштабные проекты, с возможностью дальнейшего расширения, но так же не стоит забывать про CMS, которые могут в течении пару часов создать небольшую веб-платформу.

Современные методы разработки backend так же можно разделить на простой и более продвинутый вариант, к простому можно отнести php с веб сервером Nginx или Apache, с БД Oracle, MySql, MariaDB. К продвинутым способам относится: Yii, CodeIgniter, Symfony, Laravel, Phalcon PHP на языке PHP, так же Flask, Django, Web2py, Tornado на языке Python и ASP.Net Core с использованием C#. Для базы данных можно использовать PostgreSql, MySql и его расширения, Teradata. В качестве веб сервера Nginx, Apache, Node.js.

Для написания кода для frontend и backend используют текстовые редакторы для небольших сайтов, такие как Sublime, NotePad++, Brackets, но для больших проектов нужны серьезные инструменты в виде различных IDE, которые могут быть ориентированы под конкретную задачу.

IDE – среда разработки, имеет большое количество функций, которые облегчают написание, тестирование, отладку, сборку проекта. Функции по интеграции инструментов языка, рефакторинг, подсказки, автозаполнение, подключение к сервисам контроля версий, подключение плагинов, компиляция различных языков и скриптов в одном проекте, smart code navigation. Существуют ряд узконаправленных IDE, в зависимости от поставленных задач. Популярные IDE: Visual Studio/Code, WebStorm, PyCharm, PhpStorm, Eclipse, Atom.

2 Технологический раздел

Данный раздел включает в себя описание стека технологий, которые использовались при создании работоспособного проекта, общее понимание для каких задач использовались и какие имеют особенности работы.

2.1 Angular Framework

Angular framework – открытая и свободная платформа для разработки SPA веб приложений, с использованием языка TypeScript, разрабатываемая командой из компании Google, а также сообществом разработчиков из различных компаний. Основными компонентом фреймворка является NgModule, который собирает связный код в функциональный набор, так же предоставляет двустороннее связывание, которое может динамически менять данные в одном месте при изменении данных модели в другом месте.

Компоненты представляют собой набор элементов экрана, которые фреймворк может манипулировать в зависимости от логики компонента, содержат приписку components в названии и содержат декоратор @Component который идентифицирует класс непосредственно и предоставляет шаблон и связанные с ним метаданные, относящиеся к конкретному компоненту. Компоненты могут использовать службы, предоставляющие какую-либо функциональность, но не являющиеся частью компонента, для этого используются зависимости, которые добавляют модульность, расширяемость, эффективность, так же компоненты можно рассматривать как отдельные небольшие части интерфейса. Модули, службы, компоненты — это классы, которые используют декораторы, те в свою очередь отмечают свой тип и дают метаданные которые сообщают как фреймворку их использовать. Метаданные для класса сервиса предоставляют информацию, необходимую фреймворку, чтобы сделать ее доступной для компонентов посредством внедрения зависимостей DI.

Каждое приложение Angular имеет корневой модуль, условно называемый AppModule, который обеспечивает механизм начальной загрузки, запускающий приложение. NgModules могут импортировать функциональность из других NgModules и позволяют экспортировать свои собственные функции и использовать их другими NgModules, так же один корневой компонент, который соединяет иерархию компонентов с объектной моделью DOM. Что бы использовать службу маршрутизации мы импортируем NgModule RouterModule.

Модуль Ng это класс, и отмечается декоратором @NgModule, описывает как скомпилировать шаблон компонента из метаданных и создать инжектор. Так же организация кода в отдельные функциональные модули помогает в управлении разработкой сложных приложений и в проектировании для повторного использования.

Так же есть сервисы файлы, которые содержат приписку service и имеют декоратор @Injectable, сервисы нужны для объединения логики или данных через DI между компонентами, что позволяет сохранить компактность кода, и повысить эффективность.

Фреймворк использует binding syntax, это позволяет получать доступ к данным полей свойств внутри HTML тегов, данные могут браться напрямую у компонентов через зависимости, либо с локального хранилища, так же имеются свои индивидуальные атрибуты применяемы к HTML тегам, что создает связывание с компонентом, это упрощает создание пользовательских интерфейсов с большим количеством интерактивных частей и упрощает процесс разработки. Фреймворк считается компонентным приложением, что позволяет большую возможность повторного использования компонентов и сервисов.

Observable – обычно используются для асинхронного программирования и выполнения нескольких значений одновременно, поддерживают передачу данных между частями проекта. Определяет функцию для публикации значений, но она не выполняется до тех пор, пока потребитель не подпишется (subscribe()) на нее. Затем подписанный потребитель будет получает уведомления до тех пор, пока функция не завершится или пока он не откажется от подписки. Для работы с Observable используется библиотека RxJS, которая реализует Observable тип данных, предоставляет службы для работы с Observable. Службы могут: создавать несколько асинхронных потоков, фильтровать эти потоки, перебирать значения в потоке, преобразование асинхронного кода в Observable значения в потоке.

Декоратор @Pipe используется для написания правила, который используется в совокупности binding, и может видоизменять предоставление данных.

Material Design язык дизайна, созданный от Google, можно использовать Angular Material вместо или совместно с Bootstrap в проекте, имеет совместимость с фреймворком, имеет элементы категорий управлений форм, таблицы данных, всплывающие окна, макет, навигация, кнопки, индикаторы и многое другое, которые можно установить в свой проект, поискав на официальном сайте. Используется во многих проектах, поддерживается популярными браузерами, разбита на отдельные модули и поддерживает стили Material Design.

Karma инструмент фреймворка для тестирования проекта, автоматизирует тесты и имеет широкий функционал. Конфигурация прописывается в файле karma.conf.js.

2.2 Node.js Express App

Node.js Express App это веб-фреймворк для Node.js, имеет готовые модули http для RestAPI, маршрутизации, абстракции для создании сервера, серверной логики, работы с запросами, куками. Как и все вышеперечисленное устанавливаем с помощью менеджера пакетов npm, в проекте используем для связывания frontend и БД, обработки GET, POST, PUT, UPDATE, DELETE запросов.

2.3 MySql

MySql – это система управления реляционными базами данных, с открытым исходным кодом, поддерживаемый компанией Oracle, в проекте использовалась расширенная и облегченная версия MySqli, SQL скрипт прописывался в Microsoft Sql Server. Реляционная база данных – это структурированные данные в таблицах с предопределенными взаимосвязями между ними и подчиняющиеся 6 формам нормализации БД, то есть не должно быть повторений, каждый не ключевой атрибут неприводимо зависит от Primary Key, каждый ключевой атрибут нетранзитивно зависит от Primary Key, каждая нетривиальная и неприводимая слева функциональная зависимость обладает детерминантом, все нетривиальные многозначные значения зависимости являются функциональными зависимостями от её потенциальных ключей, отсутствие зависимых соединений между атрибутами, удовлетворяет всем нетривиальным зависимостям соединения, данные формы могут выполняться только последовательно, для приведения к нормальной реляционной форме.

2.4 IDE

Web Storm – IDE среда направленная на разработку на JavaScript и связанных на ней технологиях, полная поддержка JavaScript, TypeScript, React, React Native, Electron, Vue, Angular, Node.js, HTML, CSS, встроенные инструменты разработчика, продвинутая интеграция с VCS, HTTP client, Code With Me, Git, кастомизация. Данная среда была выбрана так как наилучшим образом подходит для написания проекта Angular Framework.

PHP Storm – IDE среда направленная на разработку на PHP и связанных на ней технологиях, отладка, тестирование, профилирование, поддержка Git, SVN, Mercurial, Perforce, работа с БД, Docker, Composer, полная поддержка всех фреймворков на PHP, Node.js, поддержка TSLint и Angular. Данная среда была

выбрана для написания backend части приложения, и она отлично подходит по требуем запросам проекта и имеет взаимосвязь с frontend частью проекта.

3 Проектная часть

Разработка проектной части включает, разработку frontend и backend части интернет-магазина, ведения контроля версий, отладку, и сборку проекта. Все изменения в проекте наход ятся на репозитории GitHub по ссылке в приложении.

Общая схема взаимодействий технологий, применяемых в проекте изображена на рисунке-3.1.

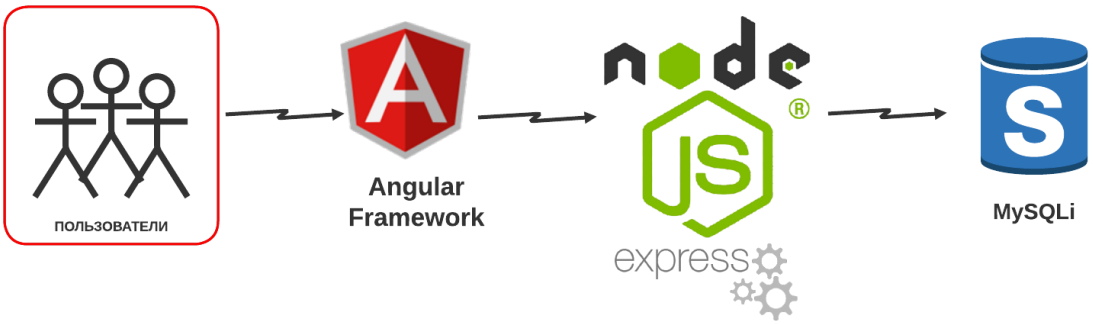


Рисунок-3.1 – Стек взаимодействий технологий веб-сайта

3.1 Разработка frontend

Разработка началась с создания пустого проекта Angular framework с помощью пакетного менеджера npm. Для начала устанавливаем Node.js, далее создаем проект в WebStorm, либо через команду npm i @angular/cli -g в терминале. Для запуска используем сочетание клавиш ctrl + F5 или команду ng serve в терминале.

Небольшие этапы создания frontend части, все этапы можно будет просмотреть с помощью контроля версий Git на моём репозитории, ссылка будет в приложении:

- первым этапом будет создание необходимых компонентов, изображен на рисунке-3.1.1, и настройки маршрутизации, далее устанавливаем необходимую тему с использованием Angular Material;

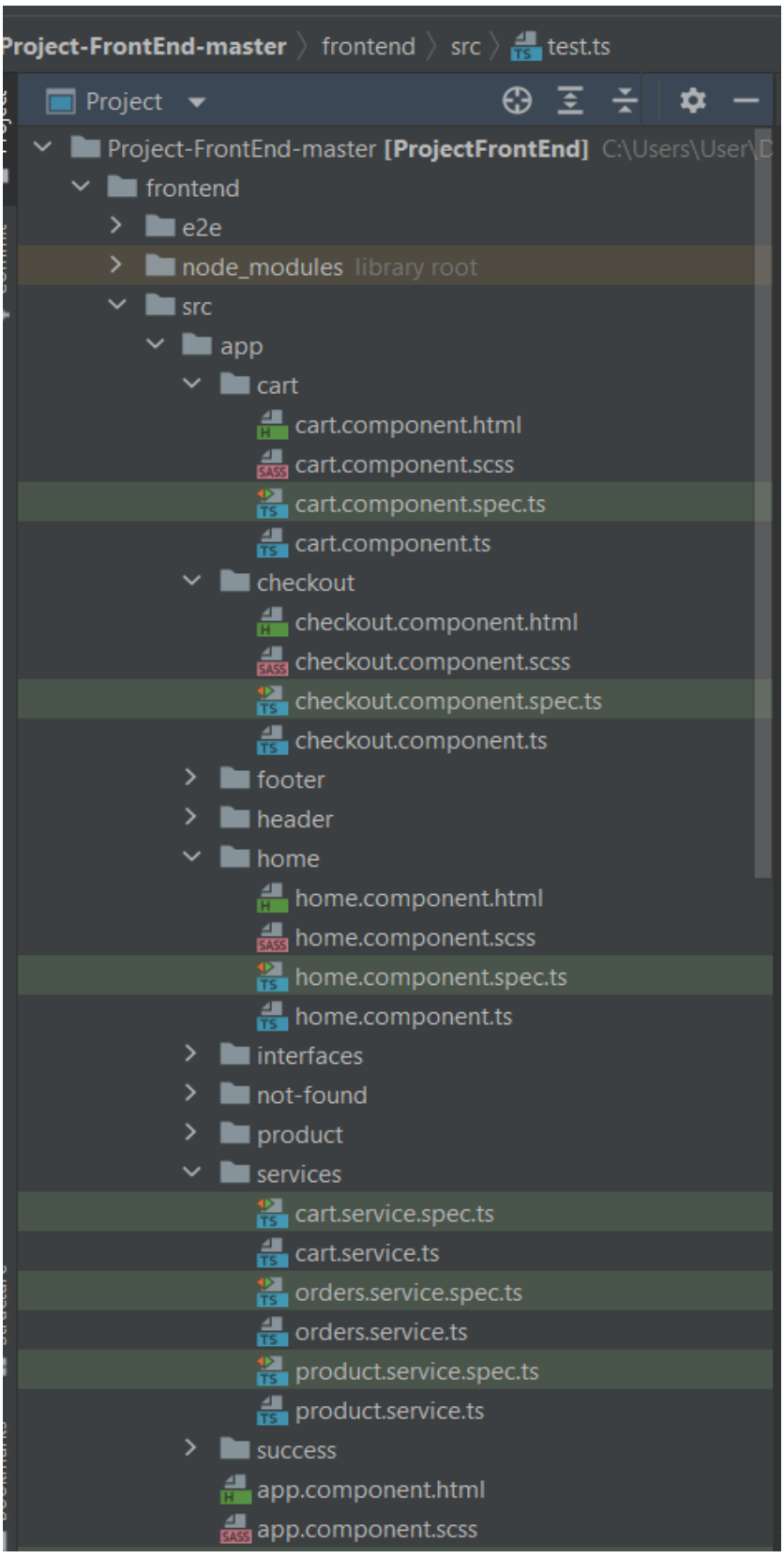
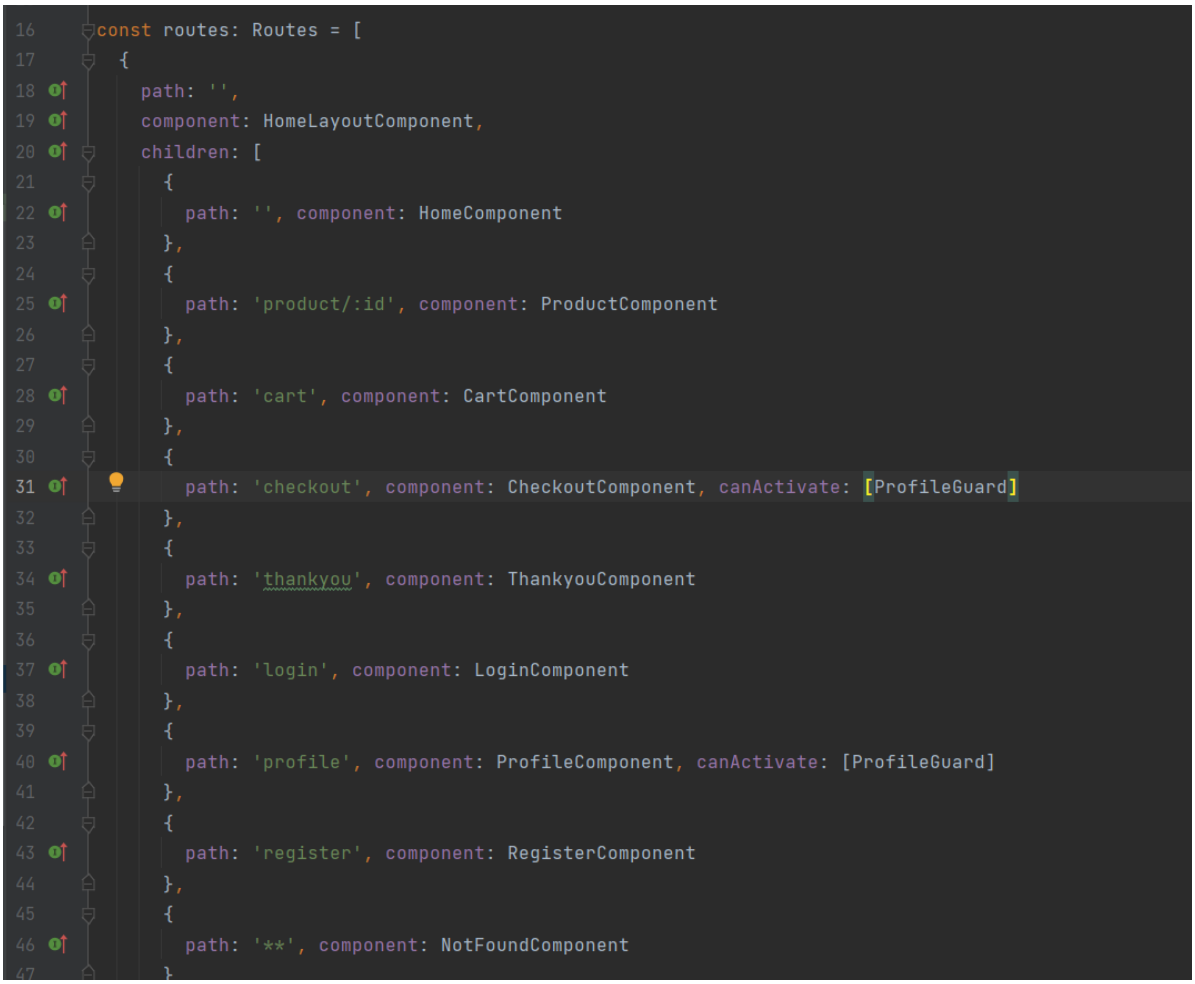


Рисунок-3.1.1 – Содержание компонентов проекта

- создание маршрутизации для компонентов, разрабатывается для корректной работы переходов между страницами, и исключения возможности попадания на приватные страницы, либо страниц которых не существует с кодом 404, код изображен на рисунке-3.1.2;



```
const routes: Routes = [
  {
    path: '',
    component: HomeLayoutComponent,
    children: [
      {
        path: '', component: HomeComponent
      },
      {
        path: 'product/:id', component: ProductComponent
      },
      {
        path: 'cart', component: CartComponent
      },
      {
        path: 'checkout', component: CheckoutComponent, canActivate: [ProfileGuard]
      },
      {
        path: 'thankyou', component: ThankyouComponent
      },
      {
        path: 'login', component: LoginComponent
      },
      {
        path: 'profile', component: ProfileComponent, canActivate: [ProfileGuard]
      },
      {
        path: 'register', component: RegisterComponent
      },
      {
        path: '**', component: NotFoundComponent
      }
```

Рисунок-3.1.2 – Создание маршрутизации

- создание сервиса с GET запросом одного продукта из БД, с использованием hhtp модуля, используется с целью получения одного продукта и в дальнейшем использовать в других частях программы, этап изображен на рисунке-3.1.3;

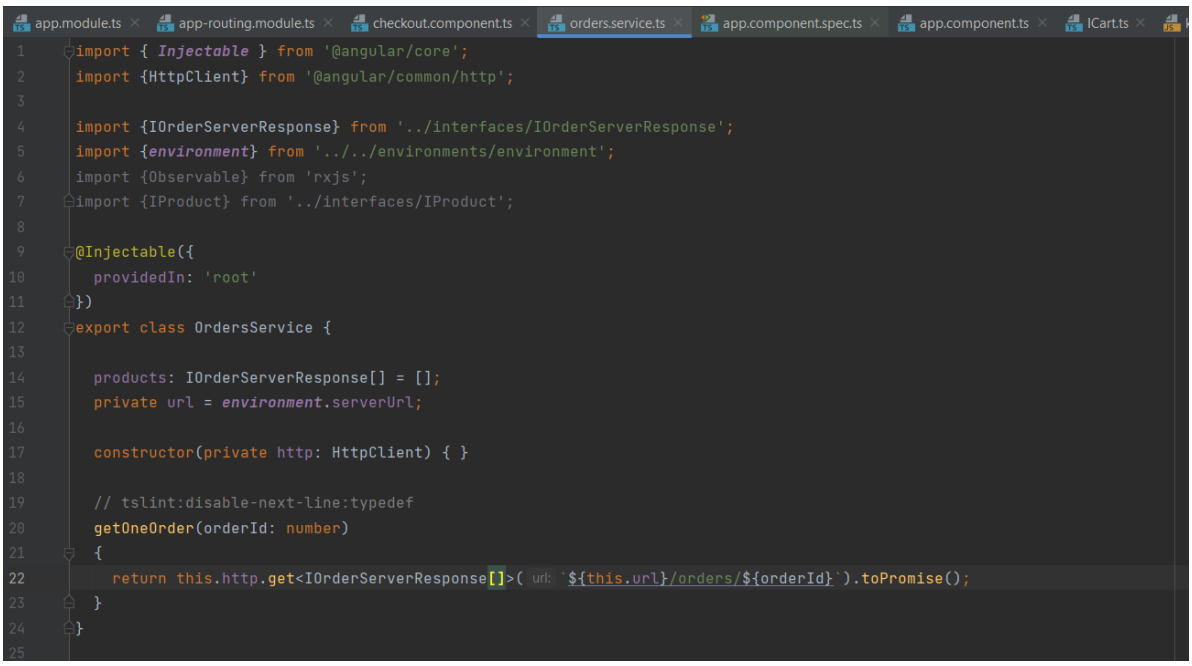


Рисунок-3.1.3 – GET запрос одного продукта

- создание сетки и карточки продукта, с использованием MatCardModule от Material Design, использование фреймворка css дает возможность автоматизированной адаптивности вёрстки сайта, код изображен на рисунке-3.1.4;

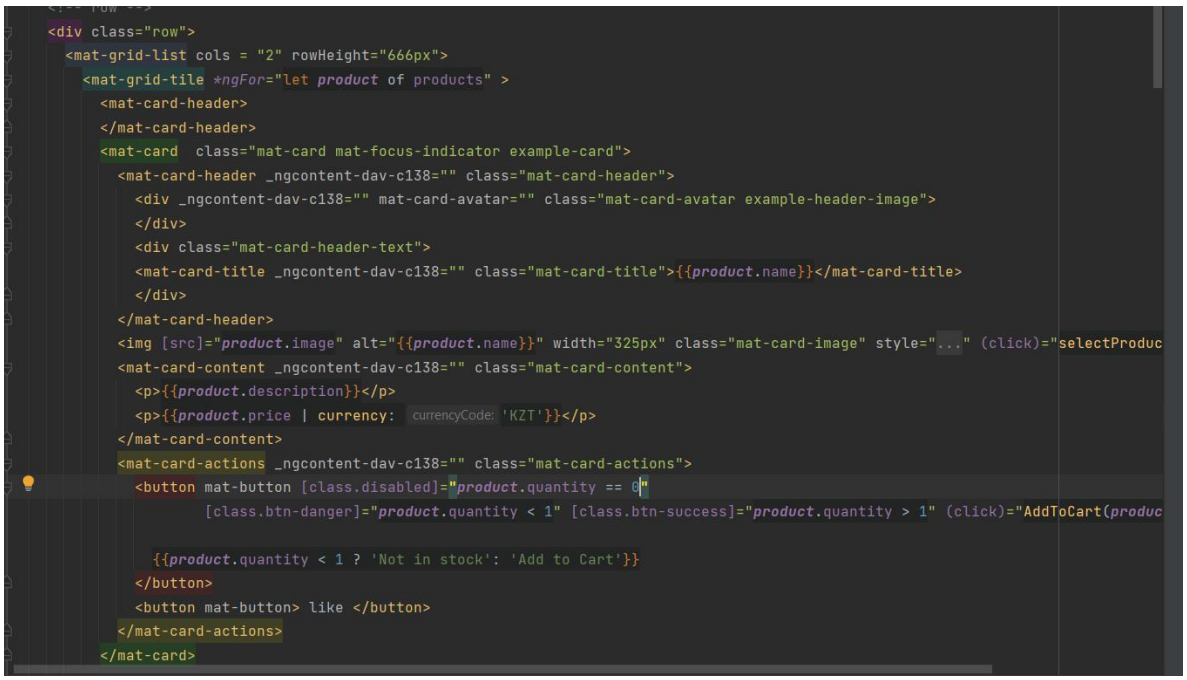


Рисунок-3.1.4 – Создание сетки и карточки продукта

- добавление binding, для получения информации по продукту, в компоненте, данная технология в фреймворке позволяет обращаться к значениям

свойств находящихся в компоненте и прописывать оптимизированную логику получения и отображения значений, пример использования изображено на рисунке-3.1.4;

- получение одного продукта из одной категории, позволяет получить точечную информацию об продукте для реализации дальнейшей логики компонента, код изображен на рисунке-3.5, так же исходя из этой логики, получаем данные всех продуктов для дальнего отображения ассортимента интернет-магазина, изображено на рисунках 3.1.6, 3.1.7;

```
  }
  getOneProduct(id: number): Observable<IProduct>
  {
    return this.http.get<IProduct>(this.url + '/products' + id);
  }
  getOneProductInOneCategory(category: string): Observable<IProduct[]>
  {
    return this.http.get<IProduct[]>(this.url + '/category/' + category);
  }
}
```

Рисунок-3.1.5 – Получение одного продукта в одной категории

```
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.scss']
})
export class HomeComponent implements OnInit {

  products: IProduct[];
  constructor(private productService: ProductService, private cartService: CartService , private router: Router) { }

  ngOnInit(): void {
    this.productService.getAllProducts().subscribe( next: (prod: IServerResponse ) => {
      this.products = prod.products;

      console.log(this.products);
    });
  }

  selectProduct(id: number): void
  {
    this.router.navigate( commands: ['/product', id]).then();
  }
  // tslint:disable-next-line:typedef
  AddToCart(id: number)
  {
    this.cartService.AddProductToCart(id);
  }

}
```

Рисунок-3.1.6 – Получение данных о продукте

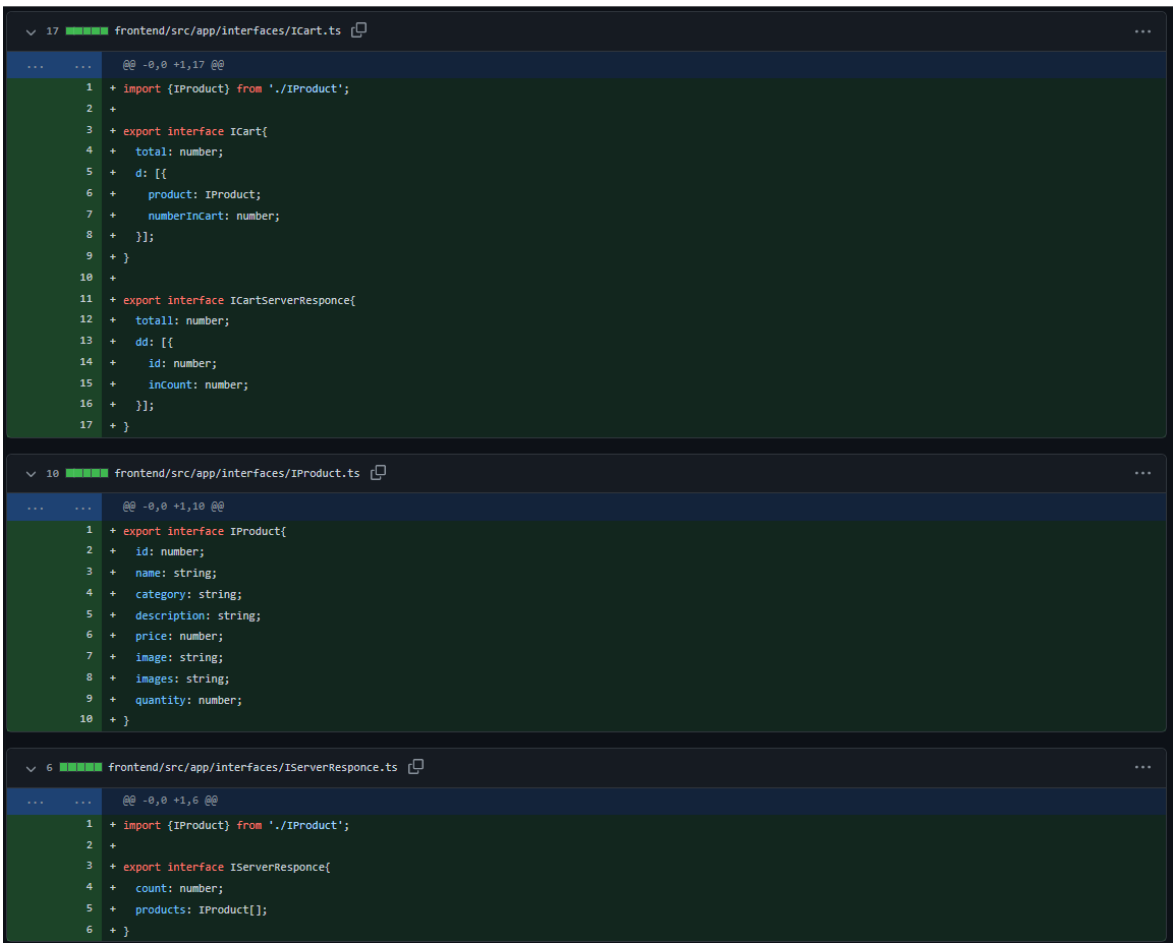
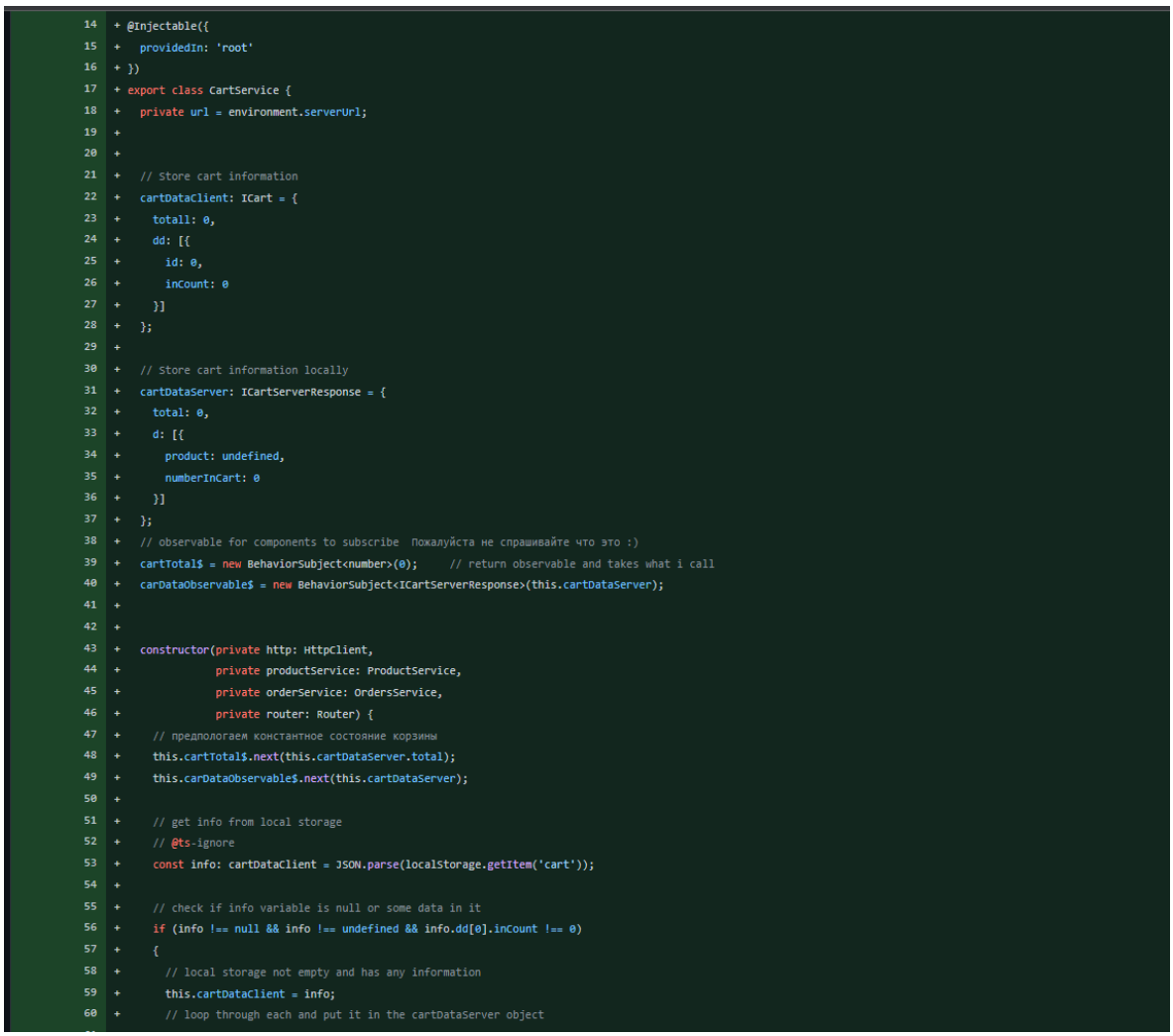


Рисунок-3.1.7 – Создание интерфейса продукта и получения данных с сервера

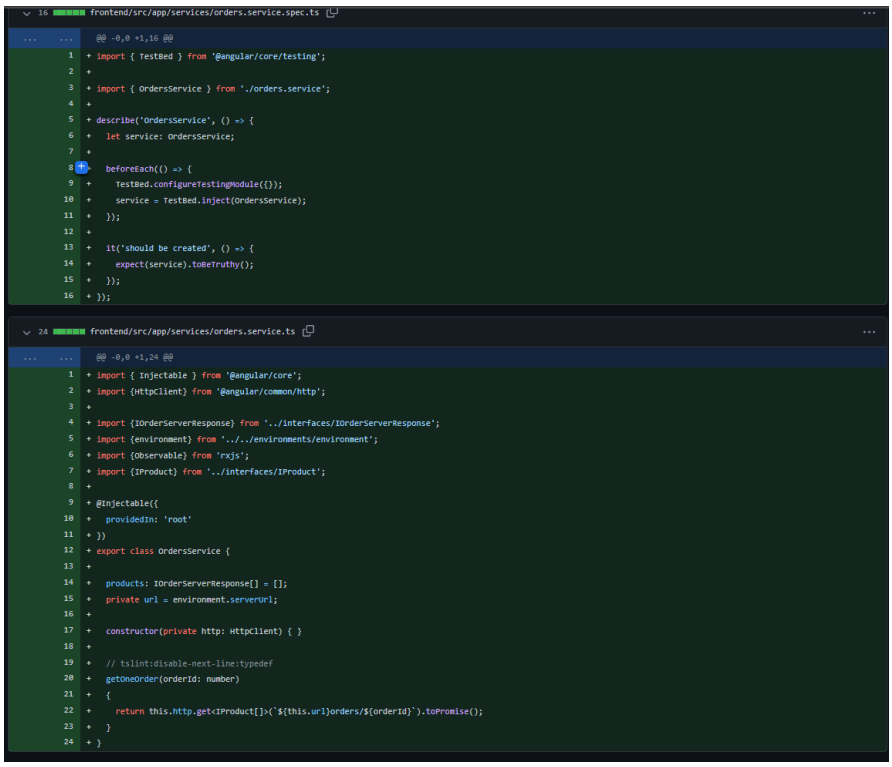
- создание компонента для корзины, позволит в дальнейшем реализовать сервис по оформлению заказа на основе данных товара добавленных в корзину товаров, что является неотъемлемой частью интернет-магазина, этап изображен на рисунке-3.1.8;



```
+ @Injectable({
+   providedIn: 'root'
+ })
+ export class CartService {
+   private url = environment.serverUrl;
+
+
+   // Store cart information
+   cartDataClient: ICart = {
+     totall: 0,
+     dd: [{
+       id: 0,
+       inCount: 0
+     }]
+   };
+
+   // Store cart information locally
+   cartDataServer: ICartServerResponse = {
+     total: 0,
+     d: [{
+       product: undefined,
+       numberInCart: 0
+     }]
+   };
+   // observable for components to subscribe  Пожалуйста не спрашивайте что это :)
+   cartTotal$ = new BehaviorSubject<number>(0);     // return observable and takes what i call
+   carDataObservable$ = new BehaviorSubject<ICartServerResponse>(this.cartDataServer);
+
+
+   constructor(private http: HttpClient,
+               private productService: ProductService,
+               private orderService: OrdersService,
+               private router: Router) {
+     // предпологаем константное состояние корзины
+     this.cartTotal$.next(this.cartDataServer.total);
+     this.carDataObservable$.next(this.cartDataServer);
+
+     // get info from local storage
+     // @ts-ignore
+     const info: cartDataClient = JSON.parse(localStorage.getItem('cart'));
+
+     // check if info variable is null or some data in it
+     if (info !== null && info !== undefined && info.dd[0].inCount !== 0)
+     {
+       // local storage not empty and has any information
+       this.cartDataClient = info;
+       // loop through each and put it in the cartDataServer object
```

Рисунок-3.1.8 – Создание компонента получения данных для корзины продуктов

- создание сервиса для заказа, позволит оформлять заказ, сохранять данные и в дальнейшем можно улучшить, для реализации аналитики продаж, этап изображен на рисунке-3.1.9;



frontend/src/app/services/orders.service.spec.ts

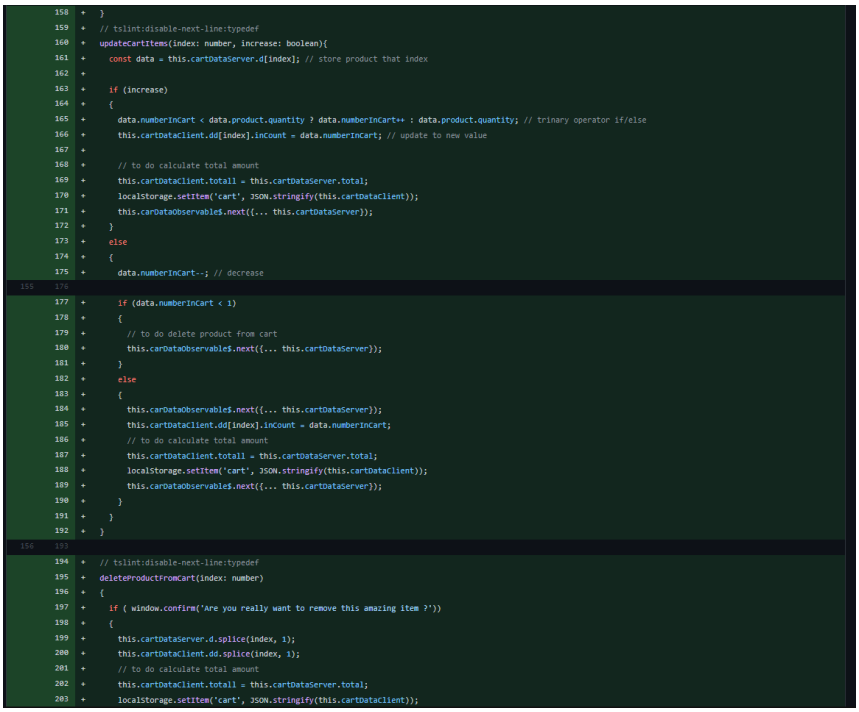
```
@@ -0,0 +1,16 @@
+ import { TestBed } from '@angular/core/testing';
+
+ import { OrdersService } from './orders.service';
+
+ describe('OrdersService', () => {
+   let service: OrdersService;
+
+   beforeEach(() => {
+     TestBed.configureTestingModule({});
+     service = TestBed.inject(OrdersService);
+   });
+
+   it('should be created', () => {
+     expect(service).toBeTruthy();
+   });
+ });
```

frontend/src/app/services/orders.service.ts

```
@@ -0,0 +1,24 @@
+ import { Injectable } from '@angular/core';
+ import {HttpClient} from '@angular/common/http';
+
+ import {IOrderServerResponse} from '../interfaces/IOrderServerResponse';
+ import {environment} from '../../environments/environment';
+ import {Observable} from 'rxjs';
+ import {IProduct} from '../interfaces/IProduct';
+
+ @Injectable({
+   providedIn: 'root'
+ })
+ export class OrdersService {
+
+   products: IOrderServerResponse[] = [];
+   private url = environment.serverUrl;
+
+   constructor(private http: HttpClient) { }
+
+   // tslint:disable-next-line:typedef
+   getOneOrder(orderId: number)
+   {
+     return this.http.get<IProduct[]>(`${this.url}orders/${orderId}`).toPromise();
+   }
+ }
```

Рисунок-3.1.9 – Создание сервиса заказа

- создание CRUD операций для продукта, добавление интерфейса заказа, обеспечивает эргономичность платформы для удобства использования пользователем, этап изображен на рисунке-3.1.10, интерфейс изображен на рисунке-3.1.11;



```
158 +   }
159 +   // tslint:disable-next-line:typedef
160 +   updateCartItems(index: number, increase: boolean){
161 +     const data = this.cartDataServer.d[index]; // store product that index
162 +
163 +     if (increase)
164 +     {
165 +       data.numberInCart < data.product.quantity ? data.numberInCart++ : data.product.quantity; // trinary operator if/else
166 +       this.cartDataClient.dd[index].inCount = data.numberInCart; // update to new value
167 +
168 +       // to do calculate total amount
169 +       this.cartDataClient.totall = this.cartDataServer.total;
170 +       localStorage.setItem('cart', JSON.stringify(this.cartDataClient));
171 +       this.carDataObservable$.next({... this.cartDataServer});
172 +     }
173 +     else
174 +     {
175 +       data.numberInCart--; // decrease
155 176
177 +       if (data.numberInCart < 1)
178 +       {
179 +         // to do delete product from cart
180 +         this.carDataObservable$.next({... this.cartDataServer});
181 +       }
182 +       else
183 +       {
184 +         this.carDataObservable$.next({... this.cartDataServer});
185 +         this.cartDataClient.dd[index].inCount = data.numberInCart;
186 +         // to do calculate total amount
187 +         this.cartDataClient.totall = this.cartDataServer.total;
188 +         localStorage.setItem('cart', JSON.stringify(this.cartDataClient));
189 +         this.carDataObservable$.next({... this.cartDataServer});
190 +       }
191 +     }
192 +   }
156 193
194 +   // tslint:disable-next-line:typedef
195 +   deleteProductFromCart(index: number)
196 +   {
197 +     if ( window.confirm('Are you really want to remove this amazing item ?'))
198 +     {
199 +       this.cartDataServer.d.splice(index, 1);
200 +       this.cartDataClient.dd.splice(index, 1);
201 +       // to do calculate total amount
202 +       this.cartDataClient.totall = this.cartDataServer.total;
203 +       localStorage.setItem('cart', JSON.stringify(this.cartDataClient));
```

Рисунок-3.1.10 – CRUD, IorderResponce

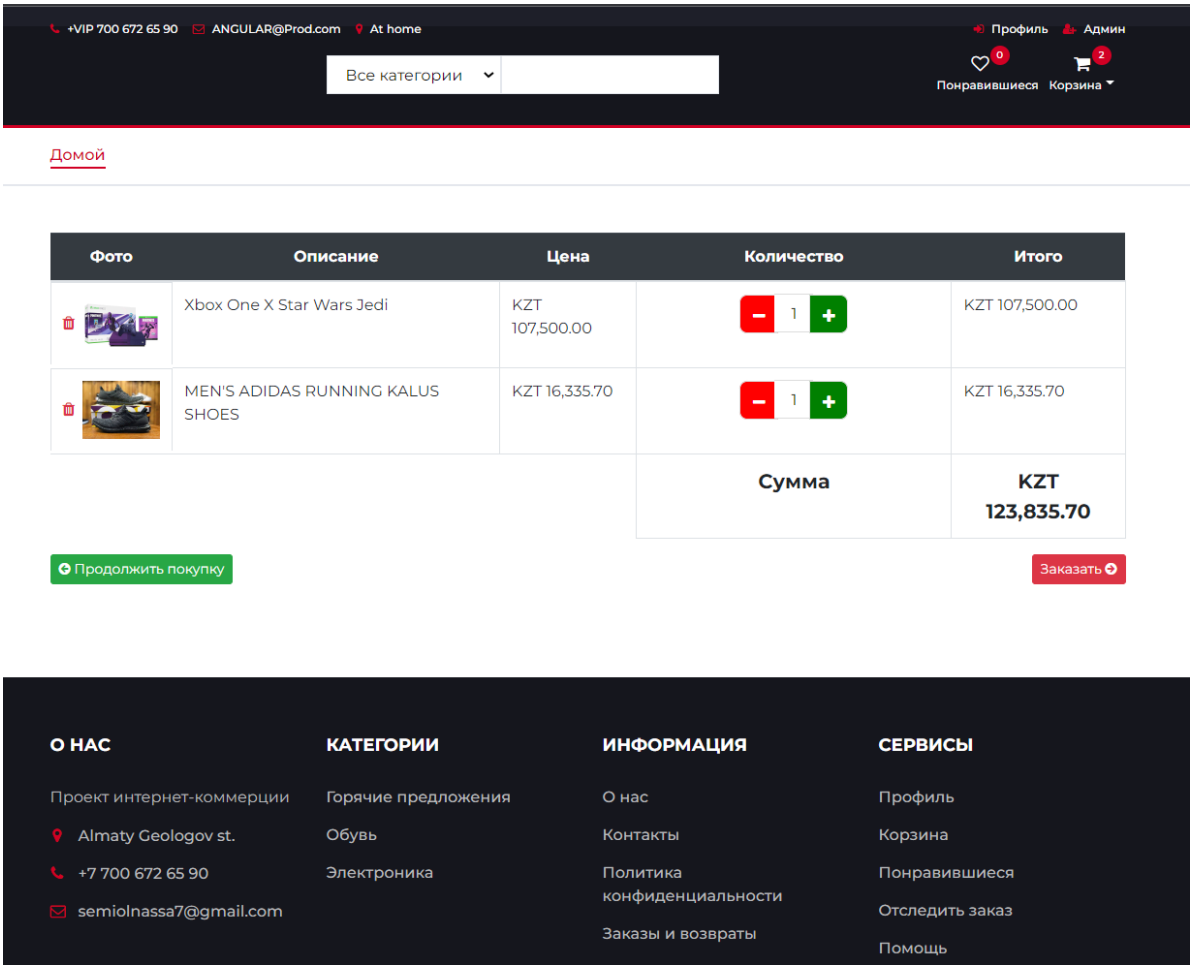


Рисунок-3.1.11 – Интерфейс корзины товара

- создание заказа с добавлением данных в БД, код изображен на рисунке-3.1.12, так же сопутствующим действием будет создание эргономичного интерфейса для оформления заказа который изображен на рисунке-3. 1.13;

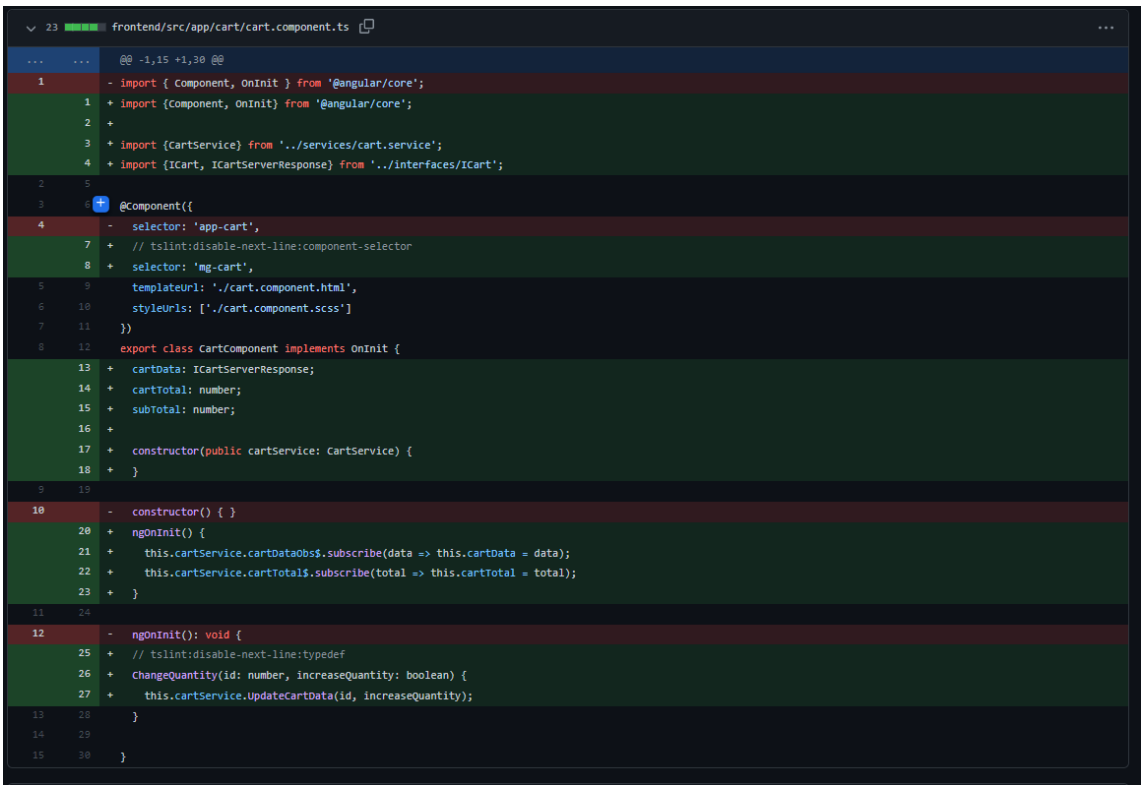


Рисунок-3.1.12 – Создание заказа и добавление данных в БД

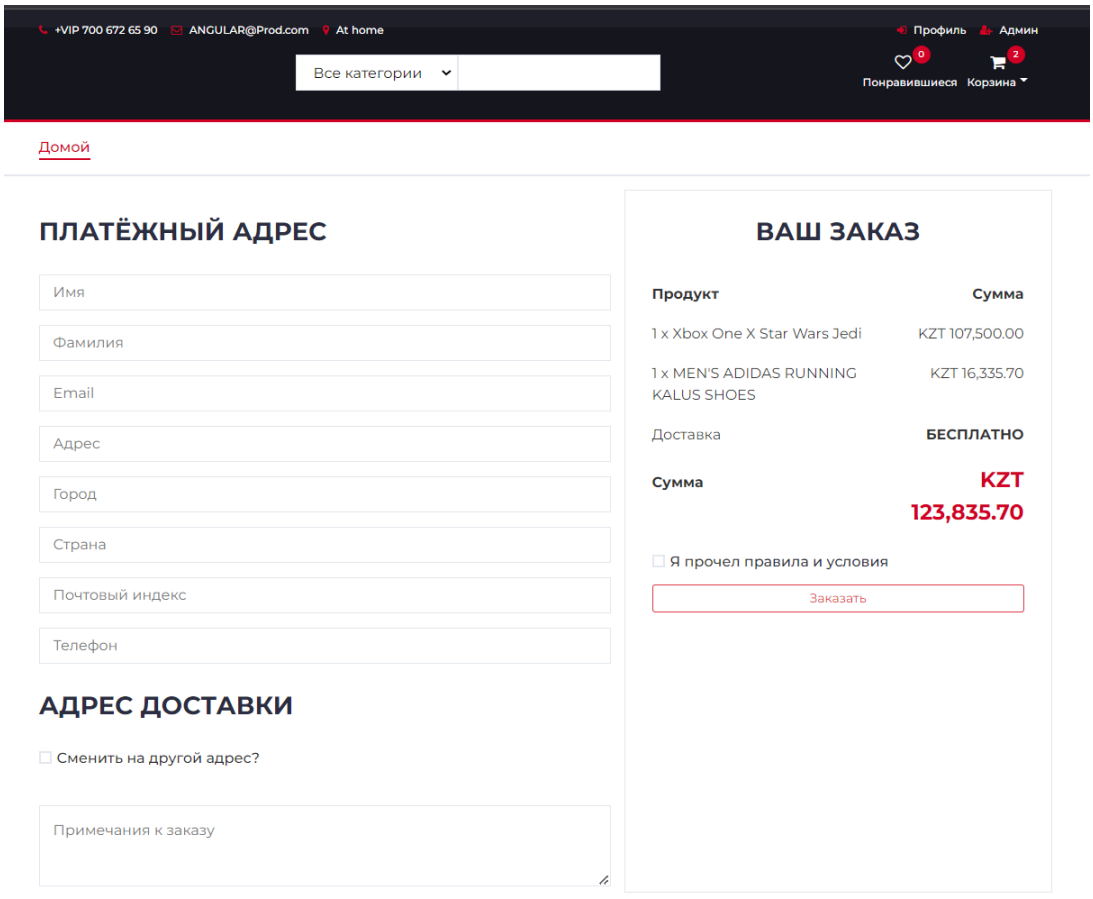


Рисунок-3.1.13 – Интерфейс оформления заказа

- разработка системы регистрации и авторизации, с дополнительной возможностью входа с помощью google account, изображено на рисунках 3.1.14, 3.1.15, 3.1.16;

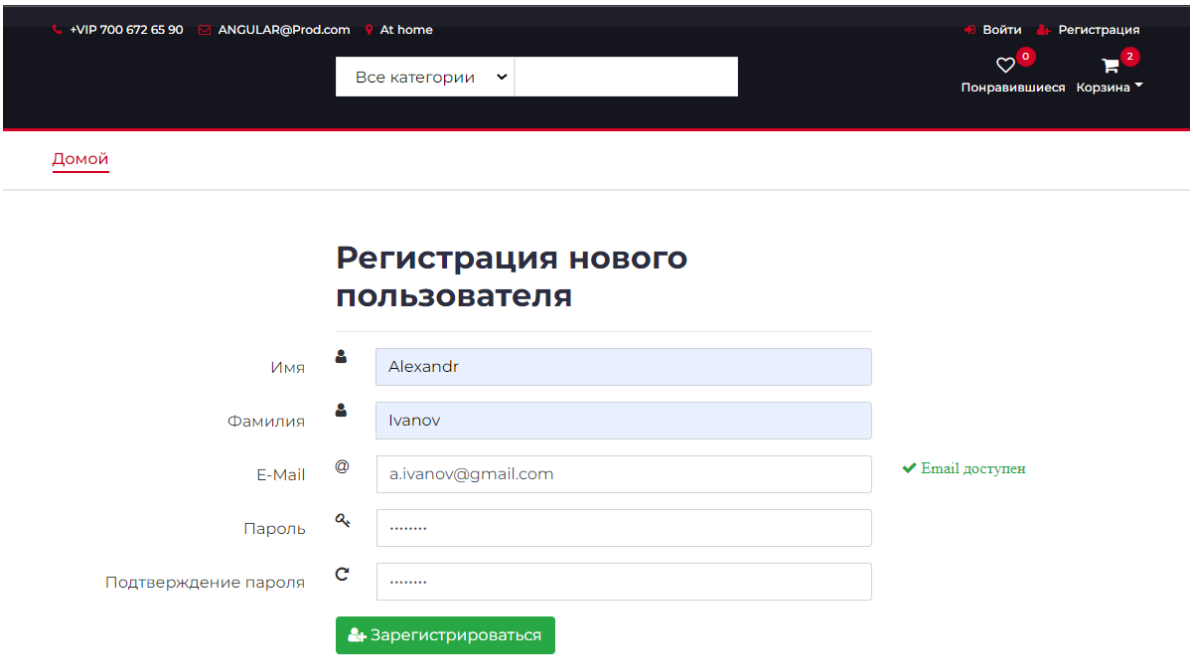


Рисунок-3.1.14 – Регистрация клиента

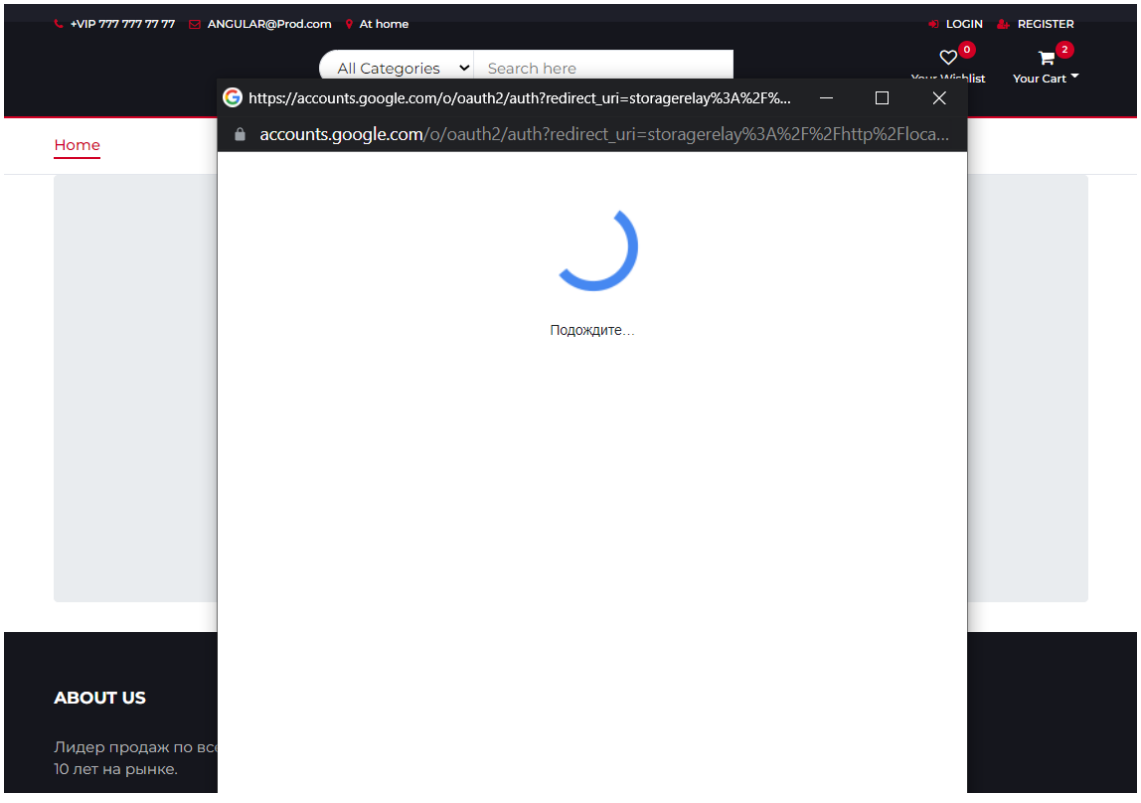


Рисунок-3.1.15 – Авторизация с помощью google account

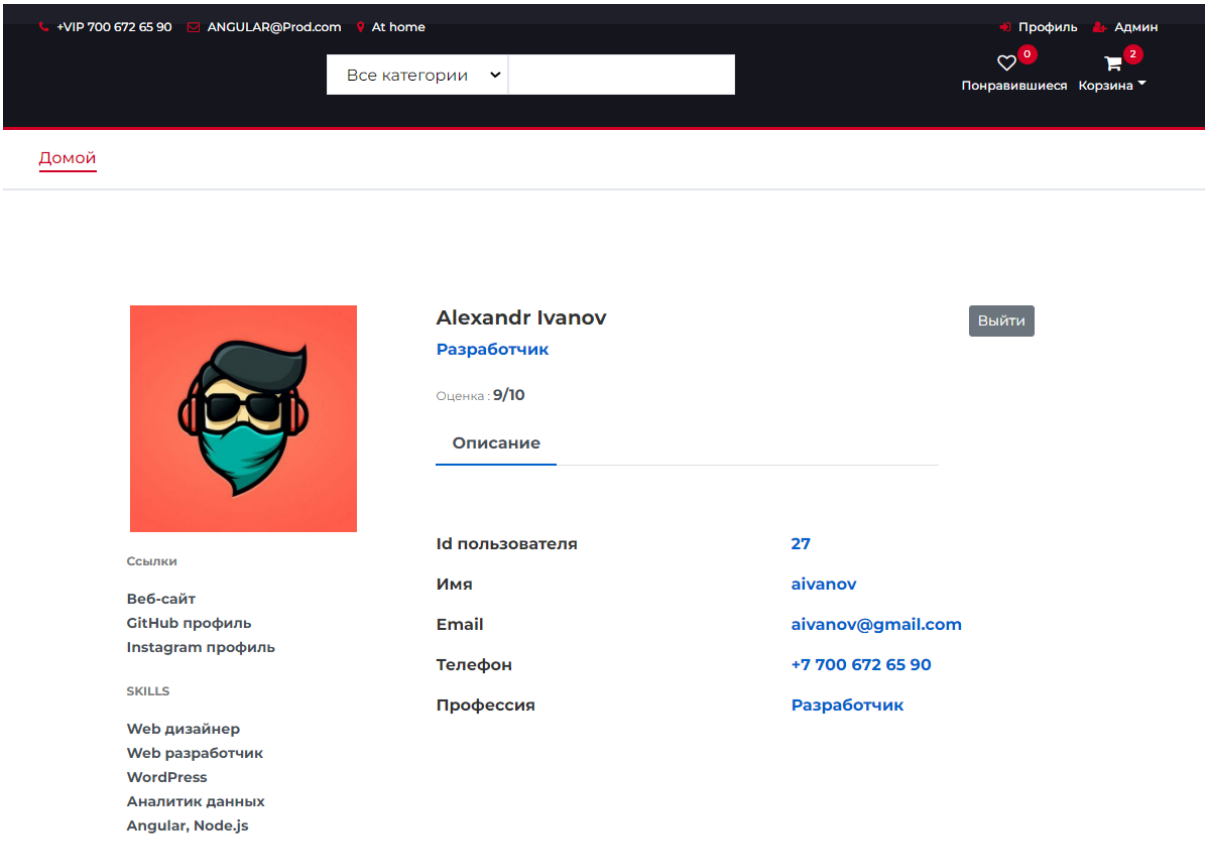


Рисунок-3.1.16 – Профиль клиента

- разработка интерфейса интернет-магазина, изображено на рисунке-3.1.17.

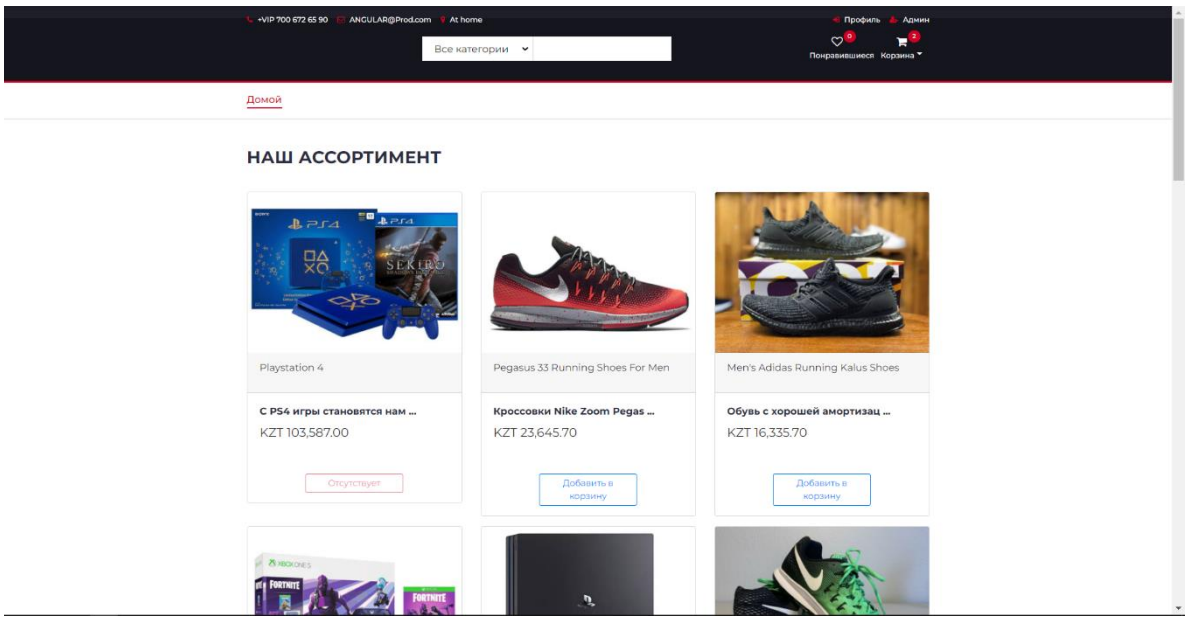


Рисунок-3.1.17 – Интерфейс

3.2 Разработка backend

Разработка backend производиться с помощью веб-фреймворка Node.js Express App, с использованием в качестве СУРБД MySqli, запускаемый на XAMPP вместе с Apache, в качестве данных для продуктов используем сгенерированный SQL-скрипт, для визуального контроля БД используем PHPMyAdmin, для проверки работоспособности запросов на сервер используем Postman.

Разработка началась с создания пустого проекта Node.js Express App с помощью пакетного менеджера npm. Для начала устанавливаем Node.js , далее создаем проект в PHPStorm, либо через команду npm install express@4.17.1 в терминале, до устанавливаем npm install serve-index@1.9.1. Для запуска используем сочетание клавиш ctrl + F5 или команду node server.js в терминале. Так же использовал nodemon для динамического мониторинга изменений на сервере.

Все этапы создания Backend части, все этапы можно будет просмотреть с помощью контроля версий Git на моём репозитории, ссылка будет в приложении:

- первым делом создаем сервер локально на localhost:3000, для обработки http запросов в БД и полноценной работы веб-сайта, проверяем отладку, данный этап изображен на рисунке-3.2.1;

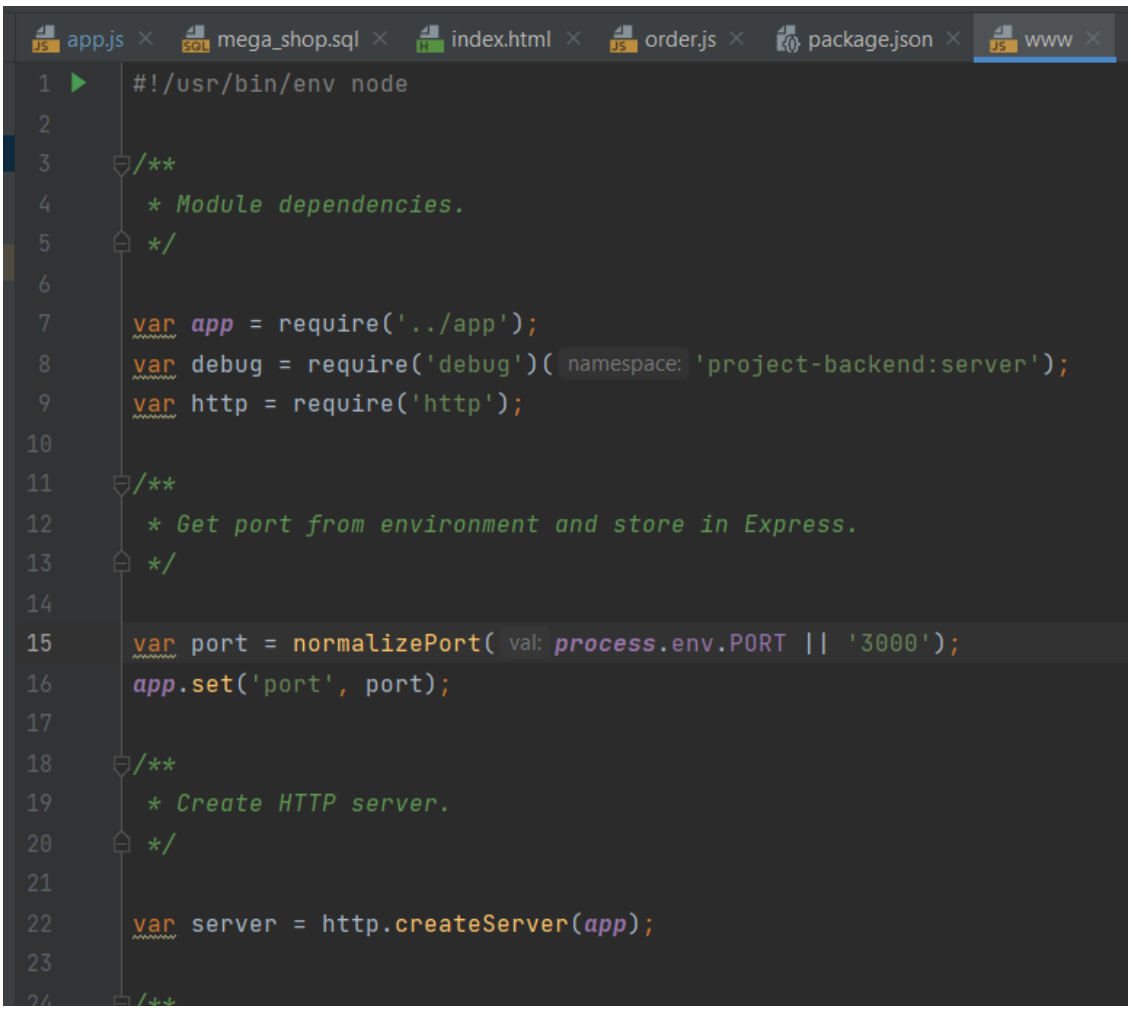


Рисунок-3.2.1 – Создание веб-сервера

- устанавливаем связь с БД, предварительно запустив комплектующие XAMPP, Apache, MySql, для полноценной работы backend части проекта, данные действия изображены на рисунке-3.2.2, 3.2.3;

app.js | mega_shop.sql | index.html | order.js | package.json | www | helpers.js

```
const MySqli = require('mysqli') // обновленная версия устаревшего MySQL расшире
let conn = new MySqli( config: {
    host: 'localhost',
    post: 3306,
    user: 'Semiol',
    passwd: '1234',
    db: 'mega_shop'
});

let db = conn.emit( fromSlave: false,  db: '' );

module.exports = {
    database: db
};

```

Рисунок-3.2.2 – Установление связи с БД

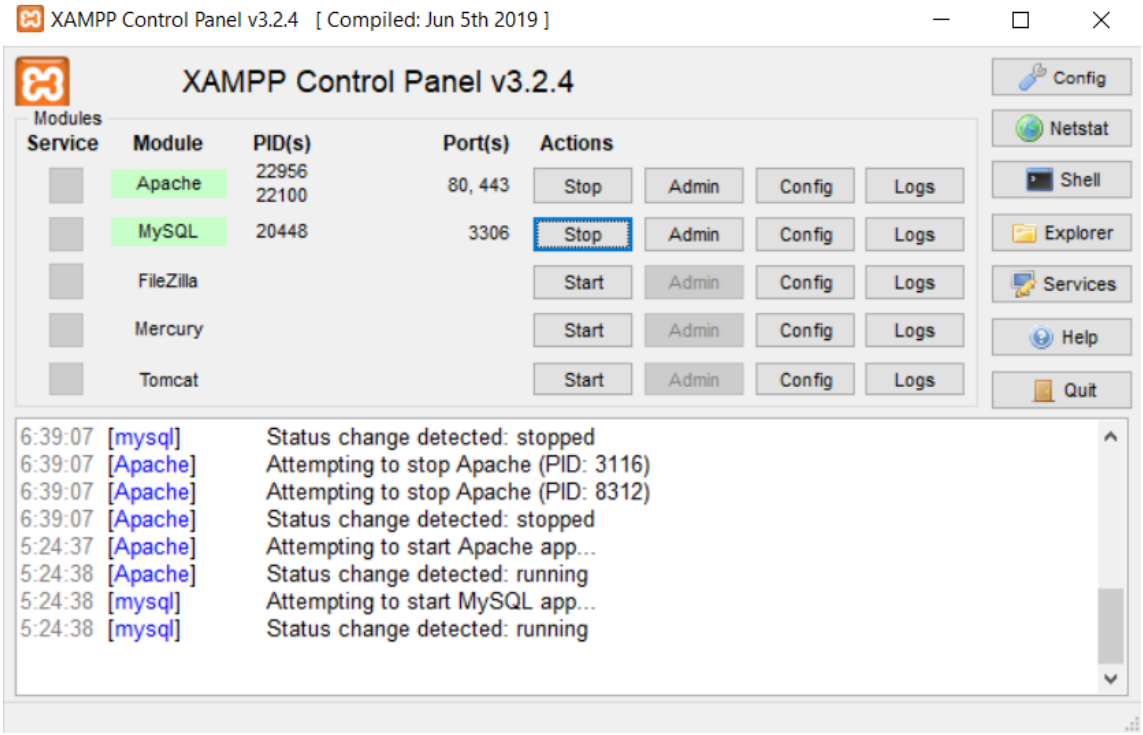


Рисунок-3.2.3 – Запуск Apache, MySql

- Запускаем скрипт из backEnd.docx, с предварительно написанным скриптом SQL, с созданием реляционной БД и добавлением данных в таблицы, в Microsoft SQL Server для заполнения данными и проверяем в PHPMyAdmin, результат данных действий изображен на рисунке-3.2.4;

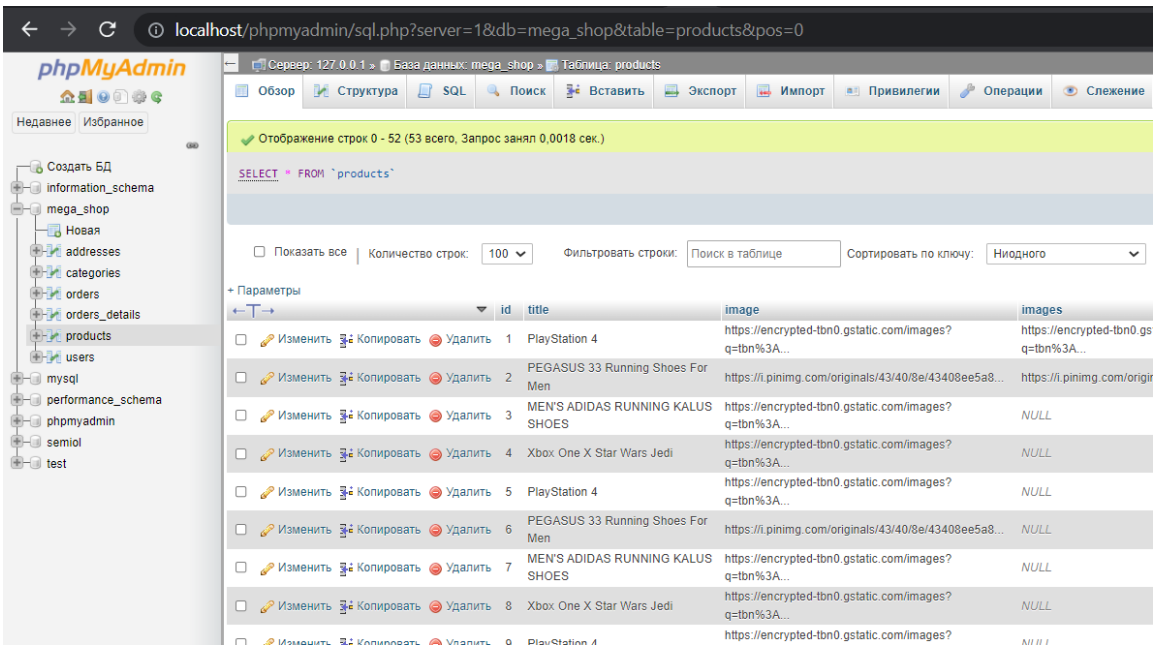


Рисунок-3.2.4 – Отображение загруженных данных

- Создание GET запроса для получения продуктов, для установление связи с методом из frontend части, для корректной передачи данных между backend и frontend, код запроса изображен на рисунке-3.2.5;

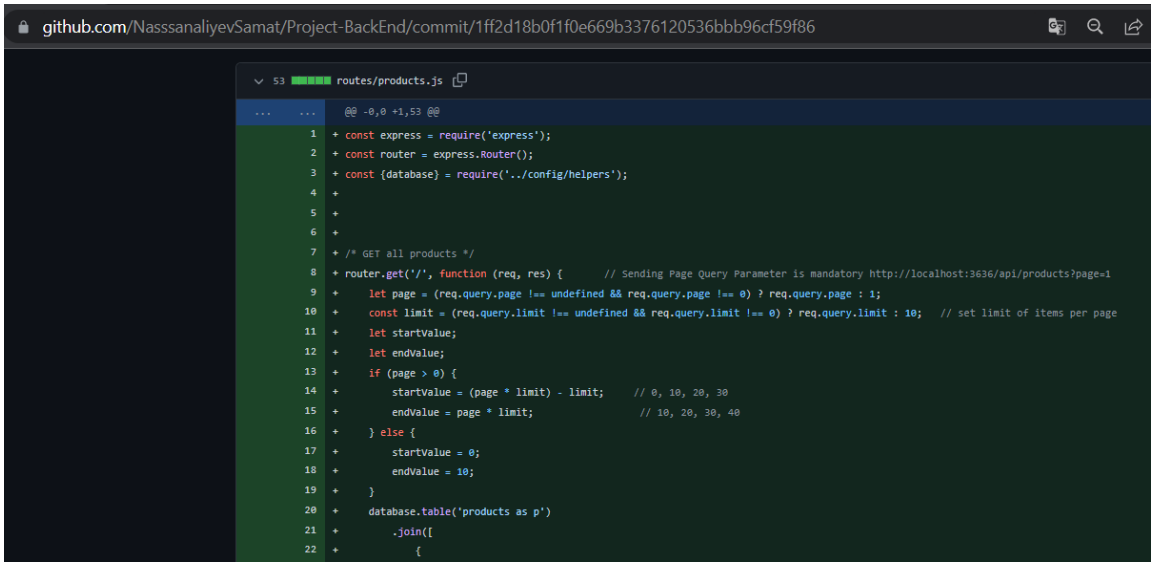


Рисунок-3.2.5 – Получение продуктов

- Создание GET запроса для получения одного продукта и категории, Рисунок-3.2.6;

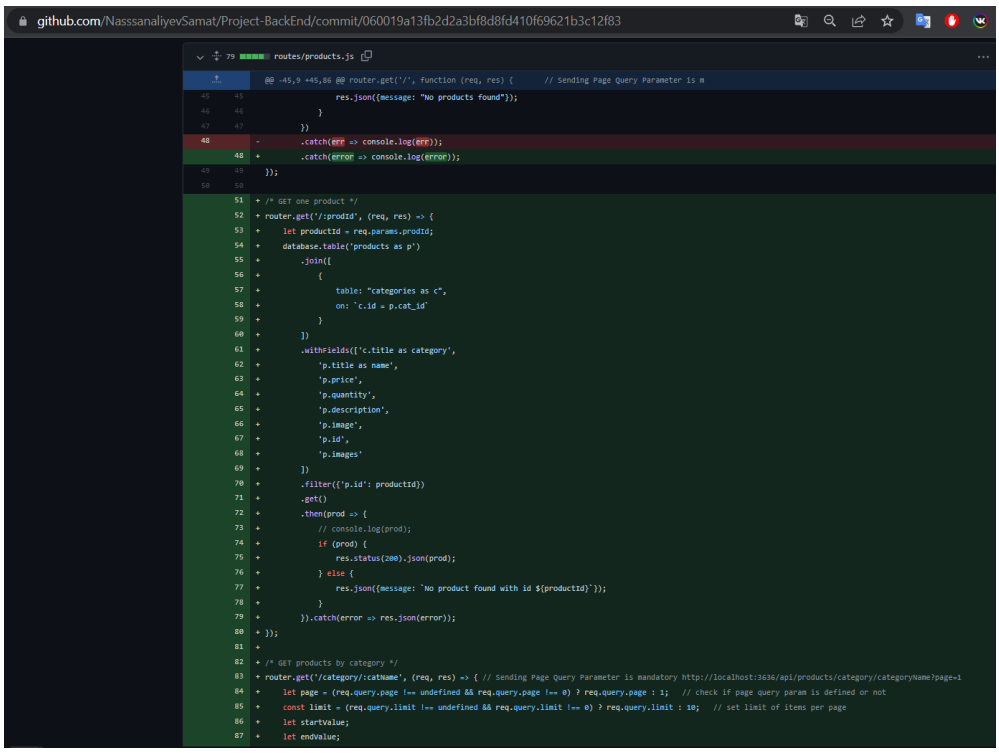


Рисунок-3.2.6 – Получение одного продукта и категории

- Создание GET запроса для получения одного и всех заказов, код на рисунке-3.2.7;

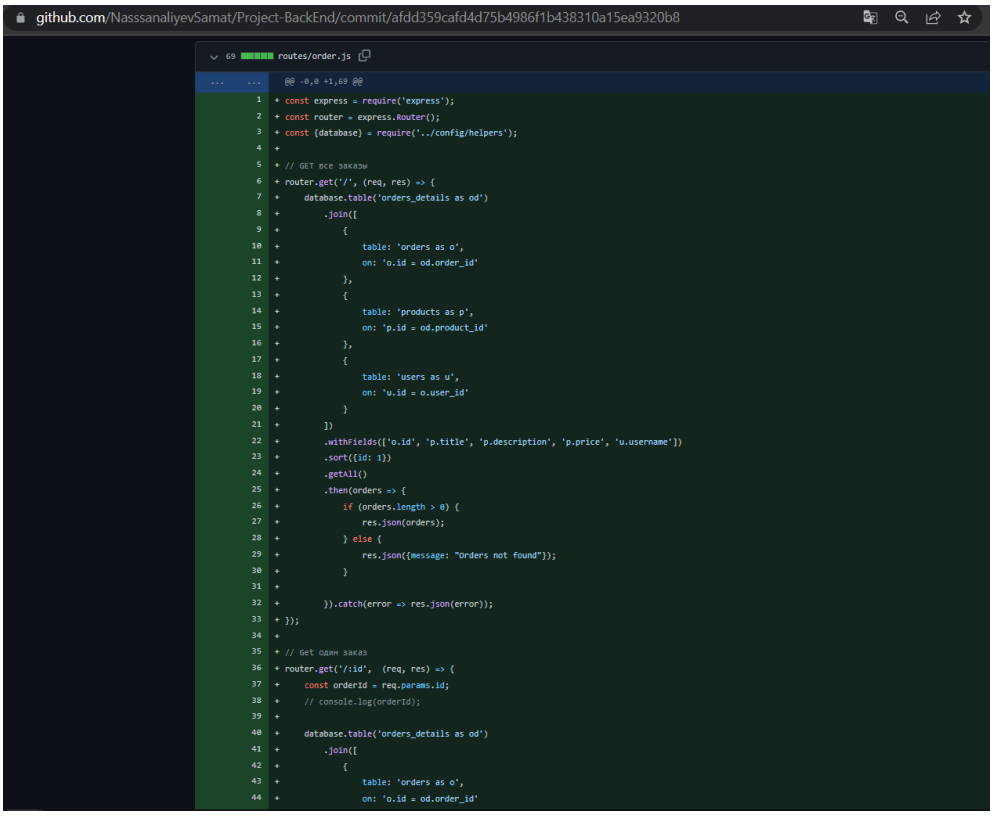


Рисунок-3.2.7 – Получение одного и всех заказов

- Создание POST запроса для создания заказа, Рисунок-3.2.8.

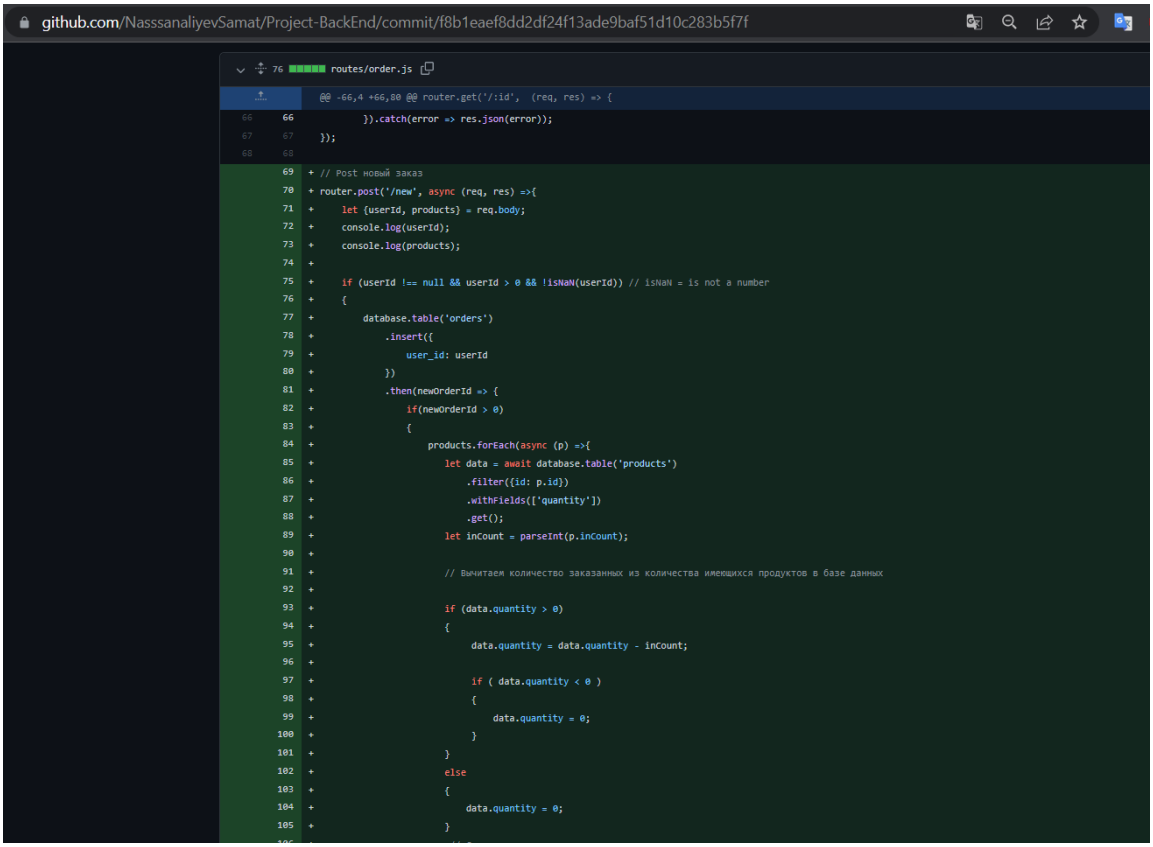


Рисунок-3.2.8 – Создание POST запроса

3.3 Диаграммы

Диаграммы – общее визуальное представление для восприятия полноты процессов, происходящих внутри отдельных систем. Существуют большое разнообразие диаграмм ориентированных на конкретную задачу, для предоставления общего видения, используется диаграмма классов ER диаграмма, USE case диаграмма,. Изображенных на рисунках 3.3.1, 3.3.2, 3.3.3.

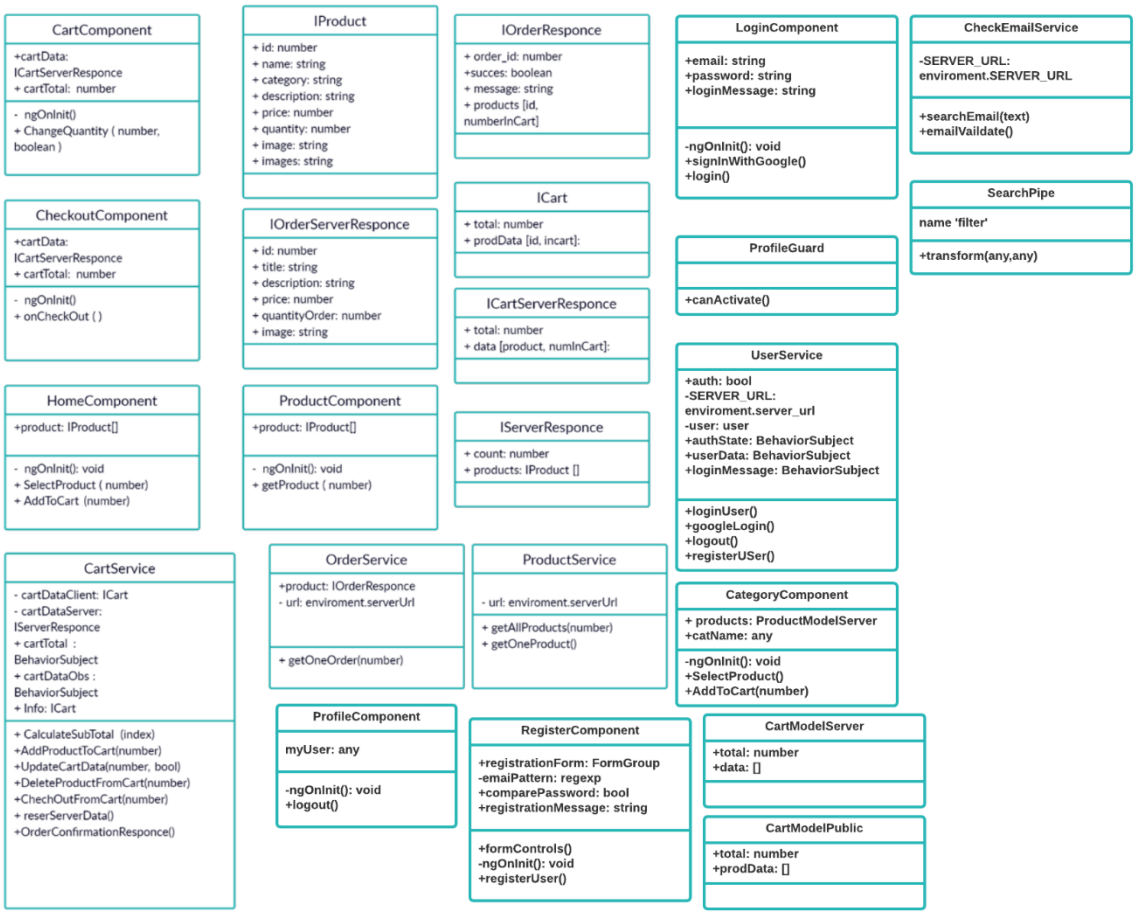


Рисунок-3.3.2 – Диаграмма классов

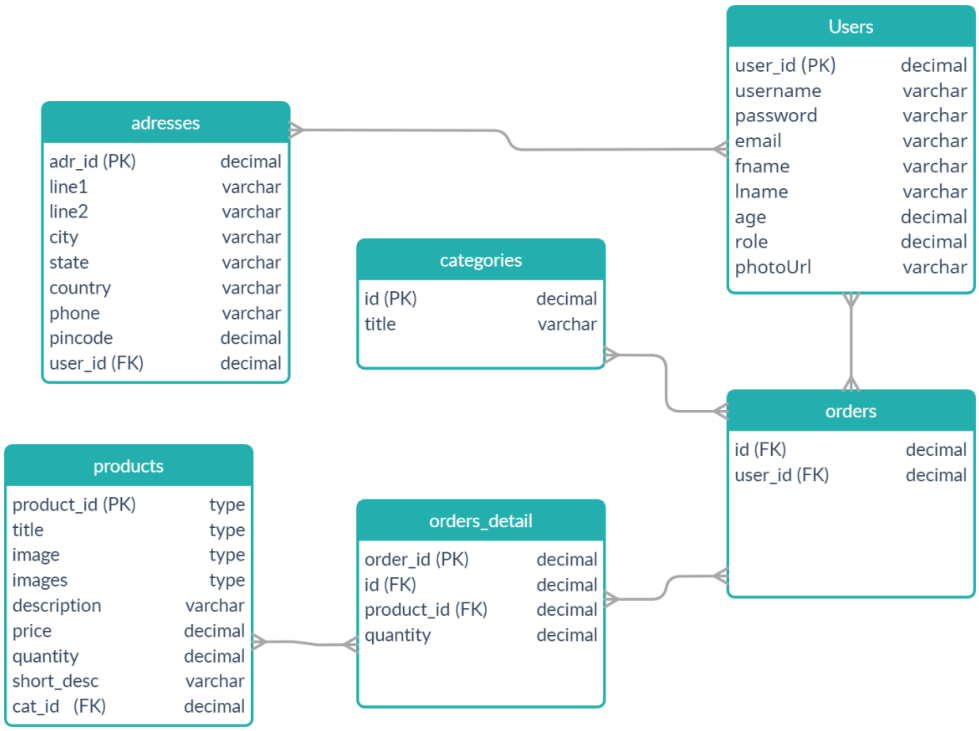


Рисунок-3.3.1 – ER диаграмма

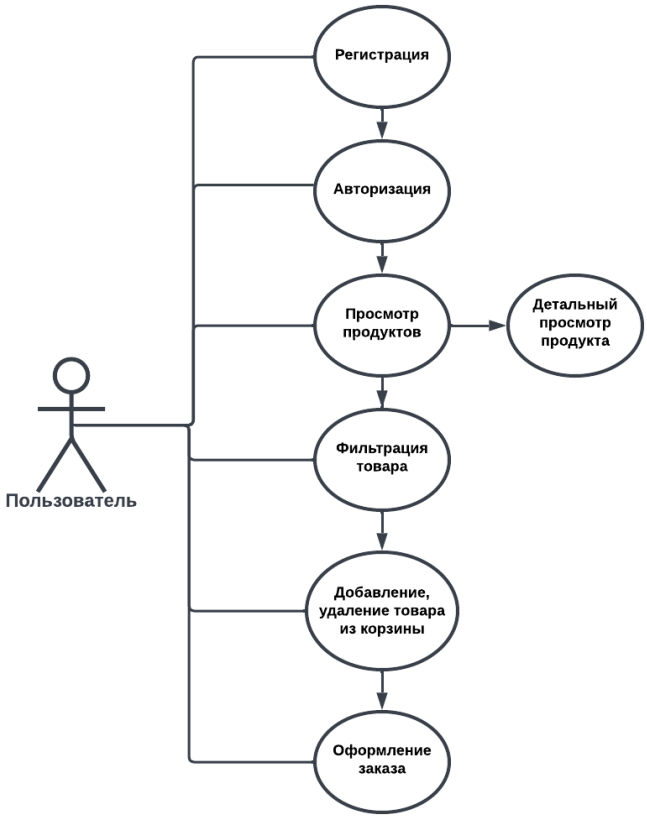


Рисунок-3.3.3 – USE case диаграмма

ЗАКЛЮЧЕНИЕ

В рамках данного дипломного проекта, поставленная цель – создания интернет-магазина на фреймворке Angular выполнена успешно, в результате которого было развернуто не большое клиент-серверная веб-платформа, которая показывает простоту создания сайта на передовых технологиях используя один из сложных фреймворков Angular, так же показана возможность интерпретации с backend частью и его технологий. Данный проект весомо поднял мои знания в frontend части и в целом дал понимание данного направления. В процессе написания проекта, подробно изучил систему контроля версий Git, узнал про паттерны архитектуры для создания веб системы, так же разобрался в тонкостях выбора оптимальных решений для коммуникации frontend и backend, в результате были выявлены наиболее удобные и быстрые . Разобрал функционал данного фреймворка, и подтянул свои знания. Данный проект имеет возможность дальнейшего масштабирования, добавления новой функциональности, так как сам фреймворк компонентно ориентирован.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1 Angular Framework introduction // Электронная версия на сайте https://angular.io/guide/

2 MySQL // Электронная версия на сайте https://ru.wikipedia.org/wiki/MySQL

3 Anna., Что Такое MySQL: Объяснение MySQL Для Начинающих // Электронная версия на сайте https://www.hostinger.com.ua/rukovodstva/shto-takoje-mysql/

4 IDE // Электронная версия на сайте https://www.jetbrains.com/ru-ru/

5 Alex Myzgin., Полное руководство по ECMAScript // Электронная версия на сайте https://frontend-stuff.com/blog/ecmascript/

6 Angular // Электронная версия на сайте https://blog.skillfactory.ru/glossary/angular/

7 DevilAngel., Нормализация отношений. Шесть нормальных форм // Электронная версия на сайте https://habr.com/ru/post/254773/

8 Node.js Express App // Электронная версия на сайте https://expressjs.com/ru/

9 Node.js // Электронная версия на сайте https://nodejs.org/en/docs/

10 Angular Material Design // Электронная версия на сайте https://material.angular.io/

11 Виды и особенности UML диаграмм // Электронная версия на сайте https://flexberry.github.io/ru/fd_editing-diagram.html

12 XAMPP // Электронная версия на сайте https://bit.ly/39xGEzN

13 Ангуляр: архитектурные паттерны и лучшие практики // Электронная версия на сайте https://dev-gang.ru/article/angular-arhitekturnye-patterny-i-luczshie-praktiki-dwlz213zt5/

14 Как организована веб-платформа. // Электронная версия на сайте https://bit.ly/3s61cWu

15 Веб приложение // Электронная версия на сайте https://bit.ly/3vIV8pj

16 8 лучших локальных серверов // Электронная версия на сайте https://timeweb.com/ru/community/articles/luchshie-lokalnye-servery

17 Статистика электронной коммерции и факты онлайн-покупок для 2022 // Электронная версия на сайте https://www.websiterating.com/ru/research/ecommerce-statistics-facts/

18 CSS фреймворк // Электронная версия на сайте https://bit.ly/38GNhiK

19 Методы HTTP – запроса // Электронная версия на сайте https://developer.mozilla.org/ru/docs/Web/HTTP/Methods

20 Двухфакторная аутентификация // Электронная версия на сайте https://www.aladdin-rd.ru/catalog/two-factor_authentication/

21 Авторизация с помощью Google Sign In // Электронная версия на сайте https://maxpfrontend.ru/vebinary/avtorizatsiya-s-pomoschyu-google-sign-in/

Приложение А
(обязательное)

Техническое задание

А.1.1 Техническое задание на разработку интернет-магазина с использованием Angular Framework

Настоящее техническое задание распространяется на разработку интернет-магазина с использованием Angular Framework, предназначенной для интернет-коммерции. Предполагается, что данный проект будет использоваться для интернет-коммерции, при развёртывании на домене и сервере, с дальнейшей масштабируемостью.

А.1.2 Основание для разработки

Веб-приложение разрабатывается на основе устного согласия научного руководителя выбранной дипломником темой.

А.1.3 Назначение

Веб-сайт предназначен для хранения, добавлении, обновлении, обработки информации, связанной с интернет-коммерцией.

А.1.4 Требования к функциональным характеристикам

С помощью языка TypeScript на фреймворке Angular, веб-сервере Node.js Express App, БД MySQL создать веб-сайт.

Должна использоваться реляционная БД для хранения, обработки информации. Веб-сайт может быть размещена локально, в дальнейшем может быть размещена на хостинге при покупке домена и сервера.

Веб-сайт должен обеспечить выполнение следующих функций:

- регистрация и авторизация пользователей;
- просмотр товаров;
- выбор категорий;
- фильтрация товара;

- просмотр товара;
- добавление, удаление товара из корзины;
- оформление заказа.

А.1.5 Требования к надежности

Обеспечить высокую скорость работы, оптимизировать взаимосвязи между технологиями, обеспечить конфиденциальность и сохранность пользовательских данных, использовать хэш функции для регистрации, ведение системы контроля версий, предусмотреть контроль вводимой информации с помощью валидации.

А.1.6 Требования к составу и параметрам технических средств

Веб-сайт должен работать на браузерах Google Chrome, Opera, Mozilla, Yandex, Safari. Адаптивно изменять размерность под разрешение дисплея.

А.1.7 Требования к информационной и программной совместимости

Веб-сайт должен поддерживать работу на операционных системах Windows, Linux, MacOS.

А.1.8 Требования к программной документации

Код проекта должен содержать в себе комментарии, возможность отслеживания изменений с помощью контроля версий.

Приложение Б
(обязательное)

Текст программы

home.component.ts

```
import { Component, OnInit } from '@angular/core';
import {ProductService} from '../../services/product.service';
import {Router} from '@angular/router';
import {ProductModelServer, ServerResponse} from '../../models/product.model';
import {CartService} from '../../services/cart.service';

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.scss']
})
export class HomeComponent implements OnInit {

  // like a local storage for data
  products: ProductModelServer[] = [];


  constructor(private productService: ProductService,
              private cartService: CartService,
              private router: Router) { }

  // get all product from DB, on loading the page
  ngOnInit(): void {
    this.productService.getAllProducts().subscribe((prods: ServerResponse) => {
      this.products = prods.products;
    });
  }

  // Select single product
  selectProduct(id: number) {
    this.router.navigate(['/product', id]).then();
  }

  // Add single product into cart
  AddToCart(id: number) {
```

```
    this.cartService.AddProductToCart(id);
     }
    }

    cart.component.ts

    import {Component, OnInit, Inject} from '@angular/core';
    import {CartModelServer} from '../../models/cart.model';
    import {CartService} from '../../services/cart.service';
    import { UserService } from './../../services/user.service';
    import { map } from 'rxjs/operators';

    @Component({
     selector: 'app-cart',
     templateUrl: './cart.component.html',
     styleUrls: ['./cart.component.scss']
    })
    export class CartComponent implements OnInit {
     cartData: CartModelServer;
     cartTotal: number;
     subTotal: number;
     constructor(public cartService: CartService) {
     }

     // get cart data on loading the page
     ngOnInit(): void {
      this.cartService.cartData$.subscribe((data: CartModelServer) =>
this.cartData = data);
      this.cartService.cartTotal$.subscribe(total => this.cartTotal = total);
     }

     // change product quantity from cart
     ChangeQuantity(index: number, increase: boolean) {
      this.cartService.UpdateCartItems(index, increase);
     }
    }

    login.component.ts

    import {Component, OnInit} from '@angular/core';
    import {NgForm} from '@angular/forms';
    import {AuthService} from 'angularx-social-login';
```

Продолжение приложение Б

```
import {ActivatedRoute, Router} from '@angular/router';
import {UserService} from '../../services/user.service';
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
})
export class LoginComponent implements OnInit {
  email: string;
  password: string;
  loginMessage: string;
  userRole: number;

  constructor(private authService: AuthService,
              private router: Router,
              private userService: UserService,
              private route: ActivatedRoute) {
  }
  // login state
  ngOnInit(): void {
    this.userService.authState$.subscribe(authState => {
      if (authState) {
        this.router.navigateByUrl(this.route.snapshot.queryParams.returnUrl      ||
'/profile');

      } else {
        this.router.navigateByUrl('/login');
      }
    });


  }

  // possibility to register with current google account
  signInWithGoogle() {
    this.userService.googleLogin();
  }

  // sign in site
  login(form: NgForm) {
    const email = this.email;
```

Продолжение приложение Б

```
 const password = this.password;

    if (form.invalid) {
      return;
    }
    form.reset();
    this.userService.loginUser(email, password);
    this.userService.loginMessage$.subscribe(msg => {
      this.loginMessage = msg;
      setTimeout(() => {
        this.loginMessage = '';
      }, 2000);
    });


  }
}
```

register.component.ts

```
import {Component, OnInit} from '@angular/core';
import {EmailValidator, FormBuilder, FormGroup, Validators} from '@angular/forms';
import {CheckEmailService} from '../../validators/check-email.service';
import {UserService} from '../../services/user.service';
import {map} from 'rxjs/operators';

@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.scss'],
  providers: [EmailValidator]
})
export class RegisterComponent implements OnInit {

  registrationForm: FormGroup;
  // regexp query for validate e-mail
  // tslint:disable-next-line:max-line-length
  private emailPattern = '(?:[a-z0-9!#$%&\'*+/=?^_`{|}~-]+(?:\\.[a-z0-9!#$%&\'*+/=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-
```

Продолжение приложение Б

```
\x7f]|\\\\[\x01-\x09\x0b\x0c\x0e-\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-
9])?\\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-
9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:(?:[\x01-
\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\\])';
        comparePassword: boolean;
        registrationMessage: string;

        constructor(private fb: FormBuilder,
                private checkEmailService: CheckEmailService,
                private userService: UserService) {
        // validator for registration form
          this.registrationForm = fb.group({
            fname: ['', [Validators.required, Validators.minLength(4)]],
            lname: ['', [Validators.required, Validators.minLength(4)]],
            email: ['', [Validators.required, Validators.pattern(this.emailPattern)],
              [this.checkEmailService.emailValidate()]
            ],
            password: ['', [Validators.required, Validators.minLength(6)]],
            confirmPassword: ['', [Validators.required, Validators.minLength(6)]],
          });
        }

        get formControls() {
          return this.registrationForm.controls;
        }
        ngOnInit(): void {
          this.registrationForm.valueChanges
            .pipe(map((controls) => {
              return this.formControls.confirmPassword.value ===
this.formControls.password.value;
            }))
            .subscribe(passwordState => {
              console.log(passwordState);
              this.comparePassword = passwordState;
            });
        }
        // register user, with add info into DB
        registerUser() {

          if (this.registrationForm.invalid) {
            return;
```

```
        }

          // @ts-ignore

this.userService.registerUser({...this.registrationForm.value}).subscribe((response:  {
message: string }) => {
            this.registrationMessage = response.message;
           });

           this.registrationForm.reset();
         }
        }

        product.service.ts

        import { Injectable } from '@angular/core';
        import {HttpClient} from '@angular/common/http';
        import {Router} from '@angular/router';
        import {environment} from '../../environments/environment';
        import {Observable} from 'rxjs';
        import {ProductModelServer, ServerResponse} from '../models/product.model';

        @Injectable({
          providedIn: 'root'
        })
        export class ProductService {
          private SERVER_URL = environment.SERVER_URL;
          constructor(private http: HttpClient) { }
          /* This is to fetch all products from the server */
          getAllProducts(numberOfResults= 10) : Observable<ServerResponse> {
            return this.http.get<ServerResponse>(this.SERVER_URL + '/products', {
              params: {
                limit: numberOfResults.toString()
              }
            });
          }

          /* GET SINGLE PRODUCT FROM SERVER*/
          getSingleProduct(id: number): Observable<ProductModelServer> {
            return this.http.get<ProductModelServer>(this.SERVER_URL + '/products/'
+ id);
          }
```

Продолжение приложение Б

```
  /*GET PRODUCTS FROM ONE CATEGORY */
  getProductsFromCategory(catName: string): Observable<ProductModelServer[]> {
    return this.http.get<ProductModelServer[]>(this.SERVER_URL + '/products/category/' + catName);
  }
}
```

cart.service.ts

```
import {Injectable} from '@angular/core';
import {HttpClient} from '@angular/common/http';
import {ProductService} from './product.service';
import {OrderService} from './order.service';
import {environment} from '../../environments/environment';
import {CartModelPublic, CartModelServer} from '../models/cart.model';
import {BehaviorSubject} from 'rxjs';
import {NavigationExtras, Router} from '@angular/router';
import {ProductModelServer} from '../models/product.model';
import {ToastrService} from 'ngx-toastr';
import {NgxSpinnerService} from 'ngx-spinner';

@Injectable({
  providedIn: 'root'
})
export class CartService {
  // local server url
  private serverURL = environment.SERVER_URL;

  // Data variable to store the cart information on the client's local storage
  private cartDataClient: CartModelPublic = {
    total: 0,
    prodData: [{
      incart: 0,
      id: 0
    }]
  };

  // Data variable to store cart information on the server
  private cartDataServer: CartModelServer = {
    total: 0,
    data: [{
```

```
    numInCart: 0,
    product: undefined
   }]
  };


  cartTotal$ = new BehaviorSubject<number>(0);
  cartData$ = new BehaviorSubject<CartModelServer>(this.cartDataServer);


  constructor(private http: HttpClient,
              private productService: ProductService,
              private orderService: OrderService,
              private router: Router,
              private toast: ToastrService,
              private spinner: NgxSpinnerService) {

   this.cartTotal$.next(this.cartDataServer.total);
   this.cartData$.next(this.cartDataServer);

   //  get the information from local storage
   const info: CartModelPublic = JSON.parse(localStorage.getItem('cart'));

   //  check if the info variable is null or has some data in it

   if (info !== null && info !== undefined && info.prodData[0].incart !== 0) {
    //  Local Storage is not empty and has some information
    this.cartDataClient = info;

    //  loop through each entry and put it in the cartDataServer object
    this.cartDataClient.prodData.forEach(p => {
      this.productService.getSingleProduct(p.id).subscribe((actualProductInfo:
ProductModelServer) => {
       if (this.cartDataServer.data[0].numInCart === 0) {
        this.cartDataServer.data[0].numInCart = p.incart;
        this.cartDataServer.data[0].product = actualProductInfo;
        this.CalculateTotal();
        this.cartDataClient.total = this.cartDataServer.total;
        localStorage.setItem('cart', JSON.stringify(this.cartDataClient));
       } else {
        // CartDataServer already has some entry in it
        this.cartDataServer.data.push({
```

Продолжение приложение Б

```
              numInCart: p.incart,
              product: actualProductInfo
            });
            this.CalculateTotal();
            this.cartDataClient.total = this.cartDataServer.total;
            localStorage.setItem('cart', JSON.stringify(this.cartDataClient));
          }
          this.cartData$.next({...this.cartDataServer});
        });
      });

    }

  }

  AddProductToCart(id: number, quantity?: number) {
    this.productService.getSingleProduct(id).subscribe(prod => {
      //  if the cart is empty
      if (this.cartDataServer.data[0].product === undefined) {
        this.cartDataServer.data[0].product = prod;
        this.cartDataServer.data[0].numInCart = quantity !== undefined ? quantity
: 1;
        this.CalculateTotal();
        this.cartDataClient.prodData[0].incart                                    =
this.cartDataServer.data[0].numInCart;
        this.cartDataClient.prodData[0].id = prod.id;
        this.cartDataClient.total = this.cartDataServer.total;
        localStorage.setItem('cart', JSON.stringify(this.cartDataClient));
        this.cartData$.next({...this.cartDataServer});
        this.toast.success(`${prod.name} added to the cart`, 'Product Added', {
          timeOut: 1500,
          progressBar: true,
          progressAnimation: 'increasing',
          positionClass: 'toast-top-right'
        });

      } else {
        const index = this.cartDataServer.data.findIndex(p => p.product.id ===
prod.id);  // -1 or a positive value

        //     if that item is already in the cart  =>  index is positive value
        if (index !== -1) {
```

Продолжение приложение Б

```
            if (quantity !== undefined && quantity <= prod.quantity) {
              this.cartDataServer.data[index].numInCart =
this.cartDataServer.data[index].numInCart < prod.quantity ? quantity : prod.quantity;
            } else {
              // tslint:disable-next-line:no-unused-expression
              this.cartDataServer.data[index].numInCart < prod.quantity ?
this.cartDataServer.data[index].numInCart++ : prod.quantity;
            }

            this.cartDataClient.prodData[index].incart =
this.cartDataServer.data[index].numInCart;
            this.CalculateTotal();
            this.cartDataClient.total = this.cartDataServer.total;
            localStorage.setItem('cart', JSON.stringify(this.cartDataClient));
            this.toast.info(`${prod.name} quantity updated in the cart`, 'Product
Updated', {
              timeOut: 1500,
              progressBar: true,
              progressAnimation: 'increasing',
              positionClass: 'toast-top-right'
            });

          } else {
            this.cartDataServer.data.push({
              numInCart: 1,
              product: prod
            });

            this.cartDataClient.prodData.push({
              incart: 1,
              id: prod.id
            });
            this.toast.success(`${prod.name} added to the cart`, 'Product Added', {
              timeOut: 1500,
              progressBar: true,
              progressAnimation: 'increasing',
              positionClass: 'toast-top-right'
            });

            this.CalculateTotal();
            this.cartDataClient.total = this.cartDataServer.total;
            localStorage.setItem('cart', JSON.stringify(this.cartDataClient));
```

Продолжение приложение Б

```
            this.cartData$.next({...this.cartDataServer});
          }  // END OF ELSE
        }
      });
    }

    // change product quantity from cart
    UpdateCartItems(index: number, increase: boolean) {
      const data = this.cartDataServer.data[index];

      if (increase) {
        data.numInCart < data.product.quantity ? data.numInCart++ :
data.product.quantity;
        this.cartDataClient.prodData[index].incart = data.numInCart;
        this.CalculateTotal();
        this.cartDataClient.total = this.cartDataServer.total;
        localStorage.setItem('cart', JSON.stringify(this.cartDataClient));
        this.cartData$.next({...this.cartDataServer});
      } else {
        data.numInCart--;

        if (data.numInCart < 1) {
          this.DeleteProductFromCart(index);
          this.cartData$.next({...this.cartDataServer});
        } else {
          this.cartData$.next({...this.cartDataServer});
          this.cartDataClient.prodData[index].incart = data.numInCart;
          this.CalculateTotal();
          this.cartDataClient.total = this.cartDataServer.total;
          localStorage.setItem('cart', JSON.stringify(this.cartDataClient));
        }
      }
    }

    // delete product from cart
    DeleteProductFromCart(index: number) {
      if (window.confirm('Are you sure you want to remove the item?')) {
        this.cartDataServer.data.splice(index, 1);
        this.cartDataClient.prodData.splice(index, 1);
        this.CalculateTotal();
        this.cartDataClient.total = this.cartDataServer.total;
```

Продолжение приложение Б

```
            if (this.cartDataClient.total === 0) {
              this.cartDataClient = {total: 0, prodData: [{incart: 0, id: 0}]};
              localStorage.setItem('cart', JSON.stringify(this.cartDataClient));
            } else {
              localStorage.setItem('cart', JSON.stringify(this.cartDataClient));
            }

            if (this.cartDataServer.total === 0) {
              this.cartDataServer = {total: 0, data: [{numInCart: 0, product: undefined}]};
              this.cartData$.next({...this.cartDataServer});
            } else {
              this.cartData$.next({...this.cartDataServer});
            }


          } else {
            return;
          }
        }

        // checkout order logic
        CheckoutFromCart(userId: number) {
          this.http.post(`${this.serverURL}/orders/payment`, null).subscribe((res: { success: boolean }) => {
            if (res.success) {

              this.resetServerData();
              this.http.post(`${this.serverURL}/orders/new`, {
                userId,
                products: this.cartDataClient.prodData
              }).subscribe((data: OrderResponse) => {
                this.orderService.getSingleOrder(data.order_id).then(prods => {
                  if (data.success) {
                    const navigationExtras: NavigationExtras = {
                      state: {
                        message: data.message,
                        products: prods,
                        orderId: data.order_id,
                        total: this.cartDataClient.total
                      }
                    };
```

Продолжение приложение Б

```
            this.spinner.hide().then();
            // could add e-pay transaction by Halyk Bank
            this.router.navigate(['/thankyou'], navigationExtras).then(p => {
              this.cartDataClient = {total: 0, prodData: [{incart: 0, id: 0}]};
              this.cartTotal$.next(0);
              localStorage.setItem('cart', JSON.stringify(this.cartDataClient));
            });
          }
        });
      });
    } else {
      this.spinner.hide().then();
      this.router.navigateByUrl('/checkout').then();
      this.toast.error(`Sorry, failed to book the order`, 'Order Status', {
        timeOut: 1500,
        progressBar: true,
        progressAnimation: 'increasing',
        positionClass: 'toast-top-right'
      });
    }
  });
}

// calculating product quantity in cart
private CalculateTotal() {
  let Total = 0;

  this.cartDataServer.data.forEach(p => {
    const {numInCart} = p;
    const {price} = p.product;

    Total += numInCart * price;
  });
  this.cartDataServer.total = Total;
  this.cartTotal$.next(this.cartDataServer.total);
}

// reset cart data
private resetServerData() {
  this.cartDataServer = {
    total: 0,
    data: [{
```

```
        numInCart: 0,
        product: undefined
      }]
    };

    this.cartData$.next({...this.cartDataServer});
  }

  CalculateSubTotal(index): number {
    let subTotal = 0;

    const p = this.cartDataServer.data[index];
    // @ts-ignore
    subTotal = p.product.price * p.numInCart;

    return subTotal;
  }
}


interface OrderResponse {
  order_id: number;
  success: boolean;
  message: string;
  products: [{
    id: string,
    numInCart: string
  }];
}
```