МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки і програмної інженерії
Кафедра інженерії програмного забезпечення

"ДОПУСТИТИ ДО ЗАХИСТУ"
Завідувач кафедри
_______ Катерина НЕСТЕРЕНКО
(підпис) (ім'я, прізвище)

"____"_____________2023 р.

ДИПЛОМНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

Тема: "Вебзастосунок для електронної комерції з використанням вебзбирання з оптимізованим процесом зчитування товарів"

Виконавець:	ст. гр. ПІ-221М(А) Коваленко Ілля Ігорович
Керівник:	к.т.н., доцент Терещенко Лідія Юріївна
Нормоконтролер:	к.т.н., с.н.с., доцент Оленін Михайло Вікторович

Засвідчую, що у дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____________
(підпис)

КИЇВ 2023

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
NATIONAL AVIATION UNIVERSITY
Faculty of cybersecurity and software engineering
Software engineering department

"ADMIT TO DEFENCE"
Head of the department
_______ Kateryna NESTERENKO
(signature) (name, surname)

"____"_____________2023

MASTER'S THESIS
(EXPLANATORY NOTE)

OF A MASTER'S DEGREE GRADUATE

Topic: "An e-commerce web application using web scraping with an optimized product reading process"

Performer: Student of the PI-221M(A) group, Kovalenko Illia Ihorovych

Supervisor: Candidate of Technical Sciences, Associate Professor Tereshchenko Lydia Yuriivna

Standard controller: Candidate of Technical Sciences, Senior Researcher, Associate Professor, Olenin Mykhailo Viktorovych

KYIV 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет	кібербезпеки і програмної інженерії
Кафедра	інженерії програмного забезпечення
Освітній ступінь	магістр
Спеціальність	121 Інженерія програмного забезпечення
Освітньо-професійна програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
_______ Катерина НЕСТЕРЕНКО
(підпис) (ім'я, прізвище)
"___" _________ 2023 р.

ЗАВДАННЯ
на виконання дипломної роботи
Коваленка Іллі Ігоровича

1. Тема дипломної роботи: "Вебзастосунок для електронної комерції з використанням вебзбирання з оптимізованим процесом зчитування товарів". Затверджена наказом ректора від 29.09.2023 р. № 1994/ст.

2. Термін виконання роботи: з 02.10.2023 р. до 31.12.2023 р.

3. Вихідні дані до роботи: вебзастосунок для електронної комерції.

4. Зміст пояснювальної записки:

- Аналіз вимог до програмного забезпечення: загальні положення, змістовний опис предметної області, змістовний аналіз предметної області, аналіз успішних ІТ-проєктів, аналіз вимог до програмного забезпечення, постановка комплексу завдань модулю, висновки по розділу;
- Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, обґрунтування засобів розробки, архітектура бази даних, архітектура програмного забезпечення, конструювання програмного забезпечення, аналіз безпеки даних, висновки по розділу;
- Аналіз якості та тестування програмного забезпечення: аналіз якості ПЗ, опис процесів тестування, опис контрольного прикладу, висновки по розділу;
- Впровадження та супровід програмного забезпечення: розгортання програмного забезпечення, робота з програмним забезпеченням, висновки по розділу;
- Керівництво користувача: загальні відомості, підготовка до роботи, робота з додатком, висновки по розділу.

5. Перелік обов'язкового ілюстративного матеріалу (слайдів презентації):

- Схема структурна варіантів використань;
- Схема бази даних;
- Схема структурна класів програмного забезпечення;
- Схема структурна компонентів програмного забезпечення.

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Відмітка про виконання
1.	Узгодження технічного завдання з керівником дипломної роботи	02.10.2023 – 18.10.2023	
2.	Вивчення рекомендованої літератури	19.09.2023 – 23.09.2023	
3.	Аналіз існуючих методів розв’язання задачі	24.09.2023 – 30.09.2023	
4.	Постановка та формалізація задачі	01.10.2023 – 05.10.2023	
5.	Розробка інформаційного забезпечення	06.10.2023 – 15.10.2023	
6.	Алгоритмізація задачі	16.10.2023 – 20.10.2023	
7.	Обґрунтування вибору використаних технічних засобів	21.10.2023 – 27.10.2023	
8.	Розробка програмного забезпечення	28.10.2023 – 22.11.2023	
9.	Налагодження програми	23.11.2023 – 25.11.2023	
10.	Виконання графічних документів	25.11.2023 – 31.11.2023	
11.	Оформлення пояснювальної записки	01.12.2023 – 03.12.2023	
12.	Завершення написання ПЗ. Проходження нормоконтролю. Друк ПЗ. Отримання відгуку керівника. Підготовка презентації та доповіді на перед захист	04.12.2023 – 10.12.2023	
13.	Передзахист кваліф. роботи. Отримання рецензії	11.12.2023 – 17.12.2023	
14.	Підготовка документів до захисту та здача їх секретарю ДЕК	18.12.2023 – 24.12.2023	
15.	Захист кваліф. роботи	25.12.2023 – 31.12.2023	

7. Дата видачі завдання: 02.10.2023 р.

Керівник дипломної роботи: __________ Лідія ТЕРЕЩЕНКО
(підпис) (ім’я, прізвище)

Завдання прийняв до виконання: __________ Ілля КОВАЛЕНКО
(підпис) (ім’я, прізвище)

NATIONAL AVIATION UNIVERSITY

Faculty of	Cybersecurity and Software Engineering
Department	Software Engineering
Education degree	Master
Specialty	121 Software Engineering
Educational-professional program	Software Engineering

APPROVED BY
Head of the department
_______ Kateryna NESTERENKO
(signature) (name, surname)
"___" ___________ 2023

Task
on executing the graduation work
Kovalenko Illia Ihorovych

1. Topic of the graduation work: "An e-commerce web application using web scraping with an optimized product reading process". Approved by the order of the rector from 29.09.2023 р. № 1994/ст.
2. Terms of work execution: from 02.10.2023 to 31.12.2023
3. Source data of the work: an e-commerce web application.
4. Content of the explanatory note:

- Analysis of software requirements: general regulations, comprehensive description of the subject area, comprehensive analysis of the subject area, analysis of successful IT projects, analysis of software requirements, setting the set of module tasks, conclusions on the section;
- Software modeling and design: software modeling and analysis, rationale for development tools, database architecture, software architecture, description of software architecture, data security analysis, conclusions on the section;
- Quality analysis and software testing: software quality analysis, description of testing processes, description of the test case, conclusions on the section;
- Software implementation and maintenance: software deployment, working with the software, conclusions on the section;
- User guide: general information, preparation for work, working with the application, conclusions on the section.

5. List of presentation mandatory slides:

- Structural scheme of use cases;
- Database schema;
- Structural scheme of software classes;
- Diagram of the structural components of the software.

6. Calendar schedule

#	Task	Execution term	Execution mark
1.	Coordination of the technical task with the thesis supervisor	02.10.2023 – 18.10.2023	
2.	Study of recommended literature	19.09.2023 – 23.09.2023	
3.	Analysis of existing problem-solving methods	24.09.2023 – 30.09.2023	
4.	Formulation and formalization of the problem	01.10.2023 – 05.10.2023	
5.	Development of information support	06.10.2023 – 15.10.2023	
6.	Algorithmization of the problem	16.10.2023 – 20.10.2023	
7.	Justification of the choice of the used technical means	21.10.2023 – 27.10.2023	
8.	Software development	28.10.2023 – 22.11.2023	
9.	Debugging the program	23.11.2023 – 25.11.2023	
10.	Execution of graphic documents	25.11.2023 – 31.11.2023	
11.	Issuance of an explanatory note	01.12.2023 – 03.12.2023	
12.	Completion of writing the software. Passing standard control. Software printing. Receiving feedback from the supervisor. Preparation of a presentation and a report for the defense	04.12.2023 – 10.12.2023	
13.	Preliminary defense of qualification work. Receiving the review	11.12.2023 – 17.12.2023	
14.	Preparation of documents for defense and their submission to the secretary	18.12.2023 – 24.12.2023	
15.	Defense of the qualification work	25.12.2023 – 31.12.2023	

Date of issue of the assignment: 02.10.2023

Supervisor: __________ (signature) Lydia TERESHCHENKO (name, surname)

Task accepted for execution: __________ (signature) Illia KOVALENKO (name, surname)

РЕФЕРАТ

Пояснювальна записка дипломного проєкту має чотири розділи, у яких знаходиться 13 таблиць, 54 рисунка та 44 джерела, загалом 101 сторінка.

Дипломний проєкт фокусується на розробці вебзастосунку для електронної комерції з використанням вебзбирання з оптимізованим процесом зчитування товарів.

Предмет дослідження: методи створення захищених вебзастосунків.

Об'єкт дослідження: процес розробки вебзастосунку для електронної комерції з використанням вебзбирання з оптимізованим процесом зчитування товарів.

Мета дипломного проєкту: створення вебзастосунку для електронної комерції з використанням вебзбирання з оптимізованим процесом зчитування товарів, а також шаблон конвеєру обробки замовлень та динамічний інтерфейс.

У першому розділі проводиться аналіз вимог до програмного забезпечення, представлено загальний зміст та опис предметної області, аналіз успішних IT-проєктів та вимог до програмного забезпечення, постановка завдань для модуля.

Другий розділ присвячений моделюванню та створенню програмного забезпечення, включаючи розробку архітектури та структури бази даних, а також огляд архітектури програмного забезпечення, представлення UML-діаграм.

У третьому розділі розглядається аналіз якості та тестування програмного забезпечення, включаючи розробку плану, оцінку якості програми, опис процесів тестування та результати тестів з поданням контрольного прикладу.

Четвертий розділ присвячений впровадженню та супроводженню програмного забезпечення, описує розгортання програми та використання програмного забезпечення разом з узагальненими висновками.

У п’ятому розділі детально описано покрокові інструкції щодо навігації та використання програми динамічної електронної комерції.

КЛЮЧОВІ СЛОВА: ВЕБЗАСТОСУНОК, CMS, ЕЛЕКТРОННА КОМЕРЦІЙНА ПЛАТФОРМА, ВЕБЗБИРАННЯ, ШАБЛОН КОНВЕЄРУ ОБРОБКИ ЗАМОВЛЕНЬ, ТРЬОХ-ШАРОВА АРХІТЕКТУРА, AWS, DOCKER.

ABSTRACT

The explanatory note of the diploma project consists of four sections, containing 13 tables, 54 figures, and 44 sources, totaling 101 pages.

The diploma project is focused on the development of an e-commerce web application using web scraping with an optimized product reading process.

The subject of the research: methods of creating secure web applications.

The object of the research: the process of developing an e-commerce web application using web scraping with an optimized product reading process.

The goal of the diploma project: the creation of an e-commerce web application using web scraping with an optimized product reading process, along with a base for an order processing conveyor framework and a dynamic interface.

The first section includes an analysis of software requirements, a general overview and description of the subject area, an analysis of successful IT projects and software requirements, the definition of tasks for the module, and general conclusions.

The second section is dedicated to software modeling and construction, covering the development of software architecture and database structure, as well as an overview of the software architecture. It includes UML diagrams and general conclusions.

The third section discusses software quality analysis and testing, which involves developing a plan, assessing software quality, describing testing processes, presenting test results with a sample case, and general conclusions.

The fourth section focuses on software deployment and maintenance, providing information about program deployment, software usage, and general conclusions.

The fifth section, the step-by-step instructions for navigating and utilizing the dynamic e-commerce application are detailed.

KEYWORDS: WEB APPLICATION, CMS, E-COMMERCE PLATFORM, WEB SCRAPING, BASE FOR ORDER PROCESSING, THREE-TIER ARCHITECTURE, AWS, DOCKER.

TABLE OF CONTENTS

LIST OF ACRONYMS AND ABBREVIATIONS

Stripe – is a popular payment system and a service for money transactions through an API service;

CMS – Content Management System;

CMA – is a graphical user interface subordinate to a CMS;

A CDA – is a graphical interface that provides internal content management and delivery support services, subordinate to a CMS;

PHP – stands for Hypertext Preprocessor, a scripting programming language;

XML – Extensible Markup Language;

OOP – stands for Object-Oriented Programming;

CRUD – Create Read Update Delete, the 4 main data management functions of “create, read, update, and delete”;

DB – Database;

OS – Operating System;

RAM – Random Access Memory;

SDLC – stands for Software Life Cycle;

CVV2 – Card Verification Value 2, a three-digit code for checking the validity of the payment system card;

MVC – stands for Model-View-Controller, an architectural pattern that divides a program into three main components, a model, a view, and a controller;

AWS – Amazon Web Services;

HTTP – stands for Hypertext Transfer Protocol;

API – Application Programming Interface;

JSON – stands for JavaScript Object Notation, which is a text format for data exchange between computers;

Token – is a software token that is issued to the user after successful authorization and is the key to access services.

INTRODUCTION

Currently, e-commerce web applications for selling products are gaining increasing popularity due to the rapid growth of commercial enterprises. This surge has created a significant demand for e-commerce platforms, making them indispensable in the modern e-commerce industry.

Typically, developing and managing web applications for commercial purposes requires meticulous attention to detail. Every commercial business aspires to have its personalized e-commerce platform that meets the needs and demands of its customers.

One of the possible solutions to content management challenges is the search for a ready-made web application for selling products. However, this approach often presents issues related to deployment complexity, unclear interfaces, automation of product creation and order processing, as well as the high cost of independently compiled software modules. It is important to note that even ready-made solutions have their limitations and drawbacks.

Thus, the primary objective of this diploma project is to implement automation of product creation processes through web scraping, create a base for developing a framework for order processing, and create a dynamic user interface. This will help minimize the shortcomings associated with existing solutions.

The tasks of this project include the development of a web application for content management and product sales, utilizing network data retrieval from internet stores, and implementing a base for an order processing conveyor framework.

Therefore, this diploma work aims to create a software application that not only addresses the numerous shortcomings of existing alternatives but also introduces new features to make it attractive in the e-commerce web applications market.

1 ANALYSIS OF SOFTWARE REQUIREMENTS

1.1 General regulations

E-commerce web applications for selling products have become a popular niche for commercial companies in need of creating and utilizing e-commerce platforms.

Typically, the development of e-commerce web applications for commercial purposes demands meticulous attention to implementation details, both from developers and clients. Since each commercial business is unique and has its distinct characteristics, its e-commerce platform must reflect these individual traits.

There are numerous approaches to address the challenges of creating and using e-commerce web applications, often involving the search for ready-made web applications. However, when using ready-made solutions, there can be both advantages and risks that may impact the functioning of a commercial enterprise in the long run. It is important to understand that even ready-made solutions have their limitations and potential drawbacks.

Therefore, this work is aimed at analyzing and implementing innovations aimed at improving the functionality of the e-commerce web application in this domain. In summary, these innovations encompass the automation of product creation processes through data retrieval from the network, the development of a base for an order processing framework, and enhancements to the user interface.

1.2 Comprehensive description of the subject area

The e-commerce web application using web scraping with an optimized product reading process allows users to create personalized e-commerce platforms with built-in functionality and addresses the challenges of automating product creation through data retrieval from internet stores. It also incorporates a base for an order processing conveyor framework and dynamic design capabilities with style variation. Additionally, the web application handles order payment transactions using the “Stripe” technology.

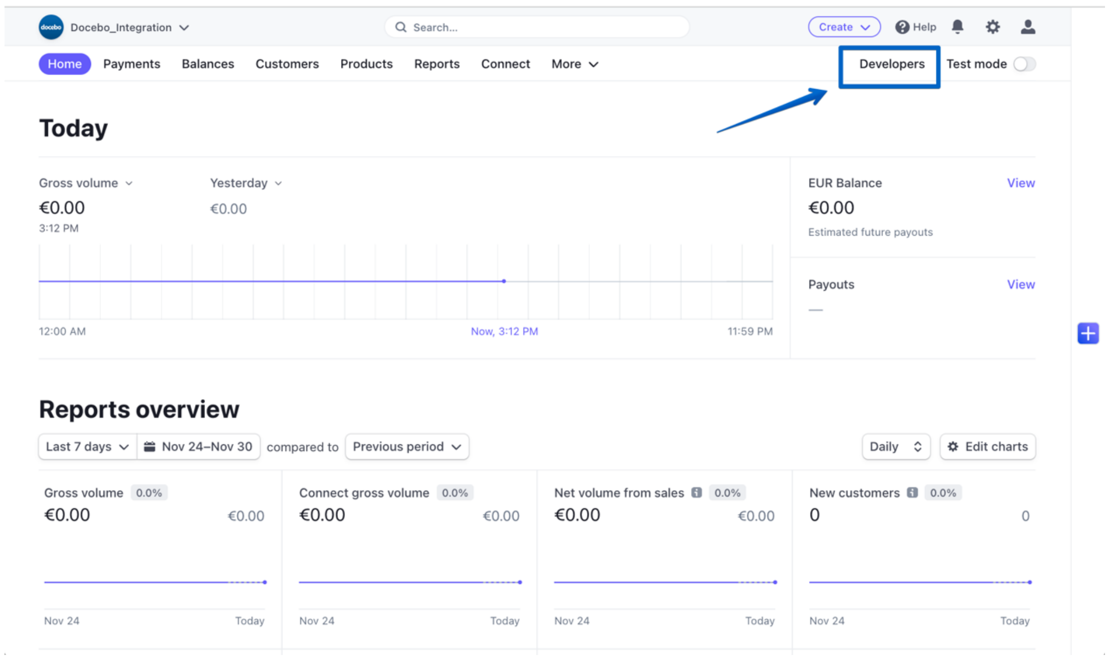


Fig. 1.1. Stripe dashboard

The participants in the web application include users representing the application’s clients, as well as administrators and managers responsible for application administration. The distribution of roles among users has been implemented to ensure a clear delineation of responsibilities and to protect the application from unauthorized access.

User convenience in the web application includes:

– An intuitive and dynamic user interface;

– The ability to view products with detailed descriptions;

– Adding products to a cart that is stored throughout the user’s session due to built-in session storage;

– A convenient order payment process facilitated by the use of the “Stripe” payment processing service;

– The option to change the application’s background.

Administrator convenience in the web application encompasses:

– A built-in administration panel consisting of two subpanels: “e-commerce” and “administrator’s menu”;

– Functions for managing products, generating text files with product data extracted from internet stores, operating the order processing conveyor framework, and managing inventory and its items.

Thus, this work implements a unique e-commerce web application using web scraping with an optimized product reading process, a base for an order processing conveyor framework, and a dynamic user interface.

1.3 Comprehensive analysis of the subject area

The e-commerce web application using web scraping with an optimized product reading process, as well as a base for an order processing conveyor framework, encompasses the concept of CMS and its components. It also includes the concept of an e-commerce platform and its types. These concepts will be further explored for a more in-depth understanding of the environment.

Fig. 1.2. CMS variety

1.3.1 Definition of a content management system

A content management system, or CMS, is software that allows users to create, edit, collaborate, publish, and store digital content. Content in a CMS is stored in a database and displayed in the software's presentation layer based on a set of templates.

A CMS consists of two components:

– Content management application or CMA: This is a graphical user interface that enables users to design, create, modify, and delete website content without the need for HTML expertise;

– Content Delivery Application or CDA: This comprises internal services that provide support for content management and delivery.

Content management systems or web applications based on them have the following features:

– Workflow processes, such as assigning permissions for content management based on roles like authors, editors, and administrators;

– Content creation, including the ability for users to create and format content;

– Content storage in a database, ensuring that content is consistently stored in one place;

– Providing access to multiple users, with each CMS having unique user permissions or defined roles such as editor, manager, author, or administrator [1].

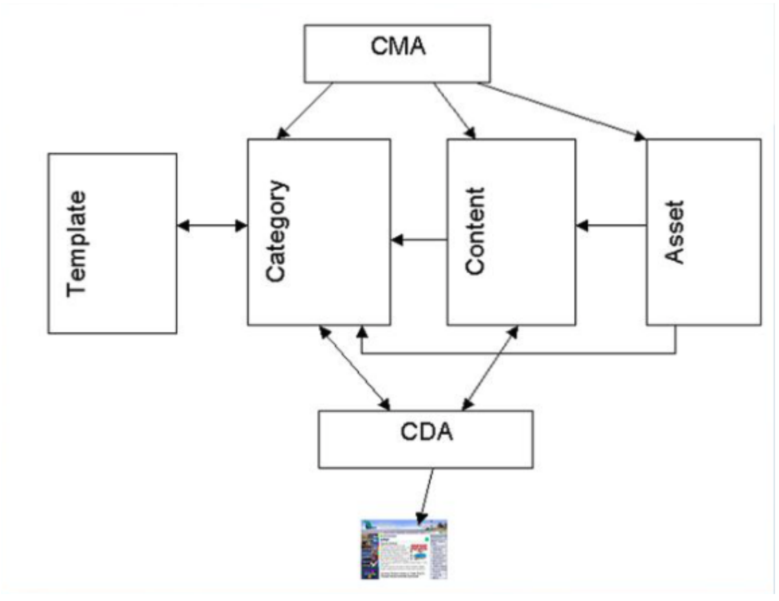


Fig. 1.3. CMA and CDA interactions

1.3.2 E-commerce platform

An e-commerce platform is a comprehensive software solution that provides internet stores with the means to manage their business. This type of service encompasses the creation of e-commerce websites, inventory, and stock management systems, as well as customer service infrastructure [2].

There are three main types of e-commerce:

– Business-to-Business (B2B): This involves internet sales from one business to another, essentially wholesale trade;

– Business-to-Consumer (B2C): Currently, a significant portion of e-commerce falls into this category as companies find it easy to target specific consumers online, display their products on the internet, and offer consumers the ability to make purchases without leaving their homes, thus saving time;

– Consumer-to-Consumer (C2C): This is a business model in which consumers facilitate transactions for products or services between private individuals, without the involvement of a primary business at any stage of the sale.

1.3.3 Web scraping

Network data retrieval, also known as “web scraping” is the process of automated collection of structured web data, in other words, extracting information from the web. This data is gathered and then exported in a format that is more convenient for the user.

Typically, network data retrieval is carried out using software or libraries that perform web scraping to extract specific pieces of information from various websites. Some common use cases for web scraping include price monitoring, price analysis, news tracking, lead generation, and market research, among many others [3].

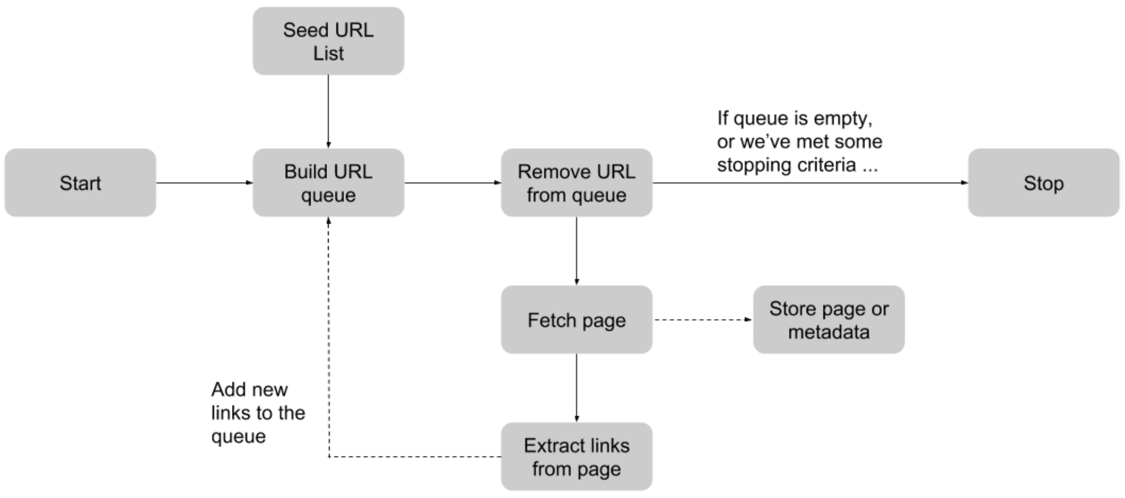


Fig. 1.4. Abstract web scraping core functionality

1.3.4 Platform Camunda

The Camunda platform is a lightweight framework based on the Java programming language. The development process consists of two parts: creating a “flow process” in the specialized Camunda Modeler tool and writing “Java code” to handle the steps of the process defined in the diagram. The platform provides developers with REST APIs and specialized client libraries to create programs that can interact with the remote workflow processing mechanism [4].

1.4 Analysis of successful IT projects

1.4.1 Analysis of known technical solutions

After analyzing the literature related to this subject area, it can be stated that freely available ready-made e-commerce web applications have their advantages and disadvantages.

This work aims to create the implementation of the e-commerce web application, addressing specific shortcomings of existing e-commerce web applications related to the automation of administrator actions. Specifically, the web application introduces the functionality of network data retrieval from internet stores, adds a base for an order processing framework, and provides a dynamic user interface.

1.4.2 Analysis of known software products

The demand for content management systems or web applications for content management is steadily increasing day by day. This growth is so significant that over 60% [5] of all websites today operate on various types of content management systems or platforms, and web applications.

There are numerous ready-made content management systems for selling products that have gained popularity among commercial businesses. Among such products, we can mention WordPress, Drupal, Joomla, and Magento. Let’s explore their definitions and features further.

WordPress is one of the most popular content management systems in the world. Approximately 43.3% [6] of all websites on the internet are powered by WordPress CMS. It is free to download and use, easy to learn, and search engine optimized. Additionally, it includes thousands of available themes and plugins in one repository, making it one of the most customizable platforms [7].

Drupal is an open-source content management system. It is written in the PHP programming language and distributed under the GNU General Public License. Drupal also includes modules, themes, JavaScript, CSS, and image files. This system helps create various web projects using its template resource, which contributes to convenient outcomes [8].

Joomla is a free content management system written in PHP and JavaScript. Joomla CMS uses MySQL and MS SQL databases for data storage [9].

Magento is an e-commerce platform built on PHP and XML technologies. It is a popular open-source content management system specialized in creating internet stores. This CMS primarily utilizes PHP and Zend Framework technologies [10].

Below is Table 1.1 with a comparison of existing content management systems and their drawbacks. This comparison was conducted based on a source that includes an analysis of content management systems [11].

Table 1.1.

Comparison of existing content management systems

Characteristic	WordPress	Drupal	Joomla	Magento
Use	Arbitrary	Arbitrary	Arbitrary	Arbitrary
Number of free templates	> 4000	> 1000	> 2000	> 1000
Number of free plugins	> 50000	> 37000	>7000	>3000
One-click installation	Supports	Supports	Supports	Supports
Loading Time	5 minutes	10 minutes	10 minutes	10 minutes

Table 1.1. (continue)

Documentation	Present in different languages	Present in English	Present in different languages	Present in English
Security system with “patches”	High level of security	High level of security	High level of security	High level of security
Language support	Present	Present	Present	Present
Visual Editor	There is a partial visual editor when buying a plugin	There is a partial visual editor	No visual editor	There is a complete visual editor with plugins
Basket	N/a	N/a	N/a	Present
Price	Missing	Missing	Missing	Missing

1.5 Analysis of software requirements

The main goal of the developed software is to create an e-commerce web application using web scraping with an optimized product reading process from internet stores and a base for an order processing conveyor framework for global use.

Additionally, the software should feature an intuitive dynamic user interface and implement a financial transaction service.

To achieve this goal, the software must meet the following requirements:

– Development using fundamental principles of object-oriented programming (OOP) and modern programming patterns;

– Support for web standards and access control to the application through user authentication and authorization methods with different roles;

– An administrator role responsible for CMS management and a client role that can view products, the customer’s cart, and create and pay for orders;

– Implementation of an admin panel and manager for process and application configuration management;

– Automation of product creation processes through network data retrieval from internet stores and the establishment of a base for an order processing conveyor framework;

– Implementation of personalized CRUD (Create, Read, Update, Delete) operations for content management and data storage in the database;

– Calculation of the total price of products in an order;

– Integration of the “Stripe” financial transaction service with an improved transaction interface;

– An intuitive and dynamic interface using Bulma and Vue.js program solutions infrastructure, adaptable for computers, tablets, and mobile devices.

In summary, the main purpose of the software requirements is to develop high-quality software that satisfies real customer needs within the budget.

1.5.1 Additional software requirements

To reduce the potential number of errors in the database, validation of user information input forms and all financial transactions has been implemented. Validation will contribute to the quality storage of data in the database table.

To distinguish between the administrative panel and the store menu, two menus have been created: the “Store Menu” and the “Administrator Menu”. This solution addresses the separation of functions between the store and the administration of the e-commerce web application.

When creating a product, the price will be calculated with precision to the tenth place and rounded during input. The application will include one administrator with a default login and password, and the administrator will have the ability to create new users in the administrative panel. Also, during the development status, the web application will include default user data for order creation. The e-commerce web application should have an authentication process based on sessions and cookie files with a twenty-minute time limit.

To simplify the order processing process, it should be divided into three stages of management and processing: order formation, order packaging, and order shipping. In the future, the “Camunda” process automation technology will be integrated into the order processing conveyor framework, and customers will have access to view processing stages.

1.5.2 Data storage requirements

The data storage for the e-commerce web application using web scraping with an optimized product reading process should be integrated into the project’s data layer, along with the generation of migration folders. It should also ensure high-performance data writing and be compatible with various operating systems.

1.5.3 Database software requirements

The requirements for the database software for the e-commerce web application using web scraping with an optimized product reading process are as follows:

- Usage of operating systems like Linux, Windows;
- Recommended minimum of 4 CPU cores;
- 4 GB of RAM or more.

These requirements define the software requirements for the database. The project’s data storage should maintain appropriate relationships between information and data.

1.5.4 Software deployment requirements

To run and use this project, the following system and hardware requirements are needed:

– Minimum 1 GB of RAM;
– Support for operating systems such as Windows, MacOS, and Linux;
– Installed Visual Studio, preferably version 2016 or higher;
– At least 8 GB of physical memory;
– Processor frequency of 1 GHz or higher;
– Broadband internet connection.

1.5.5 Development of functional requirements

Based on the use case diagram presented in Fig. 2.11. titled “Structural scheme of use cases” the following use cases and functional requirements for this web application have been developed, as outlined in Table 1.2.

Table 1.2.

Application use cases and functional requirements

UC1.1.1 Use Case	
Name	Authorization
Description	The administrator enters the username and password, then goes to the administration panel
Participants	Administrator
Prerequisites	The web application is running, authorization fails
Postconditions	The administrator is authorized in the application
UC1.1.2 Use Case	
Name	Authentication
Description	For the duration of the usage session, the web app stores session tokens stored in the app for 20 minutes
Participants	Administrator, User
Prerequisites	Running web application

Postconditions	Saving authorized data for 20 minutes in the app
UC1.1.3 Use Case	
Name	Access to the admin panel
Description	The administrator can go to the administrative panel
Participants	Administrator
Prerequisites	The web application is running, the administrator is authorized
Postconditions	Successful transition and access to the admin panel
UC1.1.4 Use Case	
Name	View and edit products
Description	The administrator chooses to view, edit, create, and delete products

Table 1.2. (continue)

Participants	Administrator
Prerequisites	Running web application, access to the administrative panel
Postconditions	A page with the functionality of viewing and editing products is displayed
UC1.1.5 Use Case	
Name	Generating a product list file from internet stores
Description	The administrator generates a text file with a sheet of product data read from internet stores
Participants	Administrator
Prerequisites	Running web application, access to the administrative panel
Postconditions	A page with the functionality for generating a product list file is displayed
UC1.1.6 Use Case	
Name	View and edit the order processing process
Description	The administrator has access to the order processing panel and its editing
Participants	Admin access to the admin panel
Prerequisites	Running web application, access to the administrative panel

Postconditions	A page with order processing functionality is displayed
UC1.1.7 Use Case	
Name	View and edit the number of products in stock
Description	The administrator selects viewing, editing, creating, and deleting the quantity of products in the warehouse
Participants	Administrator
Prerequisites	Running web application, access to the administrative panel
Postconditions	A page with warehouse editing functionality is displayed
UC1.1.8 Use Case	
Name	Creating new users

Table 1.2. (continue)

Description	The administrator creates new users in the application with naming and access rights
Participants	Administrator
Prerequisites	Running web application, access to the administrative panel
Postconditions	A page with user creation functionality is displayed
UC1.1.9 Use Case	
Name	Product Page View
Description	The user has access to view the product page
Participants	User
Prerequisites	Running web application, home page
Postconditions	Displayed product page
UC1.1.10 Use Case	
Name	Editing a product in the cart
Description	The user adds, subtracts, removes items from the cart
Participants	User
Prerequisites	Running web application, cart page
Postconditions	The cart page is displayed, it is possible to edit it
UC1.1.11 Use Case	

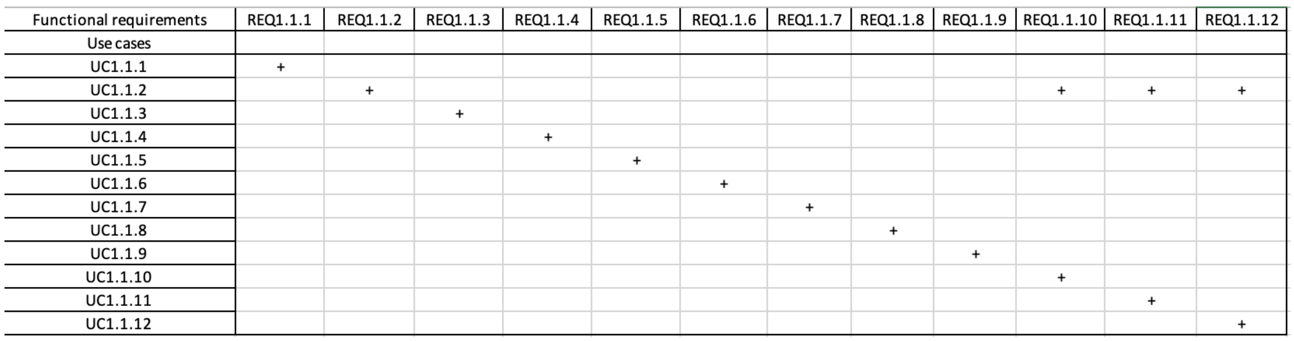
Name	Checkout
Description	The user fills in the fields of the order delivery address and pays for the order on a new page
Participants	User
Prerequisites	Running web application, checkout pages
Postconditions	Order confirmation page displayed
UC1.1.12 Use Case	
Name	Changing the background of the app
Description	Both actors can choose a personalized app background for the product page

Table 1.2. (continue)

Participants	Administrator, User
Prerequisites	Running web application, home page, menu
Postconditions	The main page is displayed with a redesigned interface

Based on the use cases, functional requirements for the e-commerce web application using web scraping with an optimized product reading process were created and described. These requirements are listed in Table 1.2 and are subject to the description section. It’s important to note that each requirement is assigned an identification number, introduced in parallel with the use cases, and starts with the prefix “REQ.”

As a result, using the discussed use cases and functional requirements, a requirements traceability matrix was developed, which can be viewed in Fig. 1.5.



Functional requirements	REQ1.1.1	REQ1.1.2	REQ1.1.3	REQ1.1.4	REQ1.1.5	REQ1.1.6	REQ1.1.7	REQ1.1.8	REQ1.1.9	REQ1.1.10	REQ1.1.11	REQ1.1.12
Use cases												
UC1.1.1	+											
UC1.1.2		+								+	+	+
UC1.1.3			+									
UC1.1.4				+								
UC1.1.5					+							
UC1.1.6						+						
UC1.1.7							+					
UC1.1.8								+				
UC1.1.9									+			
UC1.1.10										+		
UC1.1.11											+	
UC1.1.12												+

Fig. 1.5. Requirements tracing matrix

1.5.6 Development of non-functional requirements

To determine the quality attribute of the web application, the following non-functional requirements were identified for the user who will deploy this application locally:

– The software should work on the Windows operating system and requires Microsoft Visual Studio version 2016 or higher;

– Supported server module operating systems are Linux, Windows, MacOS;

– Support for all modern browsers for the client-side software;

– An internet connection is required.

1.6 Setting the set of module tasks

The purpose of the development is to create an e-commerce web application using web scraping with an optimized product reading process, implement a base for an order processing conveyor framework, and dynamic design, and handle financial transactions in the administrative panel.

The goal of the development is to create a personalized e-commerce web application with increased efficiency through the automation of product creation processes using network data retrieval and the creation of a base for a working order processing conveyor aimed at automating order acceptance and processing by the administrator. The goal also includes creating a dynamic user interface to improve user interaction with the web application and its administrative and user functionality.

To achieve the set goal, the following tasks need to be addressed:

– Implement authentication logic;

– Introduce authentication logic;

– Develop functionality for viewing and editing products;

– Develop functionality for generating products and their descriptions taken from internet stores;

– Implement a base for a framework and template for the order processing conveyor;
– Develop functionality for viewing and editing products in stock;
– Develop logic for creating new users based on roles;
– Implement logic for changing the application’s background;
– Add functionality for viewing products;
– Implement the ability to add products to the shopping cart;
– Develop logic for canceling the shopping cart;
– Introduce authentication logic throughout the entire session;
– Implement functions for handling financial transactions.

Conclusions on the section

Conclusions for this section include that after analyzing the requirements for the software, the general principles, purpose, and scope of the software have been detailed. In addition, existing modern solutions were analyzed in the “Analysis of successful IT projects” section, highlighting their advantages and disadvantages. Requirements for the software were analyzed, covering the software itself, data storage, functional, and non-functional requirements. A requirements traceability matrix was also created.

2 SOFTWARE MODELING AND DESIGN

To create an e-commerce web application using web scraping with an optimized product reading process, the software development process was employed, known as the Software Development Life Cycle (SDLC). SDLC is an iterative logical process aimed at creating computer software to achieve a specific goal or task. It establishes an international standard used for the development and improvement of software products. It provides a structured set of actions to be followed during the development, creation, and maintenance of high-quality software [12]. The diagram of the SDLC life cycle is depicted in Fig. 2.1.

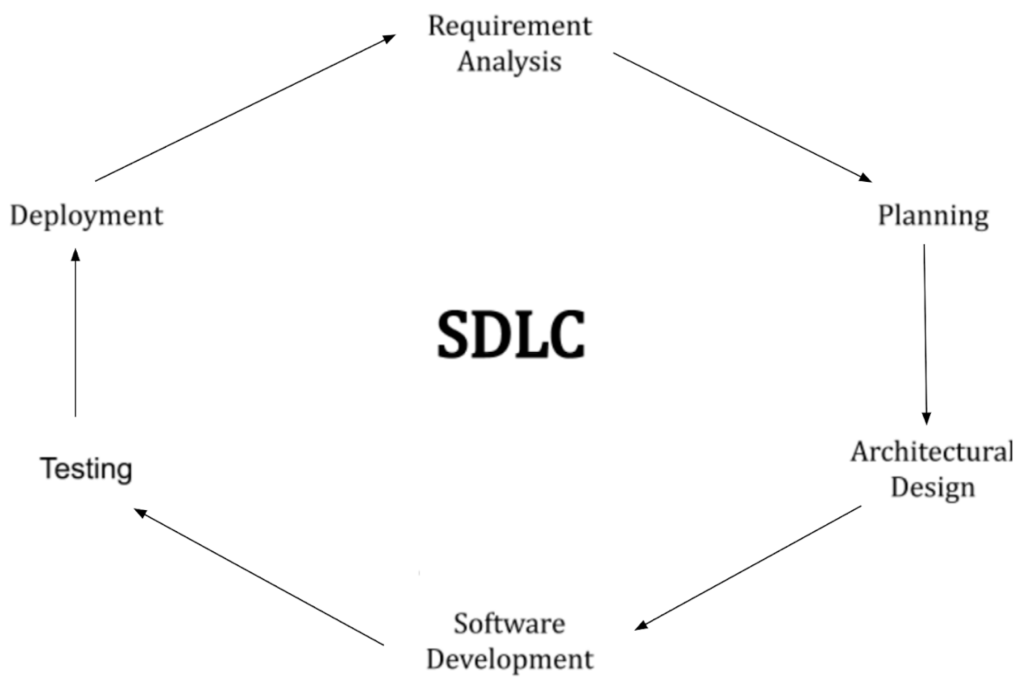


Fig. 2.1. Diagram of the SDLC life cycle

The main objective of the software development process is to create an efficient product. This project adheres to this process to achieve a successful outcome [13].

2.1 Software modeling and analysis

For users of the software, the following common processes are essential: viewing the main product page, adding products to the shopping cart, processing and paying for orders through the “Stripe” payment service, and the ability to change the application’s background.

The stage of viewing the main product page involves actions by the user, who launches the web application, lands on the product page, and can view product details.

The stage of adding a product to the shopping cart allows the user to select a product and add it to the cart, review the cart's contents, and edit the quantity of items.

The stage of order processing and payment includes entering the user's personal information, making payment for the order through the "Stripe" service, and completing the process.

Finally, the stage of changing the application's background enables users to change the background on the main page of the application using a dedicated button in the toolbar.

At this stage, the user can change the application's background. Additionally, the general processes that the software administrator goes through include processes of authorization and authentication, access to the administration panel, viewing and editing products, processing orders, viewing and editing the quantity of products in stock, creating new users, and all the processes available to users.

Sequential description of the authorization and authentication stage:

– On the first visit to the website, the administrator clicks on the "Log In" button, which redirects them to a page for entering login credentials;

– Next, the administrator enters their login and password;

– After correctly filling in the required fields and clicking the "Log In" button, the administrator returns to the main application page with an updated part of the interface indicating access to the administration panel;

– In case of an urgent exit from the web application and re-entry, the administrator can access it without losing data due to the authentication process.

At this stage, the administrator can authorize, thereby opening a new part of the interface, the administration panel. Also, in case of exiting the web application and re-entering, the administrator can access it without losing their role, thanks to the authentication process.

Sequential description of accessing the administration panel:

– After authorization, on the main page in the admin tools panel, the administrator has a link to the administration panel;

– By clicking on this link, the administrator is directed to a new page with the administration panel;

– The administration panel includes tools for managing both the platform itself and its users.

At this stage, the administrator can access the administration panel and have access to two parts of the panel: platform administration and user administration.

Sequential description of viewing and editing products:

– After entering the administration panel, in the administration menu, the administrator is presented with a sub-panel for viewing and editing products;

– By clicking on this sub-panel, the administrator dynamically switches to the product editing panel;

– This panel includes an interface with functions for creating, viewing, editing, and deleting products.

At this stage, the administrator can access the menu for viewing and editing products and use its functions to manage products.

Sequential description of generating a text file and describing products from internet stores:

– In the administration panel, the administrator selects a sub-panel for generating text files of products;

– After clicking on this sub-panel, the administrator dynamically switches to the product text file generation panel and can choose products from three Ukrainian internet stores;

– After clicking the “Generate” button, the administrator receives a generated text file with the listed products on their local system.

At this stage, the administrator can access the menu for generating text files of products and use its functions to obtain product descriptions from internet stores.

Sequential description of order processing:

– In the platform menu, the administrator selects a sub-panel for order processing;

– The sub-panel includes three order processing stages: pending, packaging, and dispatch;

– The administrator can manually switch orders between these processing stages until the order is completed.

At this stage, the administrator moves to the platform’s order processing sub-panel, which includes a three-stage order processing system.

2.2 Rationale for development tools

2.2.1 Software development environments

Since the e-commerce web application using web scraping with an optimized product reading process is developed in the C# programming language and ASP.NET Core framework, the decision was made to use the integrated development environment Visual Studio by Microsoft with the necessary plugins suitable for web application development. Additionally, a range of technologies and processes listed below in the sections were utilized.

2.2.2 Programming language C#

To develop an e-commerce web application, the decision was made to use the C# programming language as the primary language in this implementation.

The C# is a modern, object-oriented language that allows for the creation of secure and reliable programs, working in conjunction with .NET [14].

The choice of language is justified by the fact that C# is one of the most versatile programming languages in the world.

2.2.3 ASP.NET Core framework

The software implemented on the ASP.NET Core platform is a high-performance platform for creating modern applications with cloud support and open-source code.

Among the advantages of this platform, the following aspects can be highlighted:

– Support for multiple platforms, allowing ASP.NET Core applications to work on various operating systems, eliminating the need to create separate programs for each platform;

– Built-in "IoC" container for automatic dependency management;

– Integration with modern user interface frameworks such as AngularJS, ReactJS, Vue.js, Umber, Bootstrap, and Bulma [15]. The diagram of the ASP.NET division is depicted in Fig. 2.2.

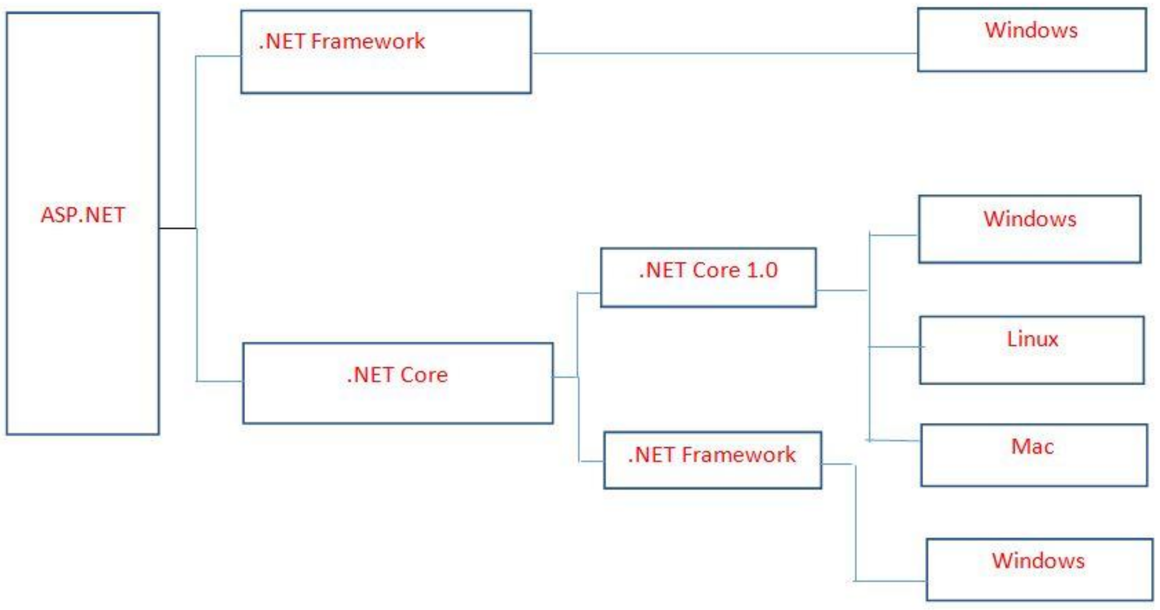


Fig. 2.2. Diagram of the ASP.NET division

As a result of utilizing these capabilities of the ASP.NET Core platform, the e-commerce web application was developed, characterized by high performance, lower system memory requirements, easy deployment, and convenient maintenance.

2.2.4 MVC architecture pattern

The e-commerce web application using web scraping with an optimized product reading process built using an MVC pattern due to its high effectiveness.

Model-View-Controller, or MVC, is an architectural pattern that divides a program into three main components: model, view, and controller. Each of these components is designed to handle specific aspects of software development. MVC is often used to create scalable and expandable projects [16]. The interaction of the components of the MVC design pattern is depicted in Fig. 2.3.

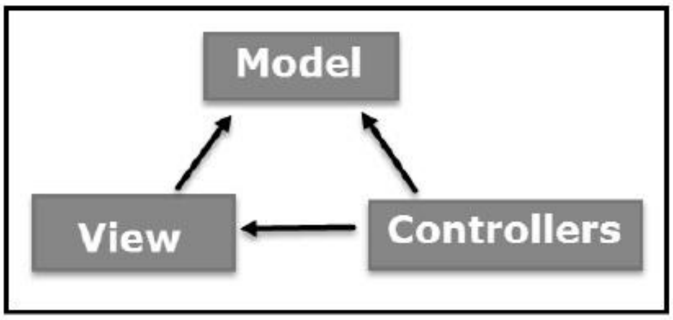


Fig. 2.3. Diagram of the interaction of MVC template components

The three components of the MVC software design pattern can be described as follows:

– The model manages data and business logic and does not directly interact with the user;

– The view describes the program’s external appearance;

– The controller acts as an intermediary between the model and the view. It receives data from the user, passes it to the model, receives the processed result, and passes it to the view.

2.2.5 AWS cloud platform

Amazon Web Services, known as AWS, is a comprehensive remote computing service that provides a variety of online services through cloud storage. AWS encompasses more than 200 products and services. Accessing Amazon Web Services is done through HTTP, using the REST architectural style and SOAP protocol, typically for older systems, and JSON for newer ones [17].

2.2.6 Docker platform

For the e-commerce web application, Docker technology was implemented to facilitate the deployment of the software. Docker is an open-source platform for containerization.

The Docker platform utilizes containers, which are implemented through process isolation and virtualization built into the Linux kernel. These capabilities allow multiple components of an application to share resources of a single host operating system instance, much like a hypervisor enables multiple virtual machines to share the CPU, memory, and other resources of a single physical server [18]. The docker system architecture scheme is depicted in Fig. 2.4.

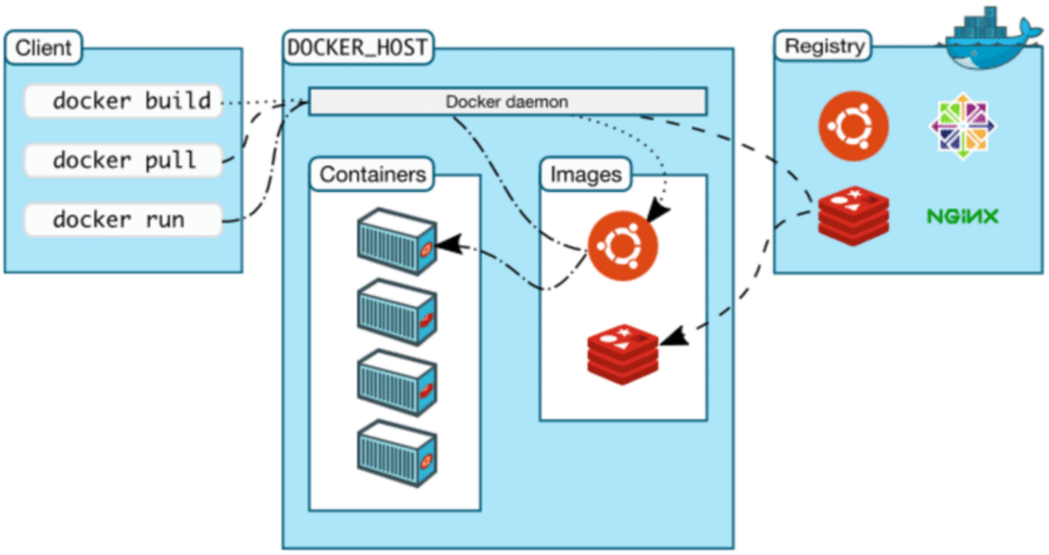


Fig. 2.4. Docker system architecture scheme

As a result, container technology offers all the functionality and benefits of virtual machines, including application isolation and cost-effective scalability.

2.2.7 Stripe payment service

To facilitate convenient and efficient management of financial transactions in this software, the Stripe service has been implemented. Stripe is a payment service provider that business owners can use to accept various forms of payments, including credit cards, direct purchases, and deferred payments. It’s important to note that Stripe charges a commission for each transaction.

As a payment processor, Stripe allows business owners to accept and process payments from credit and debit cards. Additionally, by using Stripe, companies can accept payments via mobile wallets and make immediate purchases, subsequently paying for the goods or services. Stripe also supports payments in various currencies through its built-in service called “Stripe Payments”, which processes payment data.

The operation of this software involves the following steps:

– The client provides information about their card;

– Card data is encrypted by the Stripe payment gateway;

– Stripe forwards this data to the acquiring bank for transaction processing on behalf of the seller, with Stripe acting as the seller. This means that Stripe users do not need to create a seller account;

– Payment data is sent to the card-issuing bank (the bank that issued the card) of the cardholder through the credit card network, such as Visa or Mastercard;

– The card-issuing bank approves or declines the transaction;

– The transaction result signal is sent back to the client through Stripe;

– After the card-issuing bank of the cardholder completes its approval, the user can transfer funds from Stripe to their bank account, and Stripe users can receive payouts after transaction processing.

The stripe payment gateway system design is depicted in Fig. 2.4.

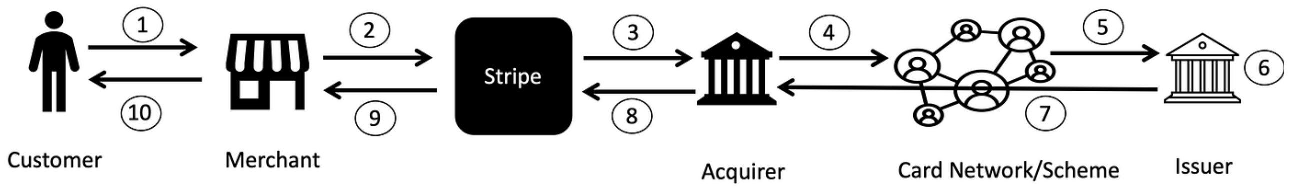


Fig. 2.5. Stripe payment gateway system design

Users will pay Stripe a certain fee for processing each transaction. The amount of these fees depends directly on the type of transaction; for example, each payment may cost 2.9% of the item’s price [19].

2.2.8 Bulma framework

To create the user interface for the project, we implemented the Bulma framework. Bulma is a free CSS solution based on the Flexbox layout. Thanks to Bulma, we have access to a wide range of built-in features, greatly simplifying the CSS coding process [20].

2.2.9 Vue.js framework

Also, to create the user interface, we implemented the Vue.js framework. It's a user interface framework written in JavaScript. It's built on HTML, CSS, and JavaScript and provides a declarative programming model that helps streamline user interface development. Vue.js uses a virtual DOM. Instead of making changes directly to the DOM, a copy of it is created, represented as JavaScript data structures. As a result, the final changes are updated to the real DOM [21]. This is a significant optimization advantage.

2.2.10 Microsoft SQL server database management system

To manage the software, SQL Server is used – it is a relational database management system, developed by Microsoft. It is based on SQL, which is a standard programming language for interacting with relational databases. SQL Server is associated with Transact-SQL or T-SQL from Microsoft, which includes a set of special programming constructs [22].

2.2.11 HTML Agility Pack library

To implement the logic of fetching products from internet stores, the HTML Agility Pack library was used – it is a syntactic HTML parser that creates a DOM structure for reading and writing and supports common XPATH or XSLT. This is a .NET library that allows analyzing HTML files "offline". The syntactic parser is very tolerant of HTML [23].

2.3 Database Architecture

For the e-commerce web application using web scraping with an optimized product reading process, domain models were built, an ER diagram was developed, and a database schema was implemented.

Generally, the better the database architecture is developed, the faster the program can retrieve and process the necessary amount of data.

2.3.1 Building a domain model

The domain model is a type of metadata that consists of multiple tables [24].

To gain a clear understanding of the domain of the e-commerce web application using web scraping with an optimized product reading process, an entity table (and inter-project entities) for the domain was created. You can view it below as Table 2.1.

Table 2.1.

All domain entities in the database

Essence	Attribute
Customer Information	First name, last name, mail, phone number, required address, optional address, city, zip code, order date
Order	Order code, order link, stripe service token link, first name, last name, email, phone number, required address, optional address, city, zip code, order date, order status
Composition of orders	Warehouse code, order code, quantity of goods
Product or Product	Product code, naming, specification, price, warehouses
Product in cart	Product code, product name, warehouse code, price, quantity
Warehouse	Warehouse code, description, quantity, product code, order warehouse
Warehouse on hold	Pending warehouse code, session token code, warehouse code, quantity, expiration date

2.3.2 ER diagram

When developing the database for the e-commerce web application, an entity-relationship diagram representing the relationships between sets of entities stored in the database was implemented. An entity is an object, a data component, and a set of entities is a collection of similar entities [25].

In Fig. 2.6., you can see the ER diagram of the subject area.

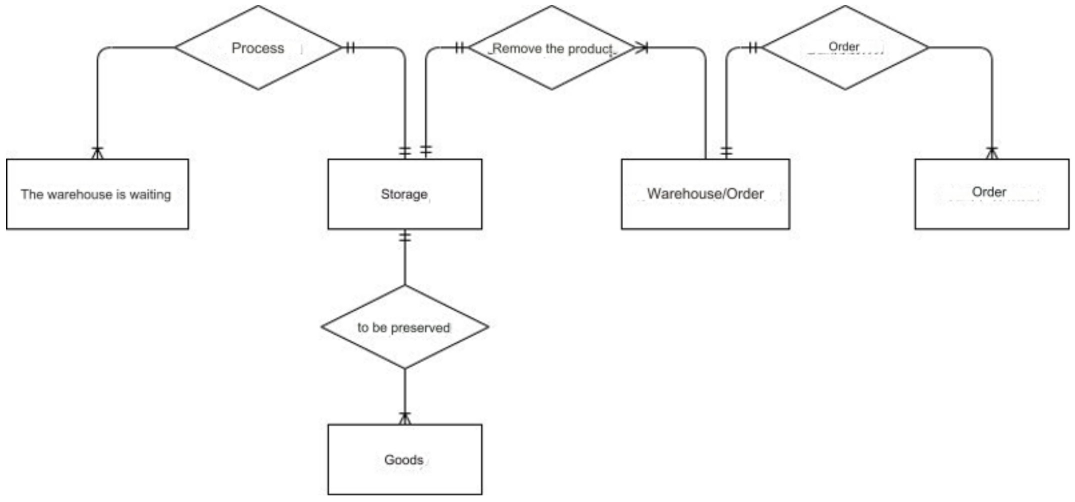


Fig. 2.6. ER diagram of the subject area

2.3.3 Database schema

To create an architecturally correct database for the e-commerce web application, a database schema was designed.

For this software, a simplified database schema or data logical model was created. In Fig. 2.7., you can see the simplified database schema of the domain.

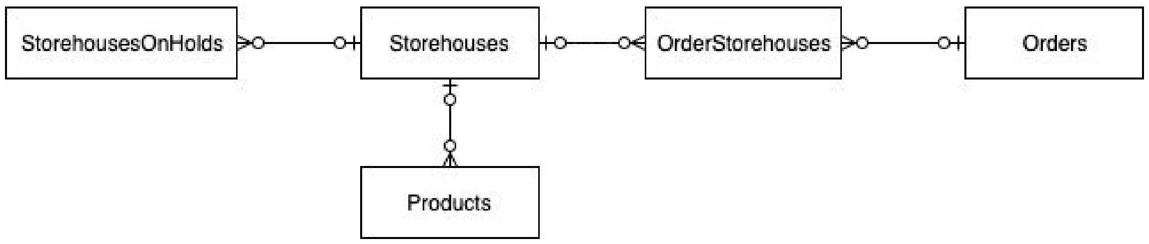


Fig. 2.7. Simplified database schema

Additionally, data logical model was generated for this web application. It is presented in Fig. 2.8. and Fig. 2.9. under the title “Database schema”.

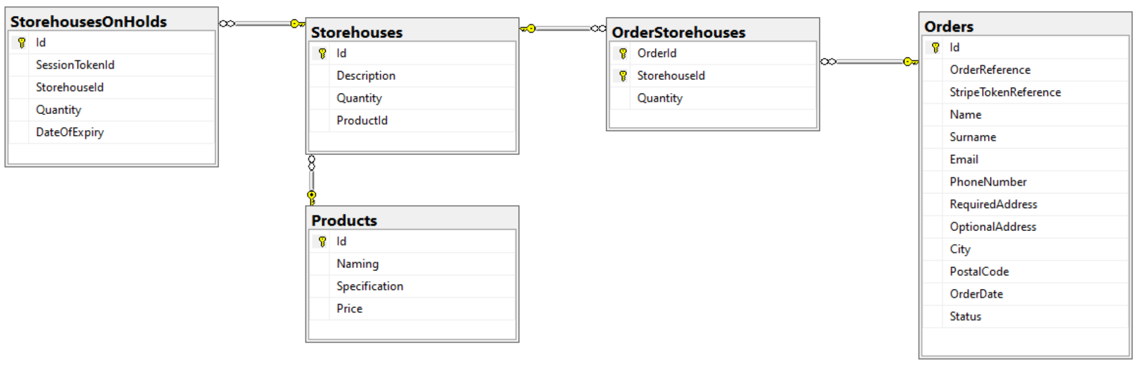


Fig. 2.8. Database schema – first part

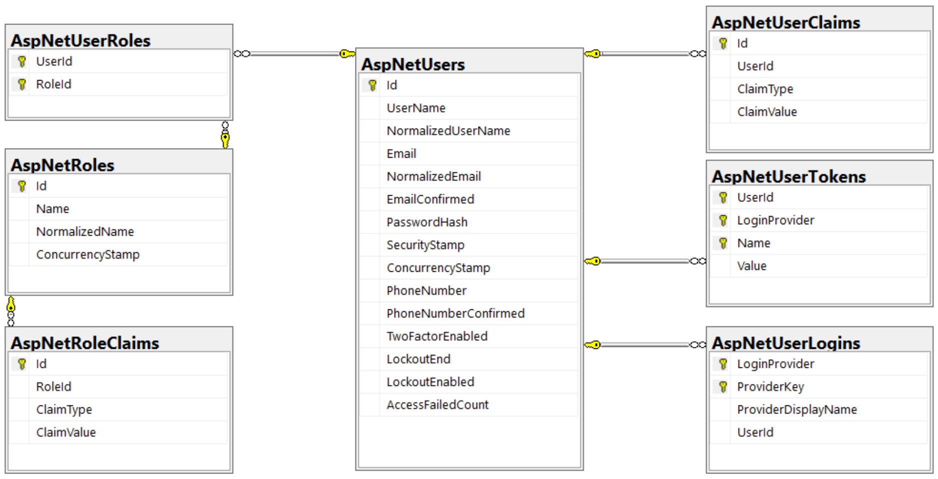


Fig. 2.9. Database schema – second part

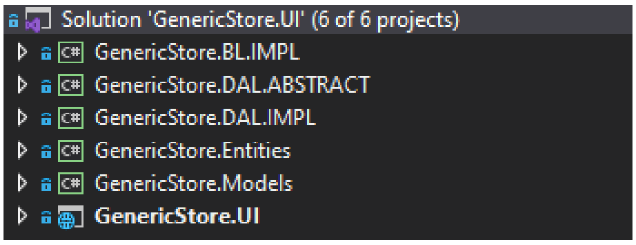
As a result, a database schema was created, which represents the structure of the database system, providing a logical representation of the entire database and supported by the database management system. It defines how data is organized, and their relationships, and formulates all the constraints that need to be applied to the data [26].

2.4 Software architecture

This software has been divided into six projects, which use a three-layer architecture, and two projects for implementing testing layers, which will be discussed in the fourth section. Thus, DAL, BL, and PL layers have been implemented. The flexibility of this approach adheres to SOLID principles.

Specifically, the DAL and BL layers are divided into class libraries, each of which is intended for the interfaces of the respective logic layer. These interfaces establish a connection between the DAL and BL layers.

You can see the architecture of the e-commerce web application using web scraping with an optimized product reading process in Fig. 2.10.



```
Solution 'GenericStore.UI' (6 of 6 projects)
  GenericStore.BL.IMPL
  GenericStore.DAL.ABSTRACT
  GenericStore.DAL.IMPL
  GenericStore.Entities
  GenericStore.Models
  GenericStore.UI
```

Fig. 2.10. E-commerce web application architecture using web scraping with an optimized product reading process

Moving on to the consideration of each project, the first in logic is GenericStore.Entities – this is a class library that includes all database entities, intermediate entities, and everything related to them.

GenericStore.Models is a class library that embodies data transfer objects.

GenericStore.DAL.IMPL is a class library responsible for the logic of the data access layer.

GenericStore.DAL.ABSTRACT is a class library responsible for the logic of the interfaces of the data access layer.

GenericStore.BL.IMPL is a project that is responsible for logic and implementing services.

GenericStore.UI is a project that is responsible for the graphical user interface of the application and implements the MVC architecture, with models stored as entity data and used to process parts of the code. Moreover, the views generated in this software can be in the form of HTML code or JSON response format. Controllers are used to combine view and model components and are applied for data processing.

2.4.1 UML software diagrams

During the development of the architecture of this software, it was essential to use UML, which stands for Unified Modeling Language. In other words, UML is a modern approach to modeling and documenting software. It is a popular method for modeling business processes and is based on schematic representations of software components. By using UML, it becomes easier to understand potential flaws or errors in the software or business processes.

Two main categories encompass all other types of UML diagrams: structural and behavioral. Based on their names, some UML diagrams analyze and depict the structure of an application or process, while others describe the behavior of the application, its actors, and components [27].

As a result, a series of UML diagrams were created and integrated into this e-commerce web application.

2.4.1.1 Use case diagram

UML uses case diagrams to model the behavior and requirements of an application. They describe high-level functions and the scope of the web application. These diagrams also define interactions between the application and its actors. Use cases and actors describe what the application does and how participants use it, but not how the application works internally. In other words, a use case diagram describes the “action” rather than the “process of creating action logic” [28].

Typically, developing use case diagrams is part of the project modeling stages. The main goals of creating use case diagrams during the process include:

– Specifying the application’s context;

– Capturing the application’s requirements;

– Verifying the application’s architecture.

A diagram of the use cases of the subject area is shown in the in Fig. 2.11. under the title “Structural scheme of use cases”.

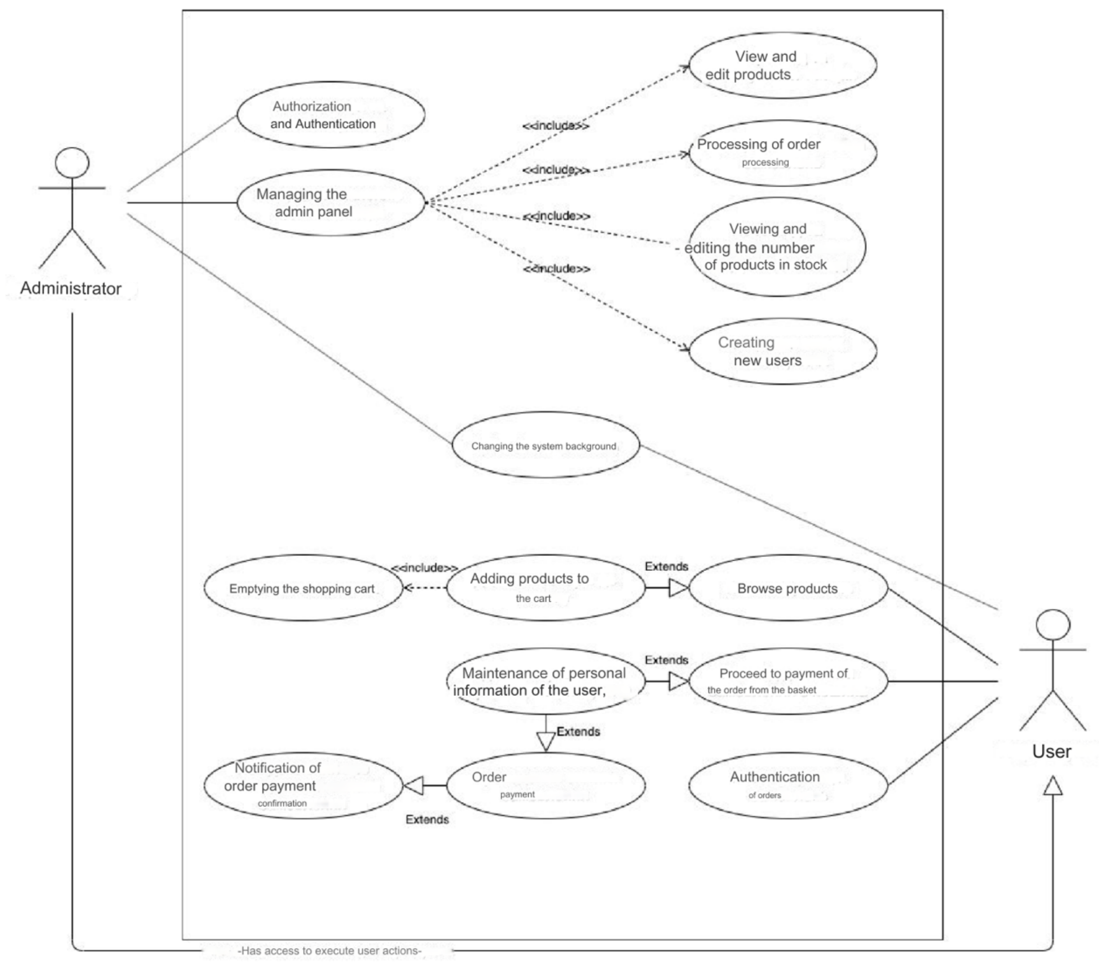

Administrator
User
Authorization and Authentication
Managing the admin panel
View and edit products
Processing of order processing
Viewing and editing the number of products in stock
Creating new users
<<include>>
Changing the system background
Emptying the shopping cart
Adding products to the cart
Browse products
Extends
Maintenance of personal information of the user,
Proceed to payment of the order from the basket
Notification of order payment confirmation
Order payment
Authentication of orders
-Has access to execute user actions-

Fig. 2.11. Structural scheme of use cases

As a result, this diagram can be referenced throughout the entire development process.

2.4.1.2 Class diagram

The e-commerce web application using web scraping with an optimized product reading process cannot be done without a class diagram, just like any other properly formed architectural solution.

The class diagram is used to illustrate and create a functional representation of classes in the software application. It serves as a vital resource throughout the software development lifecycle [29].

The diagram of the classes of the subject area is shown in the appendix B, in Fig. B.1., under the title “Structural scheme of software classes”.

Therefore, class diagrams are one of the most important types of UML diagrams and are crucial in software development.

2.4.1.3 Business Process Model and Notation diagram

To model the primary client-side business processes of the web application, a model diagram, and Business Process Model and Notation (BPMN) notation were introduced. This diagram is depicted in Fig. 2.12.

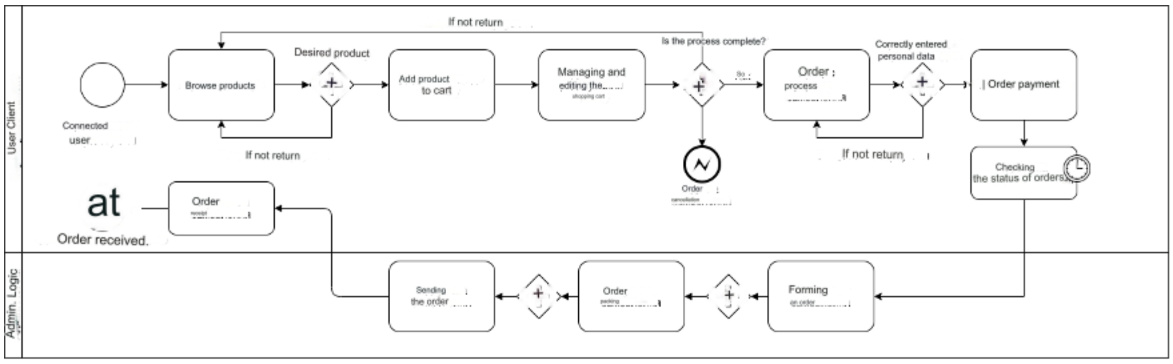


Fig. 2.12. Diagram of the model and notation of the business processes of the application

2.4.1.4 Sequence diagram

Sequence diagrams are interaction diagrams that provide a detailed description of how operations are executed. They capture interactions between objects [30].

A diagram of the application authorization sequence is shown in Fig. 2.13.

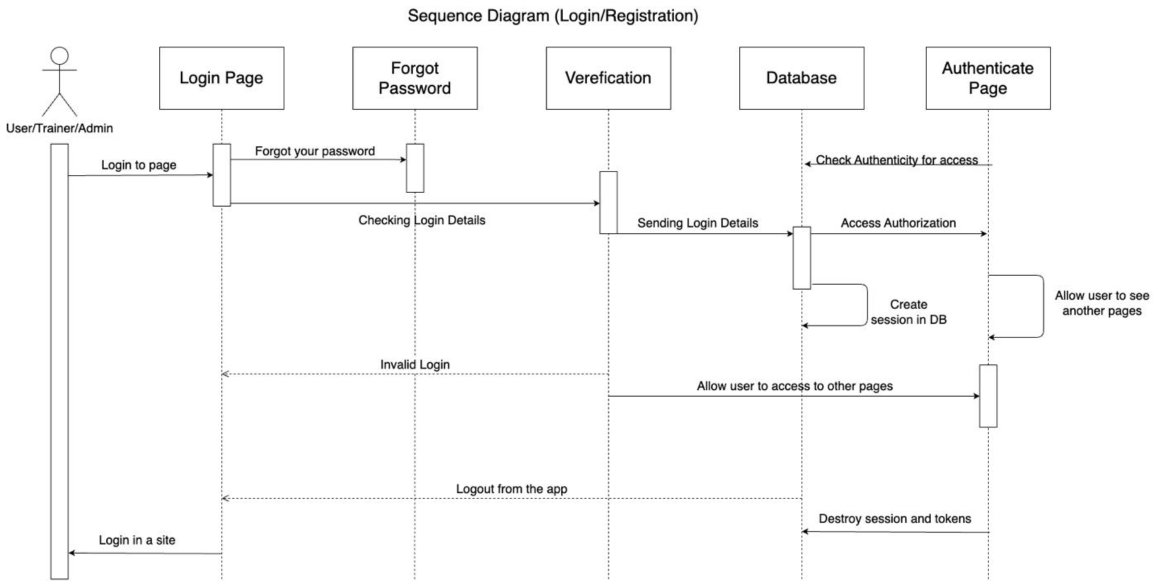


Fig. 2.13. Application authorization sequence diagram

The purpose of a sequence diagram is to:

– Model high-level interactions between active objects;

– Model interactions between object instances that implement a use case.

Fig. 2.14. shows a diagram of the sequences of the entity update process in the application.

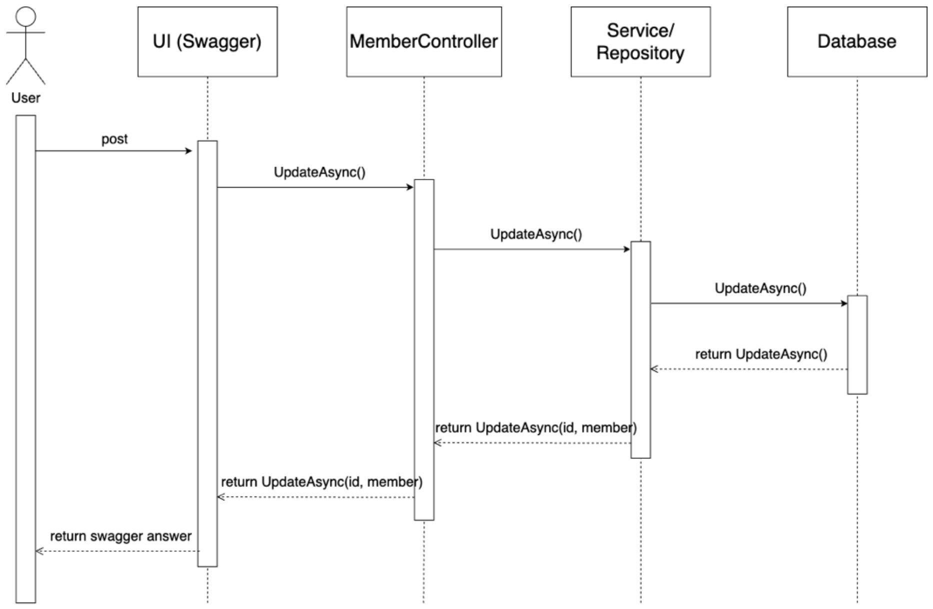


Fig. 2.14. Entity-update process sequence diagram

In Fig. 2.15, you can see a diagram of the sequence of the general logic of the user process.

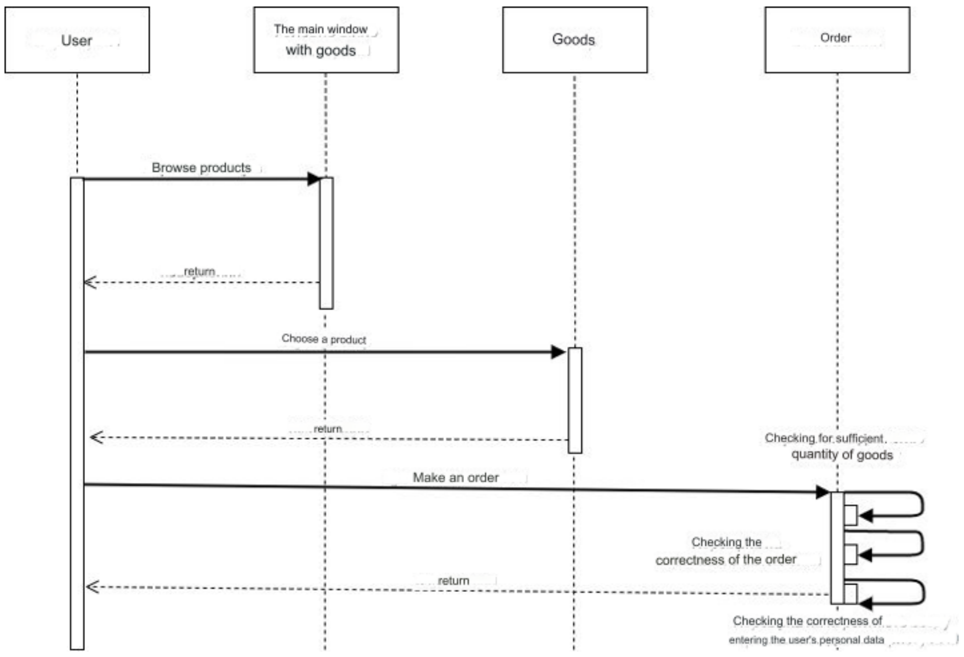

User
The main window with goods
Goods
Order
Browse products
return
Choose a product
return
Checking for sufficient quantity of goods
Make an order
Checking the correctness of the order
return
Checking the correctness of entering the user's personal data

Fig. 2.15. Diagram of the sequence of the general logic of the user process

2.4.1.5 State diagram

A state diagram is a tool for describing the behavior of a program, taking into account all possible states of an object. This behavior is represented and analyzed through a series of events that occur in one or more possible states. Each diagram represents objects and tracks their different states throughout the entire application [31].

It is worth noting separately the status diagram for the process of adding goods and paying for the order, shown in Fig. 2.16.

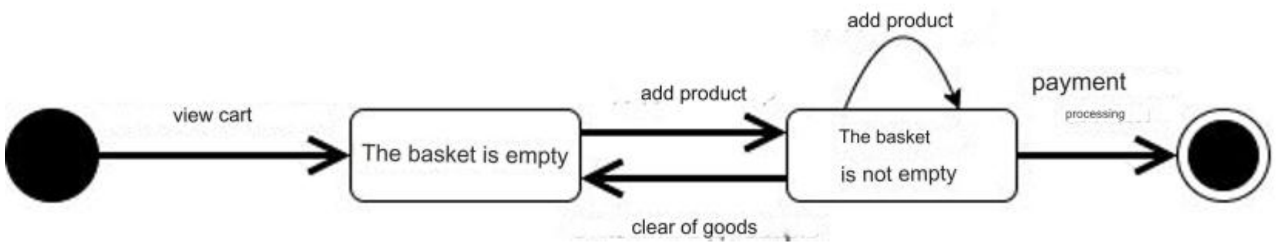
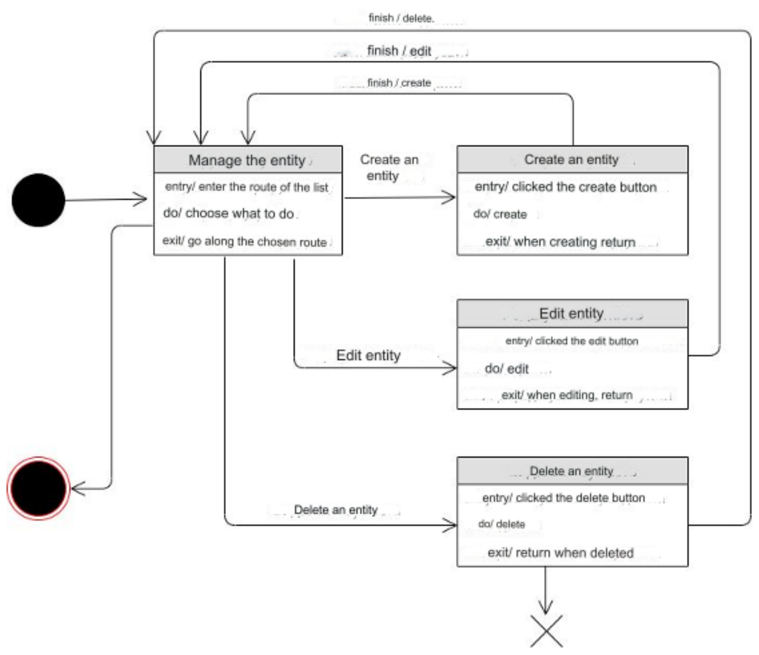


Fig. 2.16. Status diagram for the order payment process

It is also worth noting the state diagram for the CRUD process of entities, shown in Fig. 2.17.



finish / delete.
finish / edit
finish / create
Manage the entity
entry/ enter the route of the list
do/ choose what to do
exit/ go along the chosen route
Create an entity
Create an entity
entry/ clicked the create button
do/ create
exit/ when creating return
Edit entity
Edit entity
entry/ clicked the edit button
do/ edit
exit/ when editing, return
Delete an entity
Delete an entity
entry/ clicked the delete button
do/ delete
exit/ return when deleted

Fig. 2.17. State diagram for entity crud process

2.4.1.6 Component diagram

The component diagram breaks down the developed web application into various levels of functionality. Each component is responsible for a specific purpose within the entire application and interacts with other elements [32].

The component diagram for the MVC process of the main entities of the e-commerce web application is shown in in Fig. 2.18. under the title “Structural diagram of software components”.

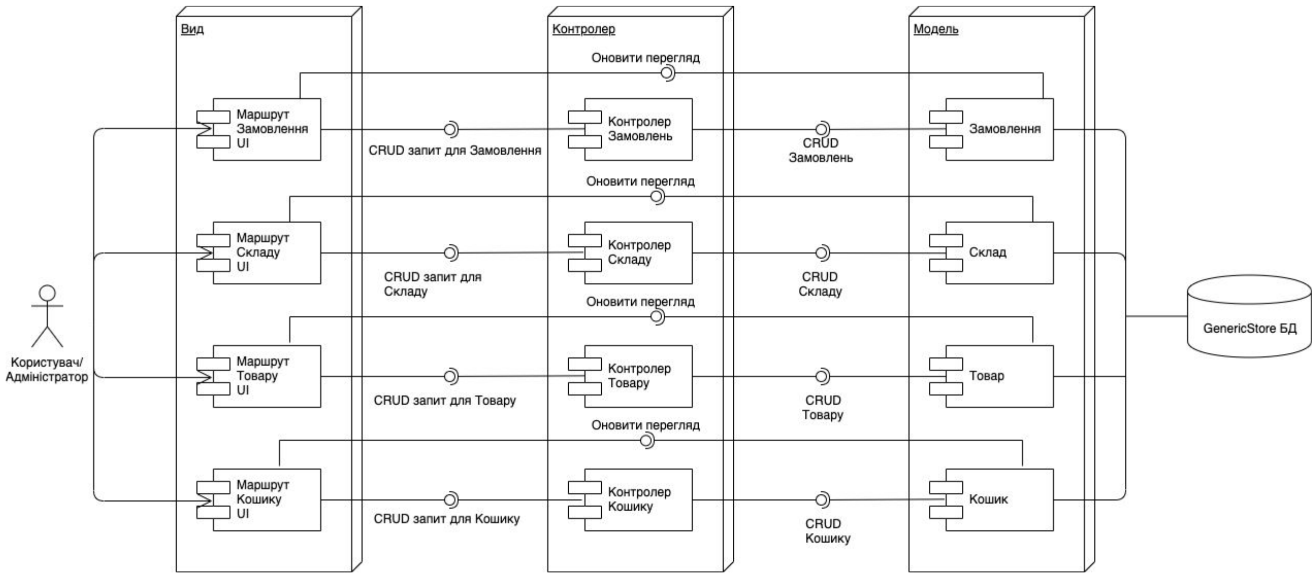


Fig. 2.18. Structural diagram of software components

2.4.2 Software architecture selection analysis

This section analyzes the choice of architecture for implementing the e-commerce web application. There are two fundamental software architectures:

– Monolithic architecture – the use of a single general module;

– Microservices architecture – a set of independent, separately deployable services.

After analyzing the monolithic architecture, the advantages and disadvantages were reviewed and described in Table 2.2.

Table 2.2 – Advantages and disadvantages of monolithic architecture

Advantages	Disadvantages
Convenience in writing	As the project grows, significant portions of code may

	need editing, even with minor changes
Fast interaction between project parts	Complexity in dissecting and comprehending long-standing projects, like a 20-year-old monolith
Interconnectedness	Learning the entire system takes a lot of time

After analyzing the microservice architecture, we reviewed and described it in Table 2.3.

Table 2.3 – Advantages and disadvantages of microservice architecture

Advantages	Disadvantages
Reliability and security	Complex test writing
Scalability and distribution	Costly maintenance – each microservice requires its own server
Minimal business logic per microservice	Network overhead and round-the-clock code churn

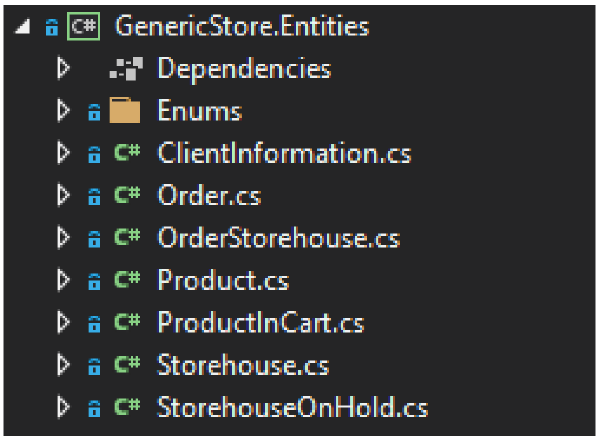
Therefore, after exploring various architectural options with all their advantages and disadvantages, a decision was made to implement a monolithic architecture for this e-commerce web application. The choice was justified by the single-module logic of the architecture and its ease of understanding and implementation, as opposed to microservices architecture.

2.4.3 Description of software architecture

Based on the architecture of the web application in Fig. 2.12. and the diagram of classes shown in the appendix B, in Fig. B.1. under the title “Structural scheme of software classes”. It is worth considering and describing the architecture of the e-commerce web application for the sale of products under the project name “GenericStore”.

Referring to a three-layer architecture, the software includes GenericStore.DAL.IMPL, GenericStore.DAL.ABSTRACT, GenericStore.BL.IMPL, and GenericStore.UI. Each of the layers and its inclusions is responsible for a separate part of the functionality. In the future, each layer of the logic of this software will be considered.

Let’s take a look at the GenericStore.Entities project, which implements entities and their fields used to store data in the database. The content of the entities is shown in Fig. 2.19.



```
GenericStore.Entities
  Dependencies
  Enums
  ClientInformation.cs
  Order.cs
  OrderStorehouse.cs
  Product.cs
  ProductInCart.cs
  Storehouse.cs
  StorehouseOnHold.cs
```

Fig. 2.19. GenericStore.Entities project, entities in the project

The details of each e-commerce web application entity for selling products are shown in Table 2.4.

Table 2.4.

Detailed description of application database entities

Class field	Description	Type
	Order	
Id	Order code	Integer
OrderReference	Order link	String
StripeTokenReference	Stripe token link	String
Name	Customer name	String
Surname	Customer’s last name	String
Email	Customer mail	String
PhoneNumber	Customer phone number	String
RequiredAddress	Customer address required	String
OptionalAddress	Optional customer address	String

Table 2.4. (continue)

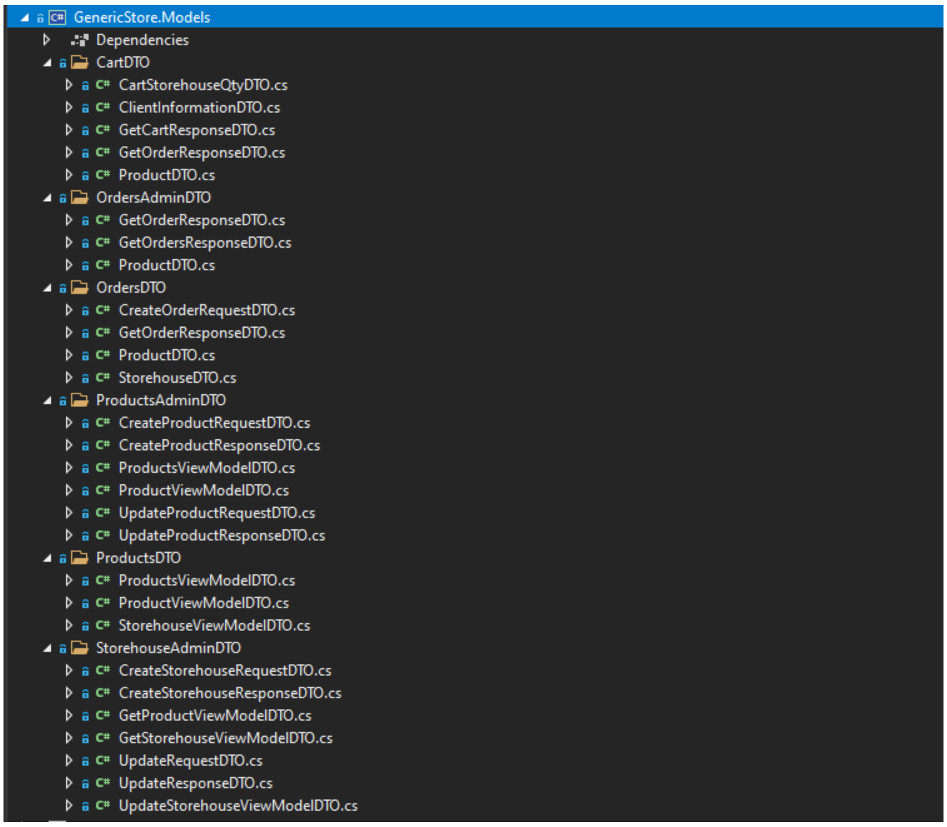
City	Customer city	String
PostalCode	Postal code of the customer	String
OrderDate	The date of the customer’s order	DateTime
Status	Customer order status	OrderStatus
OrderStorehouses	Link to warehouse	ICollection <OrderStorehouse>
OrderStorehouse		
OrderId	Warehouse code	Integer, Order
StorehouseId	Order code	Integer, Storehouse
Quantity	Quantity of goods	Integer
Product		
Id	Product code	Integer
Naming	Name of the product	String
Specification	Product specification	String
Price	The price of the product	Decimal
Storehouses	Warehouses	ICollection <Storehouse>
Storehouse		
Id	Warehouse code	Integer
Description	Product description in stock	String

Table 2.4. (continue)

Quantity	Quantity of goods in stock	Integer
ProductId	Product code	Integer, Product
OrderStorehouses	Warehouse/Order	ICollection <OrderStorehouse>
StorehouseOnHold		
Id	Warehouse code pending	Integer
SessionTokenId	Warehouse session token code	String

StorehouseId	Warehouse code	Integer, Storehouse
Quantity	Quantity of goods in stock	Integer
DateOfExpiry	Token expiration date	DateTime

Next, we will examine the GenericStore.Models project, which includes entities implemented with the DTO (Data Transfer Object) pattern, used for data transfer between the e-commerce web application’s logic. The GenericStore.Models project is depicted in Fig. 2.20.



```
GenericStore.Models
  Dependencies
  CartDTO
    CartStorehouseQtyDTO.cs
    ClientInformationDTO.cs
    GetCartResponseDTO.cs
    GetOrderResponseDTO.cs
    ProductDTO.cs
  OrdersAdminDTO
    GetOrderResponseDTO.cs
    GetOrdersResponseDTO.cs
    ProductDTO.cs
  OrdersDTO
    CreateOrderRequestDTO.cs
    GetOrderResponseDTO.cs
    ProductDTO.cs
    StorehouseDTO.cs
  ProductsAdminDTO
    CreateProductRequestDTO.cs
    CreateProductResponseDTO.cs
    ProductsViewModelDTO.cs
    ProductViewModelDTO.cs
    UpdateProductRequestDTO.cs
    UpdateProductResponseDTO.cs
  ProductsDTO
    ProductsViewModelDTO.cs
    ProductViewModelDTO.cs
    StorehouseViewModelDTO.cs
  StorehouseAdminDTO
    CreateStorehouseRequestDTO.cs
    CreateStorehouseResponseDTO.cs
    GetProductViewModelDTO.cs
    GetStorehouseViewModelDTO.cs
    UpdateRequestDTO.cs
    UpdateResponseDTO.cs
    UpdateStorehouseViewModelDTO.cs
```

Fig. 2.20. GenericStore.Models project, DTO entities

The description of DTO entities in the GenericStore.Models project is discussed and presented in Table 2.5. These DTO entities are essential for interaction between the internal logic, the database, and the user interface. These entities are organized into various directories, each implementing different aspects of the creation logic.

Table 2.5.

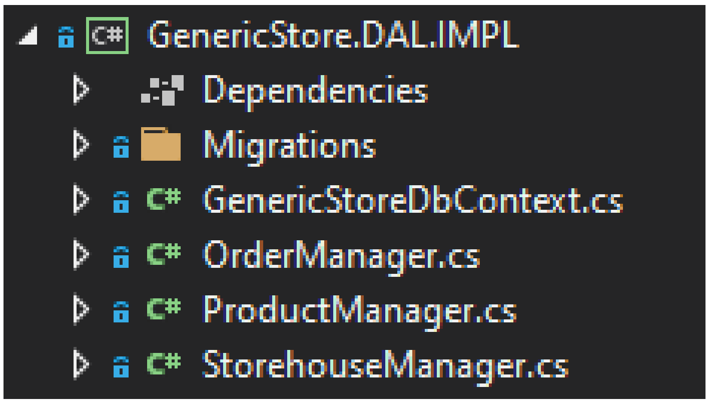
Detailed description of DTO entities for data transfer

DTO entity directory	Description

CartDTO	Recycle bin entity in DTO to execute the assigned functionality of the application recycle bin
OrdersAdminDTO	DTO entities for performing order-related functionality for administrators
OrdersDTO	DTO entities for executing order-related functionality from the user's perspective
ProductsAdminDTO	DTO entities for product-related functionality from the administrator's perspective
ProductsDTO	A collection of DTO entities for executing product-related functionality from the user's perspective
StorehouseAdminDTO	A collection of DTO entities for performing warehouse-related functionality with administrator privileges

Next, the first layer to consider, which is directly responsible for data access, includes projects named GenericStore.DAL.IMPL and GenericStore.DAL.ABSTRACT. They provide simplified access to data stored in the database.

The class library GenericStore.DAL.IMPL includes classes for initializing database entities and managing them using the "Manager" template. It is in this project that the database is initialized along with the migrations folder. The architecture and content of the GenericStore.DAL.IMPL class library are depicted in Fig. 2.21.



```
GenericStore.DAL.IMPL
    Dependencies
    Migrations
    GenericStoreDbContext.cs
    OrderManager.cs
    ProductManager.cs
    StorehouseManager.cs
```

Fig. 2.21. GenericStore.DAL.IMPL project, part of the logic

To implement this layer, the “Manager” template was used, which, in turn, is a branch of the “Repository” template. The class library GenericStore.DAL.IMPL is detailed in Table 2.6, which describes the layer itself and the classes included in it.

Table 2.6.

Detailed description of the GenericStore.DAL.IMPL project and its classes designed to store data in a database

Project or layer name	Class	Description and purpose of the class
GenericStore.DAL.IMPL	GenericStoreDbContext	This class includes all the provided entities of the e-commerce web application and initializes the connection between the program and the database
GenericStore.DAL.IMPL	OrderManager	This class manages and handles the entities of orders between the program and the database
GenericStore.DAL.IMPL	ProductManager	This class manages the entities of products between the program and the database
GenericStore.DAL.IMPL	StorehouseManager	This class handles the entities of warehouses between the program and the database

Next, we move on to the second project, GenericStore.DAL.ABSTRACT, which encompasses the implementation of data access layer interfaces. The project’s structure can be seen in Fig. 2.22.

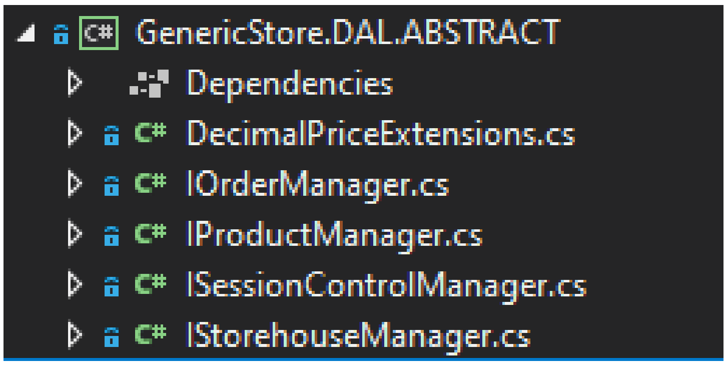

GenericStore.DAL.ABSTRACT
Dependencies
DecimalPriceExtensions.cs
IOrderManager.cs
IProductManager.cs
ISessionControlManager.cs
IStorehouseManager.cs

Fig. 2.22. GenericStore.DAL.ABSTRACT project, part of abstractions

The class library GenericStore.DAL.ABSTRACT plays a fundamental role in facilitating communication and data/method transfer among other projects and layers. It is responsible for flexible interaction between projects. A detailed description of this class library is provided in Table 2.7.

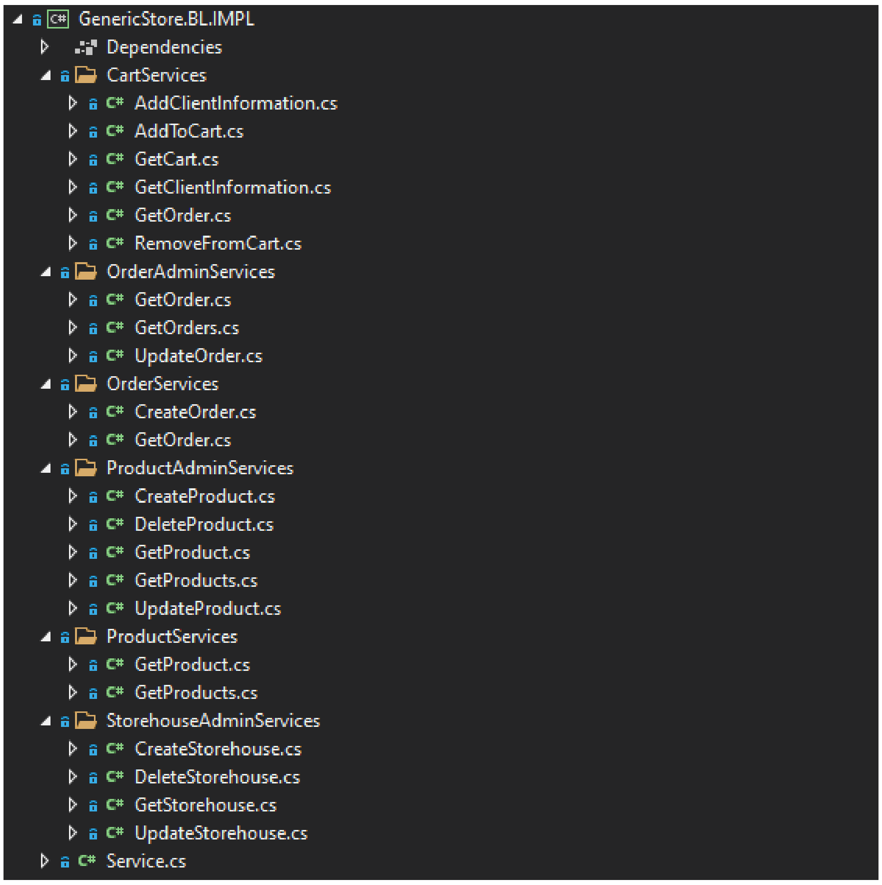
Table 2.7.

Detailed description of the GenericStore.DAL.ABSTRACT project

Project/layer name	Class	Description and purpose of the class
GenericStore.DAL .ABSTRACT	DecimalPriceExtensions	This class is a static extension that performs the function of converting and formatting the price into a string
GenericStore.DAL .ABSTRACT	IOrderManager	Responsible for abstracted shopping cart functionality and ensures the flexibility of methods
GenericStore.DAL .ABSTRACT	IProductManager	Implements abstracted product functionality
GenericStore.DAL .ABSTRACT	ISessionControlManager	Incorporates abstracted session control functionality
GenericStore.DAL .ABSTRACT	IStorehouseManager	Realizes abstracted warehouse functionality

2.5 Description of software architecture

We should consider the detailed creation of the business logic for the e-commerce web application using web scraping with an optimized product reading process. The business logic consists of the GenericStore.BL.IMPL project, which includes the core logic and application services. The structure of this project can be seen in Fig. 2.23.



```
GenericStore.BL.IMPL
  Dependencies
  CartServices
    AddClientInformation.cs
    AddToCart.cs
    GetCart.cs
    GetClientInformation.cs
    GetOrder.cs
    RemoveFromCart.cs
  OrderAdminServices
    GetOrder.cs
    GetOrders.cs
    UpdateOrder.cs
  OrderServices
    CreateOrder.cs
    GetOrder.cs
  ProductAdminServices
    CreateProduct.cs
    DeleteProduct.cs
    GetProduct.cs
    GetProducts.cs
    UpdateProduct.cs
  ProductServices
    GetProduct.cs
    GetProducts.cs
  StorehouseAdminServices
    CreateStorehouse.cs
    DeleteStorehouse.cs
    GetStorehouse.cs
    UpdateStorehouse.cs
  Service.cs
```

Fig. 2.23. GenericStore.BL.IMPL project, part of the logic

The GenericStore.BL.IMPL class library serves as the core logic of the e-commerce web application. It is responsible for implementing and utilizing the logic and functionality presented in the presentation layer. The structure of the business logic is divided into directories for convenience and flexibility in usage and future application enhancements. This approach of splitting classes to implement a clearly defined functionality is known as the “Single Responsibility Principle” and adheres to SOLID principles. A detailed description of the class library is provided in Table 2.8. This table contains all the project logic that will be integrated into the application’s presentation layer.

Table 2.8.

Detailed description of the GenericStore.BL.IMPL project

Class name	Method name with parameters	Description of the method
Service	–	Serves as a service attribute for all business logic classes
CartServices		
AddClient Information	DoAction (ClientInformationDTO request)	Adding customer’s personal information with session tokens in mind
AddProduct ToCart	DoAction (CartStorehouseQtyDTO request)	Adding a product to the customer’s cart with session tokens in mind
GetCart	DoAction()	Viewing the cart with session tokens in mind
GetClient Information	DoAction()	Retrieving customer’s personal information with session tokens in mind
GetOrder Information	DoAction()	Retrieving order-specific personal information for the customer with session tokens in mind
RemoveProduct FromCart	DoAction (CartStorehouse QtyDTO request)	Removing a product from the cart with session tokens in mind
OrderAdminServices		
GetOrder Information	DoAction(int id)	Retrieving customer’s personal information at the administrator level

Table 2.8. (continue)

GetOrders Information	DoAction(int status)	Retrieving the status of customer orders at the administrator level
UpdateOrder Information	DoActionAsync(int id)	Updating customer orders at the administrator level
OrderServices		
CreateOrder Entity	DoAction (request) та CreateOrderReference()	Creating an order at the user level
GetOrder Information	DoAction(string reference)	Getting order information at the user level
ProductAdminServices		
CreateProduct Entity	DoAction (CreateProduct RequestDTO request)	Creating a new product at the administrator level
DeleteProduct Entity	DoAction(int id)	Deleting a product at the administrator level
GetProductEntity	DoAction(int id)	Getting a product
GetProducts	DoAction()	Getting a list of products
UpdateProduct Entity	DoAction (UpdateProduct RequestDTO request)	Updating a product at the administrator level
ProductServices		
GetProduct	DoAction(string name)	Getting a product at the user level
GetProducts	DoAction()	Getting a list of products
StorehouseAdminServices		
CreateStorehouse	DoAction (CreateStorehouse RequestDTO request)	Creating a new warehouse at the administrator level

Table 2.8. (continue)

DeleteStorehouse	DoAction(int id)	Deleting from the warehouse at the administrator level
GetStorehouse	DoAction()	Getting product information at the warehouse as an administrator
UpdateStorehouse	DoAction (UpdateRequestDTO request)	Updating the warehouse at the administrator level

The user interaction layer, characterized by the presentation, is implemented as part of the software execution for this work within the software module called “GenericStore.UI”. This project embodies the visual part of the web application and includes various classes and directories related to the user interface logic. Within this layer, specifically in the “wwwroot” directory, CSS and JS code for the application is located. Additionally, there are application controllers and user interface pages implemented using the “Razor Pages” technology.

The application also includes a session control manager, validation logic, a shopping cart component, and a visual model for creating login and password for authentication. The web application also includes classes such as “Program”, “RegisterOfServices”, “Startup”, and “Dockerfile” that allow containerization of the entire web application.

2.5.1 Implementation description of the main tasks of the software

The task of generating a text file with lists of products from internet stores is implemented in the presentation layer of the “GenericStore.UI” project. To accomplish this task, a file with graphic content and interaction with task logic was created, along with a class for implementing the logic of generating the file and extracting data. Extracting data from other websites requires the use of the HTML Agility Pack library, which includes convenient classes and methods for this purpose.

The order processing framework task is implemented in the business logic and presentation layers and involves a three-stage dynamic order processing process for customers. This functionality is available only to administrators.

The task of creating a dynamic user interface is implemented in the presentation layer of the application and has two main aspects. The first aspect is the dynamism of processes, pages, and interface elements. The second aspect is the ability to dynamically change the interface background, including changing the color to "random," dynamic gradient, and adaptive black with the option to save the state in the local computer storage.

The task of conducting financial transactions is implemented in the presentation layer of the application and includes retrieving customer data and processing order payments through the "Stripe" API service.

2.5.2 Implementation of service registration logic

Within the scope of the review of service registration logic, which involves creating the "Service" class with the implementation of the abstract class "Attribute" from the "Reflection" library. This class, in turn, represents a late-binding process achieved by passing type metadata and reading it using the "Reflection" library.

Each class located in the business logic layer is marked with a significant attribute called "Service." In the presentation layer, there is a class called "RegisterOfServices," which automates the processing of services marked specifically with the "Service" attribute.

As a result, the method of the "RegisterOfServices" class is called in the initial application class named "Startup", thereby injecting all services into the "AddTransient" method, which, in turn, adds the service to the application upon each new project initialization.

2.6 Data security analysis

2.6.1 Authorization and authentication

To ensure the security of the application's data, a class, interface, and session management methods called "SessionControlManager" were developed and implemented. These methods handle the addition and removal of session tokens within a session by adding and converting tokens into JSON code. Sessions also have a timer set to thirty minutes from the moment of creation and automatically delete themselves when the time expires, thereby removing stored session data. Additionally, for data security, an authentication process was developed and implemented using the "IdentityUser" and "Claim" classes, which directly provide data for authentication, validate it, and store it in the database.

Conclusions on the section

Therefore, this section provides a detailed description of software modeling and construction. It describes in detail the general processes that users and administrators of the software must undergo. The necessary development tools are reviewed and justified, including a description of the database architecture with entity descriptions, schemas, and database diagrams. The software architecture is also described, including necessary UML diagrams and a description of the software architecture with tables of classes and methods implemented in the project, along with an analysis of the choice of software architecture. The software was designed and implemented with consideration for dependency injection methods. The construction of the software is also discussed, including a description of the implementation of the core tasks of the software and the implementation of service registration logic. Finally, a security analysis of data is conducted, including a section that discusses the authentication and authorization processes in the project.

3 QUALITY ANALYSIS AND SOFTWARE TESTING

3.1 Software quality analysis

The quality of software is defined as a research area that describes the desired attributes of software products [33]. In turn, software testing is a necessary process for evaluating and verifying the software product's compliance with the specified requirements and expected functionality, which should not be omitted during development. Software testing aims to minimize errors, reduce development costs, and enhance the productivity of the web application [34].

During the testing of the web application, a testing logic was developed, which uses the testing pyramid. The testing pyramid typically operates at three levels:

– unit tests;
– integration tests;
– end-to-end tests.

Unit tests cover the web application services located in the GenericStore.BL.IMPL class library is shown in Fig. 2.18. Services are crucial components of the application that need to be tested, and understanding their logic is essential. This approach helps identify errors at an early stage of development.

Integration tests cover most of the web application controllers located in the GenericStore.UI class library. Controllers are important for integration testing because they interact between architectural layers and act as mediators between them. Black-box testing, which doesn't require knowledge of the internal implementation, is used for integration testing.

To test the project, the xUnit, Moq, AutoFixture, Microsoft.AspNetCore.Hosting, Microsoft.AspNetCore.TestHost, and System.Net.Http libraries are used. The xUnit library serves as a testing tool [35]. The Moq library is used for creating artificial objects and simulating functionality [36]. Additionally, the AutoFixture library simplifies the creation of objects of various types and improves the safety of unit tests.

In conclusion, software testing plays a vital role in creating and implementing various levels of tests.

3.2 Description of testing processes

To implement proper testing of the e-commerce web application, a testing pyramid approach is employed.

The foundation of the automated testing pyramid primarily consists of unit tests, followed by integration tests, system tests, functional tests, acceptance tests, and, as the final stage, user interface tests. The structure of the automated testing pyramid for this software is depicted in Fig. 3.1.

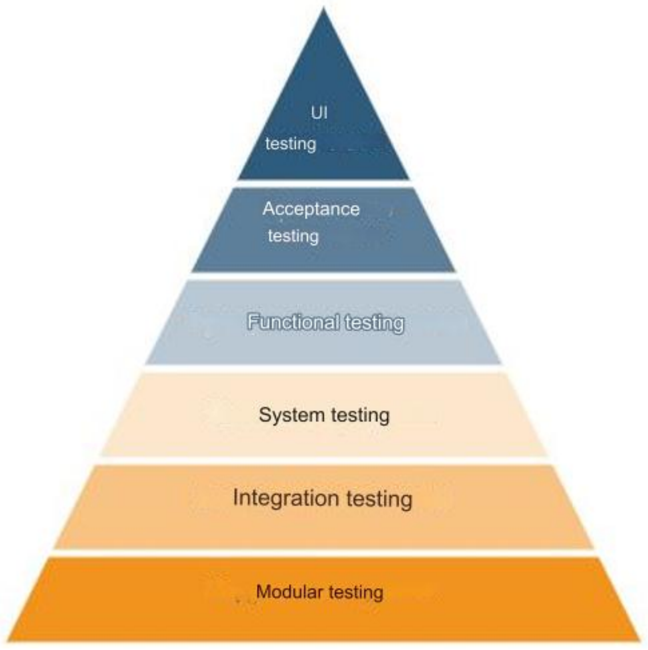


Fig. 3.1. Pyramid of automated software testing

This software undergoes a sequence of different types of tests, such as:

– Unit testing, which verifies whether each software module works correctly;

– Integration testing, which ensures that all software components interact cohesively and correctly;

– System testing, which thoroughly tests the integrated software product for compliance with all requirements;

– Functional testing, which checks the modeling of business scenarios according to functional requirements and performs functional verification using black-box tests;

– Acceptance testing, which verifies whether the entire application functions correctly;

– User interface testing, which checks the graphical user interface of the software [37].

3.3 Description of the test case

To implement effective testing for the e-commerce web application, the testing pyramid concept was employed. Therefore, this section implements each testing stage as depicted in Fig. 3.1.

As each testing stage is reviewed and implemented, the e-commerce web application using web scraping with an optimized product reading process becomes progressively more covered by tests, which will subsequently help in detecting errors more efficiently.

3.3.1 Unit testing

Unit testing is a method of software testing where individual blocks and components of the software code are tested separately. Its goal is to confirm that each unit of the code functions correctly [38].

To implement unit testing, testing technologies that isolate test components from the logic of other parts of the program were used. Libraries like xUnit and Moq were employed for creating unit tests. Consequently, a separate project named GenericStore.UnitTests was created for this purpose. A part of the structure of this project is illustrated in Fig. 3.2.

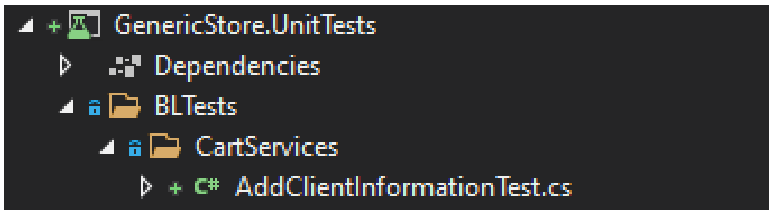
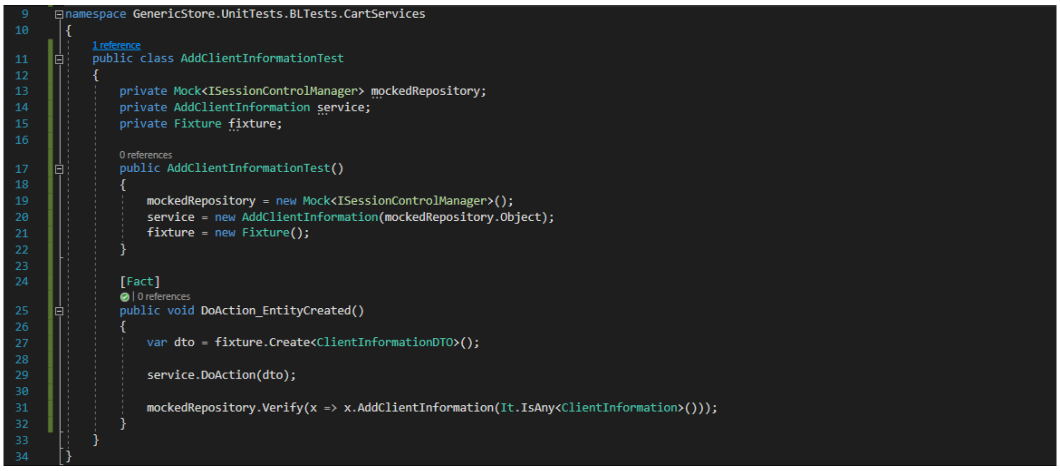


Fig. 3.2. Project structure for unit testing GenericStore.UnitTests

As mentioned earlier, xUnit and Moq libraries were used for unit testing, and a dedicated project was created for this purpose.

As an example of unit testing, consider the test class from the section responsible for handling client information, named “AddClientInformationTest”. This class includes a constructor with initial objects that are initialized and the actual test method. The implementation of the unit test “AddClientInformationTest” is shown in Fig. 3.3.

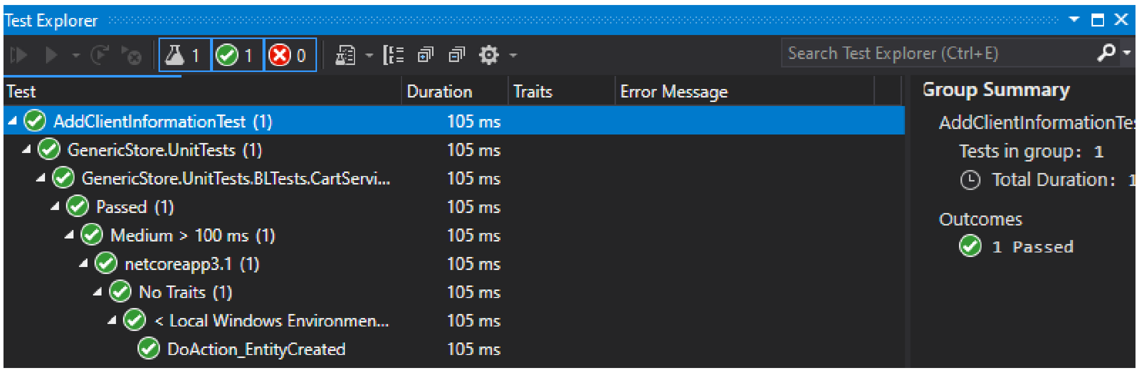


```
namespace GenericStore.UnitTests.BLTests.CartServices
{
    1 reference
    public class AddClientInformationTest
    {
        private Mock<ISessionControlManager> mockedRepository;
        private AddClientInformation service;
        private Fixture fixture;

        0 references
        public AddClientInformationTest()
        {
            mockedRepository = new Mock<ISessionControlManager>();
            service = new AddClientInformation(mockedRepository.Object);
            fixture = new Fixture();
        }

        [Fact]
        | 0 references
        public void DoAction_EntityCreated()
        {
            var dto = fixture.Create<ClientInformationDTO>();

            service.DoAction(dto);

            mockedRepository.Verify(x => x.AddClientInformation(It.IsAny<ClientInformation>()));
        }
    }
}
```

Fig. 3.3. AddClientInformationTest unit class test

The constructor includes a “mock” repository and manager interface and initializes the tested class, service, and object from the Fixture library, which allows artificially initializing objects from the project’s source code. After describing the constructor, the implementation of the unit test “AddClientInformationTest” follows. For this software, this test includes creating an “artificial” entity and testing it using the xUnit and Moq libraries.

After writing a unit test, it’s important to run it to check for errors, as shown in Fig. 3.4.



```
Test                                        Duration   Traits   Error Message
AddClientInformationTest (1)                105 ms
  GenericStore.UnitTests (1)                105 ms
    GenericStore.UnitTests.BLTests.CartServi... 105 ms
      Passed (1)                            105 ms
        Medium > 100 ms (1)                 105 ms
          netcoreapp3.1 (1)                 105 ms
            No Traits (1)                   105 ms
              < Local Windows Environmen... 105 ms
                DoAction_EntityCreated      105 ms

Group Summary
  AddClientInformationTe
    Tests in group: 1
    Total Duration: 1
  Outcomes
    1 Passed
```

Fig. 3.4. Successfully completed unit test

As a result, unit tests were created to effectively check for errors during the development of the e-commerce web application using web scraping with an optimized product reading process.

3.3.2 Integration testing

Integration testing is a type of testing designed to check the interaction and combinations of various modules [39]. To create integration tests, a separate project was created using the xUnit, TestHost, and Test.Sdk libraries, named "GenericStore.IntegrationTests". The structure of this project is shown in Fig. 3.5.

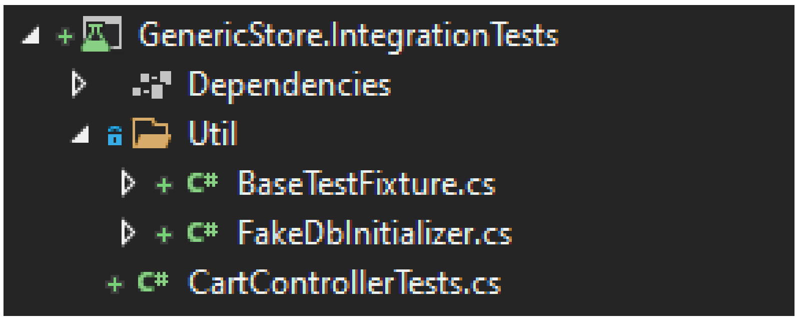
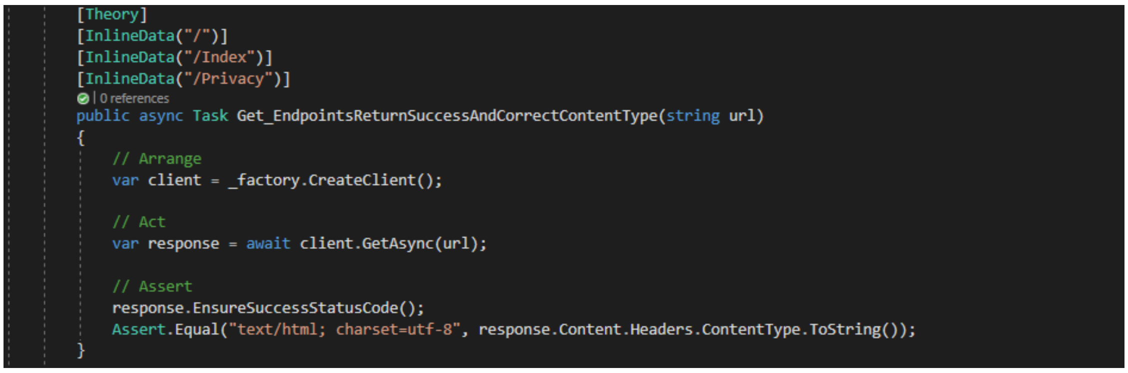

GenericStore.IntegrationTests
Dependencies
Util
BaseTestFixture.cs
FakeDbInitializer.cs
CartControllerTests.cs

Fig. 3.5. Part of structure of the project for integration testing GenericStore.IntegrationTests

As mentioned earlier, the creation of integration tests involves the use of the xUnit, TestHost, and Test.Sdk libraries, so a project utilizing these libraries was created. This project includes a directory with two necessary initialization classes. The first class, called "BaseTestFixture" contains objects such as "TestServer", "GenericStoreDbContext", "HttpClient", and "FakeDbInitializer". These objects are initialized in the class's constructor as they are required for creating basic test fixtures. A fixture, in turn, helps save the system's state to a file and then restore it. It also initializes the fake database in its constructor. The second class, named "FakeDbInitializer," contains a static method "Initialize," which initializes the fake "GenericStoreDbContext" database. "FakeDbInitializer" serves as an artificial database that is initialized along with the fixtures.

Using the example of the "ProductsControllerTests" an integration test for a products controller, one can see how these tests are created. This class includes a constructor with objects for initialization and the actual test. The implementation of the “ProductsControllerTests” integration test can be seen in Fig. 3.6.

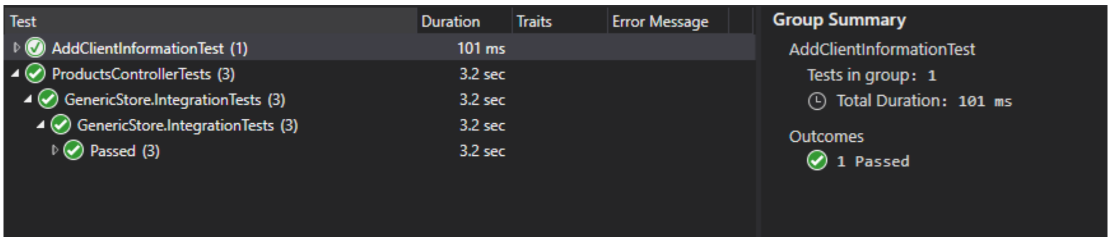


```
[Theory]
[InlineData("/")]
[InlineData("/Index")]
[InlineData("/Privacy")]
0 references
public async Task Get_EndpointsReturnSuccessAndCorrectContentType(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.GetAsync(url);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal("text/html; charset=utf-8", response.Content.Headers.ContentType.ToString());
}
```

Fig. 3.6. Integration class-test ProductsControllerTests

Overall, this integration test checks the correctness of URL returns when the controller is in operation.

As the final stage of an integration test, it needs to be executed to check for errors, as shown in Fig. 3.7.



```
Test                                      Duration   Traits   Error Message   Group Summary
▷ ✓ AddClientInformationTest (1)          101 ms                              AddClientInformationTest
◢ ✓ ProductsControllerTests (3)           3.2 sec                               Tests in group: 1
  ◢ ✓ GenericStore.IntegrationTests (3)   3.2 sec                               Total Duration: 101 ms
    ◢ ✓ GenericStore.IntegrationTests (3) 3.2 sec                             Outcomes
      ▷ ✓ Passed (3)                      3.2 sec                               ✓ 1 Passed
```

Fig. 3.7. Successfully completed integration test

Therefore, these integration tests were created to verify the logic of controllers during the development of the e-commerce web application.

3.3.3 System testing

System testing is a level of testing that involves the comprehensive testing of an integrated software product to ensure compliance with all requirements. Its main purpose is to evaluate and identify possible inconsistencies between the developed application and its initial objectives [40].

As a result, during the development of the e-commerce web application using web scraping with an optimized product reading process, thorough checks of all pages, blocks, and user interface modules were conducted. This helped identify several errors that were corrected during the development process.

3.3.4 Functional testing

Functional testing is a type and level of software testing that verifies the application's compliance with functional requirements. The main goal of functional tests is to check each function of the software application using appropriate input and output data to ensure compliance with functional requirements [41].

To implement functional testing, the functional requirements of the application were used, which can be found in Table 1.3.

Functional tests from the user's perspective of the application are provided in Table 3.1.

Table 3.1.

Functional testing by the application user

Test 1	
Purpose of the test	Checking the main product page
Input	User's personal data, user's credit card data
Description of the test	Filling out the personal information window, then filling out the user's credit card information window and clicking on "Pay"
Expected result	A successful monetary transaction must take place
Actual result	Successful monetary transaction
Test 2	
Purpose of the test	Checking for Adding Items to Cart
Initial state	Open the page with a detailed product description
Input	–
Description of the test	Add one or several items to the cart
Expected result	The cart should be filled with the specified quantity of items
Actual result	The cart is filled with the specified quantity of items

Table 3.1. (continue)

Test 3	
Purpose of the test	Payment order confirmation
Initial state	The page for entering personal information is open, followed by the page for entering credit card information for order payment by the user
Initial state	The main page with a list of all products is open
Input	–
Description of the test	As a user, review how the products are displayed on the main page
Expected result	The main page for viewing products should be displayed
Actual result	The main page for viewing products is displayed

Mixed functional testing, both on the part of the user and on the part of the application administrator, is shown in Table 3.2.

Table 3.2.

Mixed functional testing by the user and the application administrator

Test 1	
Purpose of the test	Checking for changing the application background
Initial state	Open page
Input	–
Description of the test	Click the change background button in the application menu panel
Expected result	After clicking the button, the background should change
Actual result	After clicking the button, the background has changed
Test 2	
Purpose of the test	Authentication check
Initial state	Open page
Input	Application token

Table 3.2. (continue)

Description of the test	Add products to the shopping cart or log in as an administrator and reload the website
Expected result	After reloading the website, the previous actions should be restored, preserved
Actual result	After reloading the website, the previous actions' data has been restored and preserved

Functional testing by the application administrator is shown in Table 3.3.

Table 3.3

Functional testing by the administrator

Test 1	
Purpose of the test	Authentication verification
Initial state	Open page
Input	Administrator credentials
Description of the test	Filled fields of the authorization page
Expected result	Successful authorization should occur
Actual result	Successful authorization
Test 2	
Purpose of the test	Administrative panel access check
Initial state	Open page
Input	–
Description of the test	On the main page, there should be a button in the menu for accessing the admin panel
Expected result	There should be a button for accessing the admin panel
Actual result	A button for accessing the admin panel is present
Test 3	
Purpose of the test	Checking product viewing and editing
Initial state	Open administrative panel

Table 3.3. (continue)

Input	Input data into the fields
Description of the test	Open the product sub-panel, create and save a product
Expected result	The product should be viewable, editable, deletable, and savable
Actual result	The product is viewable, editable, deletable, and savable
Test 4	
Purpose of the test	Checking proper generation of the product list file
Initial state	Open the administrative panel
Input	Input the required link into the field
Description of the test	Open the file generation sub-panel, enter the necessary data into the field, and click the file generation button to the local system
Expected result	A generated and entered product list file should be present in the local “Downloads” directory
Actual result	A successfully generated and entered product list file is present in the local “Downloads” directory
Test 5	
Purpose of the test	Checking order processing viewing and editing
Initial state	Open the administrative panel
Input	Input data into the fields
Description of the test	Open the order processing sub-panel, edit a possible order
Expected result	The order should be viewable and processed
Actual result	The order is viewable and has been processed
Test 6	
Purpose of the test	Checking product quantity viewing and editing
Initial state	Open the administrative panel
Input	Input data into the fields

Table 3.3. (continue)

Description of the test	Open the product inventory sub-panel, create and save a product in inventory
Expected result	The product in inventory should be viewable, editable, deletable, and savable
Actual result	The product in inventory is viewable, editable, deletable, and savable
Test 7	
Purpose of the test	Checking the creation of new users
Initial state	Open the administrative panel
Input	Input data into the fields
Description of the test	Open the new user creation sub-panel, create and save one user
Expected result	A user should be created
Actual result	A user has been created

3.3.5 Acceptance testing

User Acceptance Testing (UAT) is one of the final stages of software development. Its purpose is to test the software in real-world scenarios to determine if it meets its intended objectives [42]. After completing all the aforementioned levels of testing, User Acceptance Testing was performed. It involved checking the e-commerce web application to ensure the correct delivery of results in various scenarios.

As a result, several issues were identified and addressed, including an error during the payment process, which was promptly rectified, as well as an issue in the order processing.

3.3.6 User interface testing

User Interface Testing is a type of software testing that evaluates the graphical user interface of the software. Its goal is to ensure the functionality of the application by examining interfaces, pages, and user interface elements such as menus, buttons, and icons, in accordance with specifications [43]. After implementing this testing, several interface-related issues were discovered, such as dynamic menu and block changes when resizing the screen and dynamic button states during user interaction. These issues were corrected during the application's development.

Conclusions on the section

This section analyzed the software testing process and described the results of the assessments conducted. A description of the software's acceptance test case was provided, encompassing all levels of testing, including unit testing, integration testing, system testing, functional testing, user acceptance testing, user interface testing, and their incorporation into the project. During the execution of these testing types, the necessary tests were conducted, with unit testing covering 100% of the web application's services and integration testing covering the majority of controllers. Functional testing was also conducted from both the user and administrator perspectives, as well as mixed functional testing. Several issues were identified during testing, all of which were successfully rectified.

As a result of the testing, it was confirmed that the program operates as specified. Thus, the developed web application is ready for use.

4 SOFTWARE IMPLEMENTATION AND MAINTENANCE

4.1 Software deployment

The e-commerce web application using web scraping with an optimized product reading process consists of two parts: server-side and client-side. Before providing a detailed description of these components, let’s explore the available software deployment methods.

4.1.1 Overview of available software deployment methods

There are numerous methods for deploying software in production, both in terms of implementation and user access. Therefore, choosing the right deployment strategy is a crucial aspect of the software development lifecycle.

Let’s review the available software deployment methods for the user, along with explanations:

– Standard deployment: The user obtains the necessary project directory and deploys it on their local system through any download method;

– Docker-based deployment: Users can deploy the project using containerization technology if they have the project directory;

– AWS services deployment: Users can access only the graphical interface of the project remotely by utilizing a link to the project’s main page, concealing the project’s logic from them.

Additionally, let’s consider deployment methods that can be used for editing processes and code of this e-commerce web application when needed:

– Standard deployment;

– Recreate: Version A is terminated, and then version B is launched;

– Rolling deployment: Version B is gradually deployed and replaces version A;

– Shadow: Version B receives real traffic alongside version A and does not affect the response [44].

After thoroughly examining various software deployment methods, the decision was made to use standard deployment for reviewing the structure, code, and graphical interface. Deployment through AWS services are used to access only the graphical part of the project.

4.1.2 Server-side deployment

To deploy the server side of the web application, you need to:

– Install Visual Studio 2016 and above with the necessary settings for ASP.NET Core platform and C# programming language;

– Install Microsoft SQL Server 2016 and above;

– Open the project with the server-side part in the aforementioned IDE;

– Migrate the database structure to the DBMS;

– Launch the GenericStore.UI project via Internet Information Services (IIS).

4.1.3 Client-side deployment

For deploying the client-side of the web application, you need to:

– Install Visual Code for working with the program’s code;

– Open the folder with the client-side project and run the GenericStore.UI project.

4.1.4 Providing a secure communication channel

Regardless of the chosen deployment method, this software ensures a secure communication channel between the client and server parts of the program. Within the selected software architecture and specified interaction protocols to establish a secure communication channel, the HTTPS protocol has been used, which includes data encryption performed using an SSL certificate in compliance with protocol standards.

4.1.5 Working with the software

Detailed step-by-step instructions for working with the server and client parts are given in the next section.

Conclusions on the section

In this section, we described the software deployment process, including an overview of available deployment methods, server-side and client-side deployment, and system requirements for deploying the web application. We also discussed providing a secure communication channel and provided a reference to the “User guide” section.

5 USER GUIDE

5.1 General information

The e-commerce web application using web scraping with an optimized product reading process, along with a base for an order processing conveyor framework and a dynamic interface is utilized for browsing and managing a personalized e-commerce platform.

The software features a dynamic, simple, and user-friendly interface.

User-side client functionalities include:

- viewing products;
- adding products to the cart;
- canceling the cart;
- order authentication;
- order payment;
- receiving a payment confirmation message;
- changing the background.

Administrator-side user functionalities include:

- authorization;
- authentication;
- viewing and editing products;
- generating a text file and description of products from internet stores;
- processing the order processing conveyor framework;
- viewing and editing the quantity of products in stock;
- creating new users;
- changing the background.

5.2 Preparation for work

5.2.1 System requirements for correct operation

For the successful operation of this application from the user's perspective, the following requirements must be met:

– availability of a computer, tablet, or mobile device;
– availability of an internet browser;
– access to the necessary website link;
– internet access.

For the successful operation of this application from the user and administrator perspective, the following requirements must be met:

– availability of a computer, tablet, or mobile device;
– availability of administrator credentials for authorization;
– availability of an internet browser;
– access to the necessary website link;
– internet access.

5.2.2 Software installation

To start working with the program, both the client and the administrator need to launch the website using the provided link.

5.2.3 Checking correct operation

To check the proper functioning of the web application, users can use the application's website link. If everything works without errors, and the site and database data of the web application are displayed and editable, then everything is working correctly.

5.3 Working with the application

To begin using the web application, users, both clients and administrators, need to visit the generated link.

This link is created using AWS services such as EC2, RDS and their instances.

Upon launching the web application, users will see the main page of the site, which includes the main menu and the product list, as depicted in Fig. 5.1.

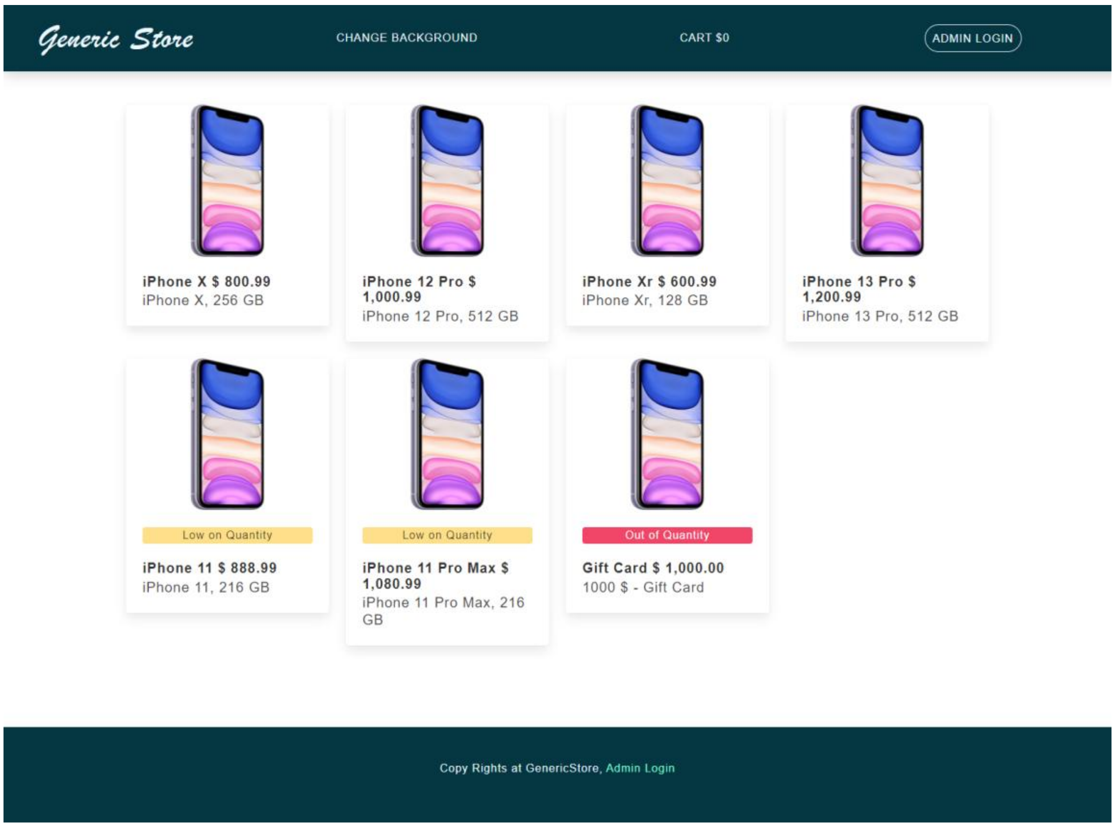


Fig. 5.1. Initial, main page of the application

On the main page, the user from the client side has access to a menu that includes the ability to click on the application's logo to return to the main page, the background change functionality, the shopping cart functionality, and administrator login.

Reviewing the application menu reveals the background change functionality, which includes:

– changing the background to a “random color”;

– changing the background to a dynamic background;

– changing the background to a “dark” background mode.

The options for changing the background are shown in Fig. 5.2.

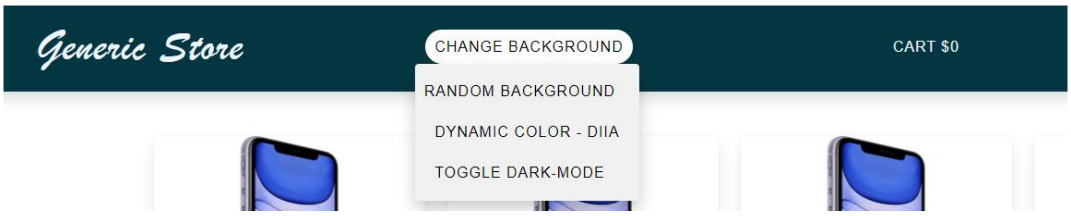


Fig. 5.2. Main menu, background change options

By clicking on the first “random background” option, the app's background takes on a “random” RGB color. The process of changing the color is shown in Fig. 5.3.

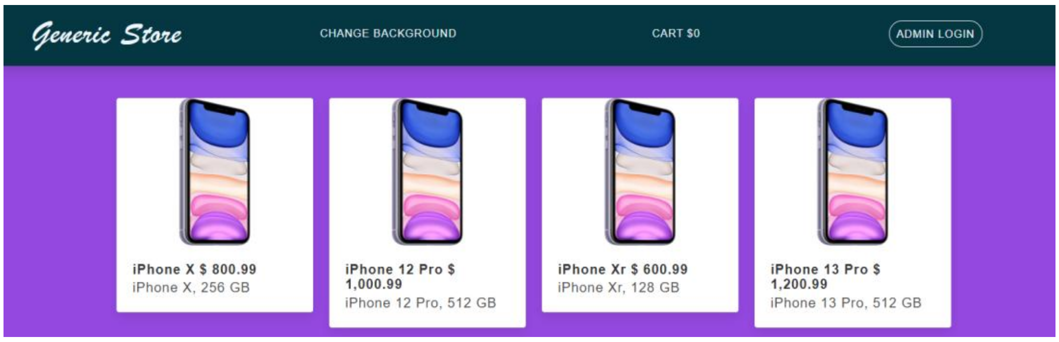


Fig. 5.3. The process of changing the color from the “random color” option

By clicking on the second option “dynamic color – Diia”, the background of the application becomes dynamic. The dynamic color process is shown in Fig. 5.4.

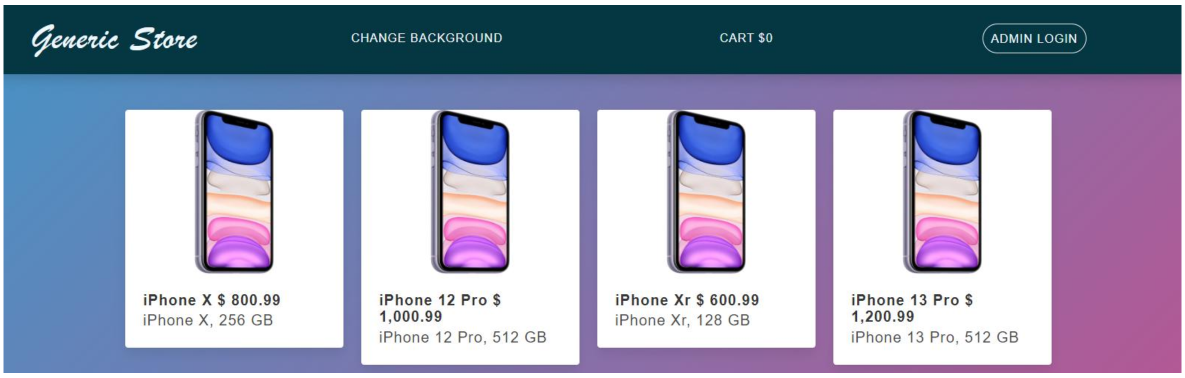


Fig. 5.4. Dynamic color process from the “dynamic color action” option

Clicking on the third option “toggle dark-mode” changes the application’s background to dark mode. The dark mode background replacement process is shown in Fig. 5.5. This mode is stored in the local storage of the site.

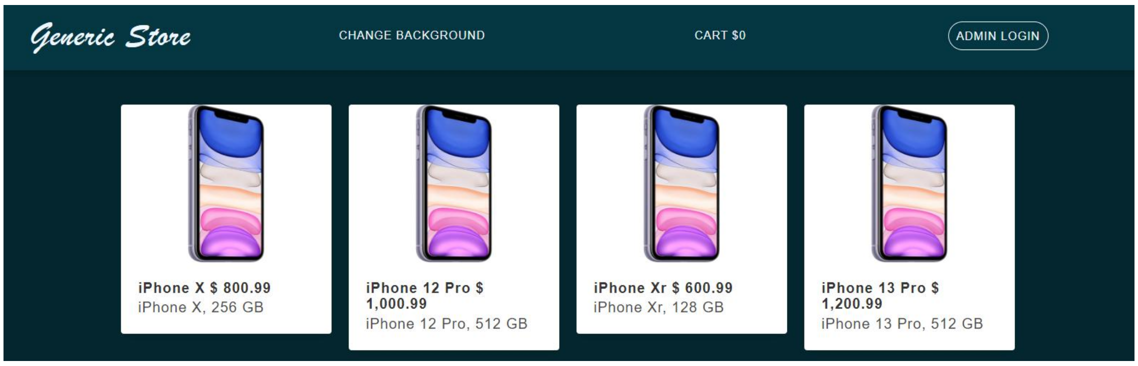
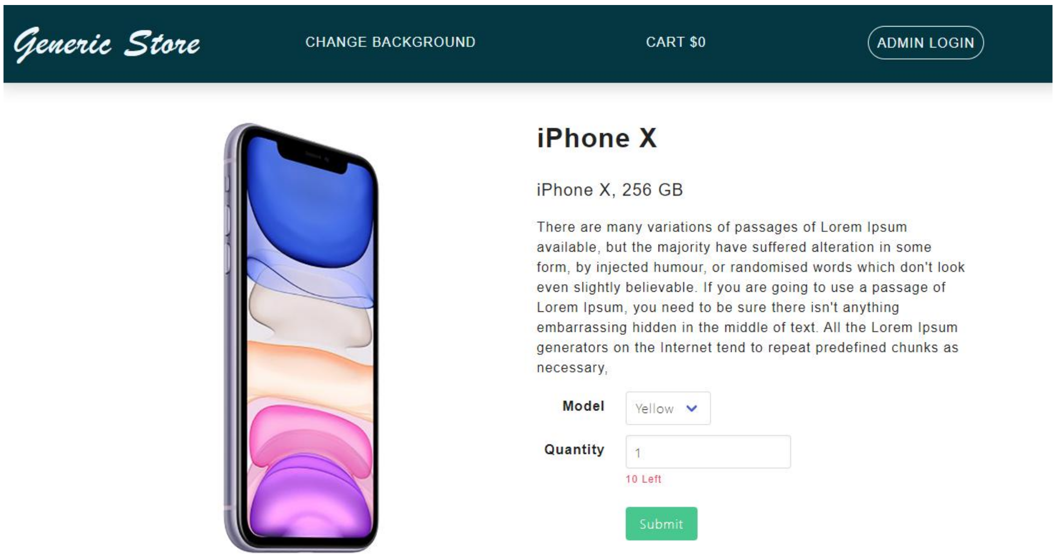


Fig. 5.5. The process of replacing the background with dark mode from the "toggle dark-mode" option

Clicking on a product allows the user to navigate to the detailed product page, which includes photos, a detailed description, model, quantity, and an "approve" button to add the product to the shopping cart. The detailed product description page is shown in Fig. 5.6.



iPhone X

iPhone X, 256 GB

There are many variations of passages of Lorem Ipsum available, but the majority have suffered alteration in some form, by injected humour, or randomised words which don't look even slightly believable. If you are going to use a passage of Lorem Ipsum, you need to be sure there isn't anything embarrassing hidden in the middle of text. All the Lorem Ipsum generators on the Internet tend to repeat predefined chunks as necessary,

Model: Yellow

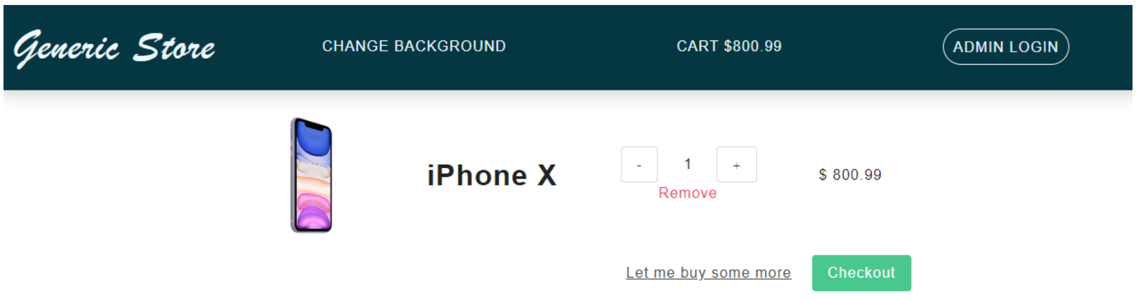
Quantity: 1

10 Left

Submit

Fig. 5.6. Detailed product description page

By selecting the product model and quantity and clicking the "approve" button, the user is taken to the shopping cart page, where the selected product is already present. If the user wishes to purchase additional items, they can exit the cart by clicking the "buy more" button and continue selecting items, as the cart is saved throughout the session. The cart page is shown in Fig. 5.7.

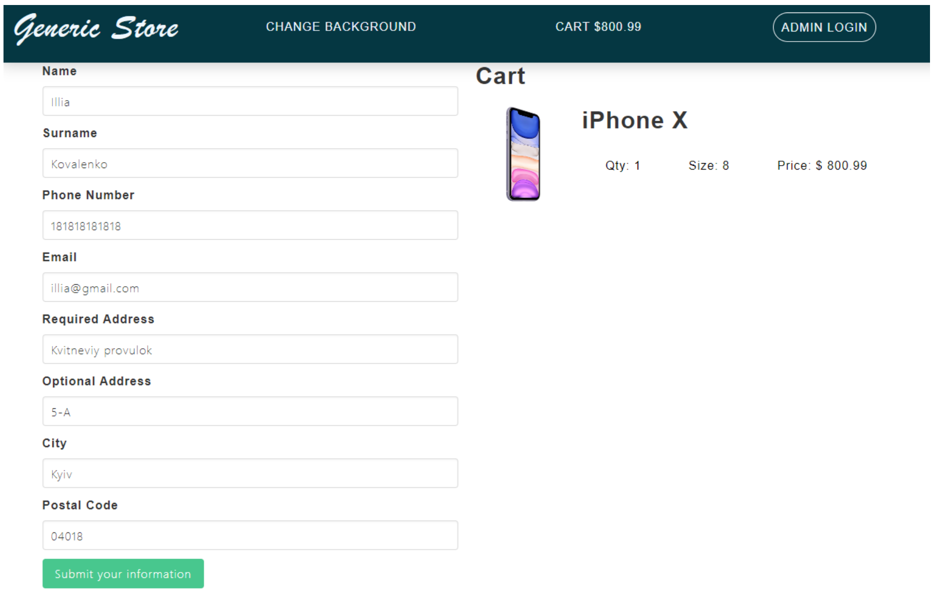


iPhone X - 1 + Remove $ 800.99

Let me buy some more Checkout

Fig. 5.7. Shopping cart page

Clicking the "checkout" button takes the user to the personal information input page for the order, which includes fields for entering information and a reduced image of the product, quantity, characteristics, and price. The page for entering user personal information is shown in Fig. 5.8.



Name	Illia
Surname	Kovalenko
Phone Number	181818181818
Email	illia@gmail.com
Required Address	Kvitneviy provulok
Optional Address	5-A
City	Kyiv
Postal Code	04018

Fig. 5.8. User’s personal information entry page

Clicking the “Submit your information” button takes the user to the order payment page. The order payment page is shown in Fig. 5.9.

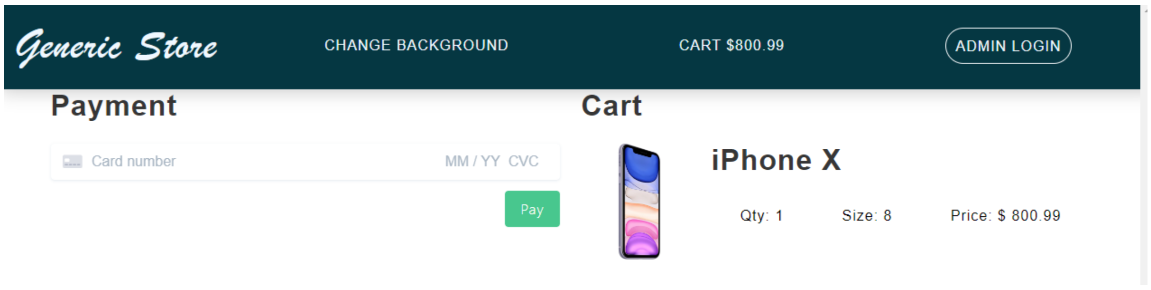
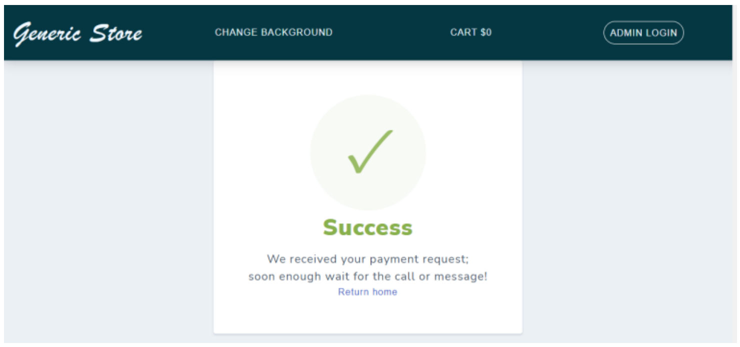


Fig. 5.9 – Order payment page

Clicking the “pay” button takes the user to the successful order payment page. The successful order payment page is shown in Fig. 5.10.



We received your payment request;
soon enough wait for the call or message!

Fig. 5.10. Page of successful payment of the order

Clicking the “return home” button allows the user to return to all the above-mentioned options or wait for their order. Now, let’s move on to the administrator’s perspective. On the main page, Fig. 5.1, the administrator has the option to use the “admin login” button. Clicking on this button takes the administrator to the authorization page. The administrator authorization page is shown in Fig. 5.11.

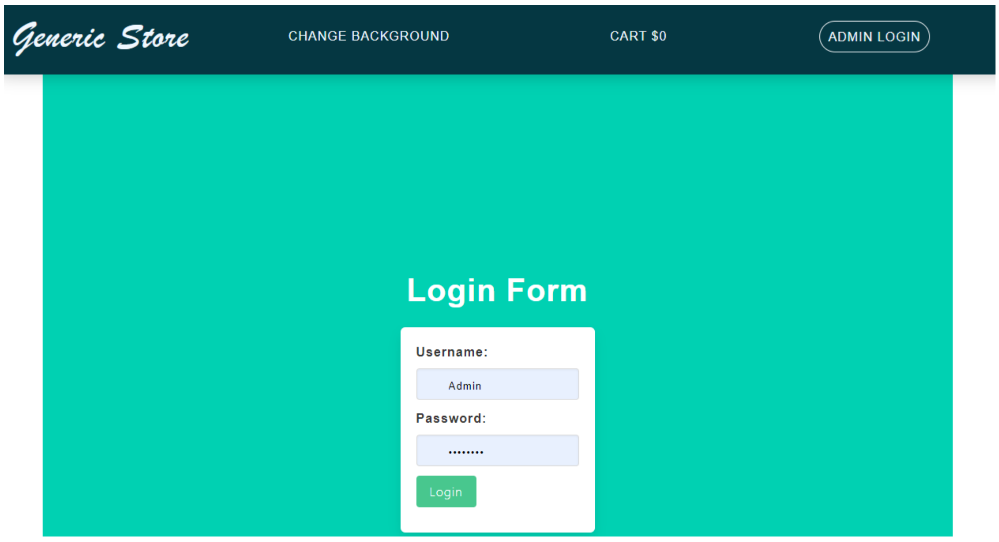
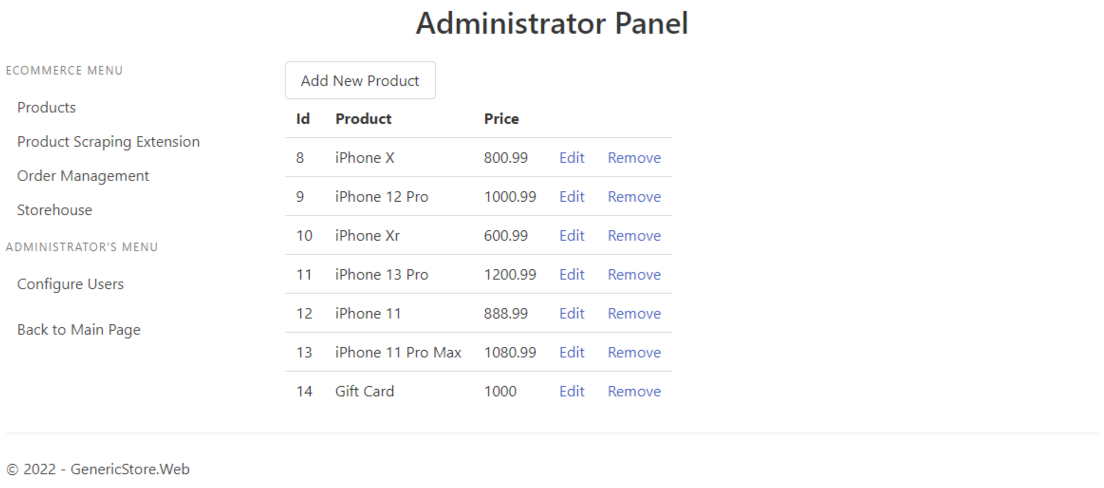


Fig. 5.11. Administrator authorization page

Clicking the “login” button allows the administrator to access the administrator panel page, administration. The administration panel page is shown in Fig. 5.12.



Administrator Panel

ECOMMERCE MENU
- Products
- Product Scraping Extension
- Order Management
- Storehouse

ADMINISTRATOR'S MENU
- Configure Users
- Back to Main Page

Add New Product

Id	Product	Price		
8	iPhone X	800.99	Edit	Remove
9	iPhone 12 Pro	1000.99	Edit	Remove
10	iPhone Xr	600.99	Edit	Remove
11	iPhone 13 Pro	1200.99	Edit	Remove
12	iPhone 11	888.99	Edit	Remove
13	iPhone 11 Pro Max	1080.99	Edit	Remove
14	Gift Card	1000	Edit	Remove

© 2022 - GenericStore.Web

Fig. 5.12. Admin authorization page, and product management page

Upon reaching the administration panel page, the administrator has two sub-panels, including the “e-commerce menu” and the “administrator’s menu”. Each sub-panel has its functionality, dynamically displayed when clicked.

The first subpanel of the “e-commerce menu” includes functionality for managing products, generating a file of product listings from well-known internet stores, a base for order processing conveyor framework, and inventory management.

The product management functionality is depicted in Fig. 5.10. This feature allows adding, viewing, editing, and deleting products within the application.

The functionality for generating a file of product listings from well-known internet stores is illustrated in Fig. 5.13. It is implemented using the HTML Agility Pack library for network data retrieval. This page includes a field with available websites for reading, a field for entering the link to an available internet product, and a button for generating a file to the local system or computer.

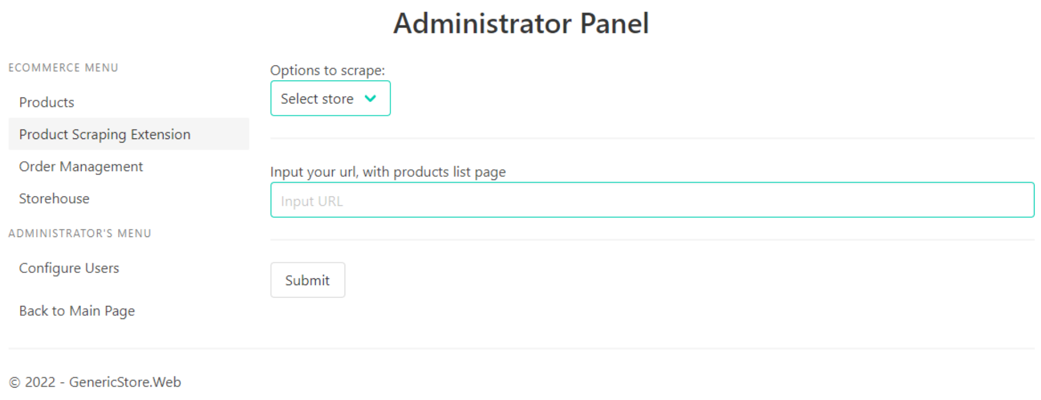
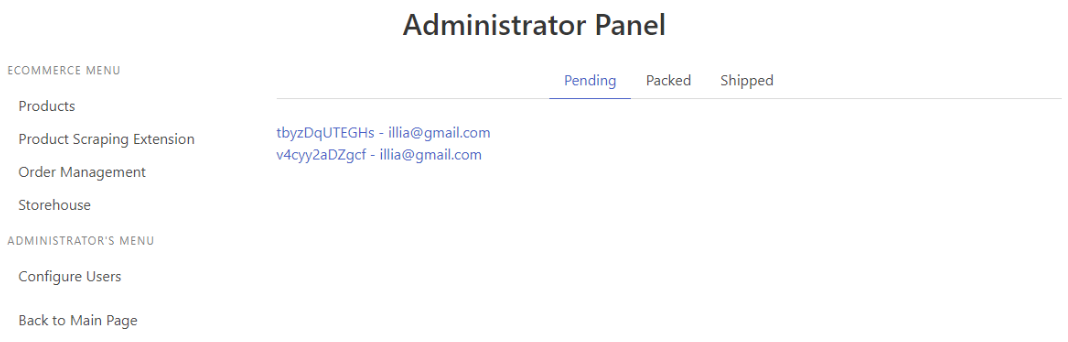


Fig. 5.13. Page for generating a product list file from well-known internet stores

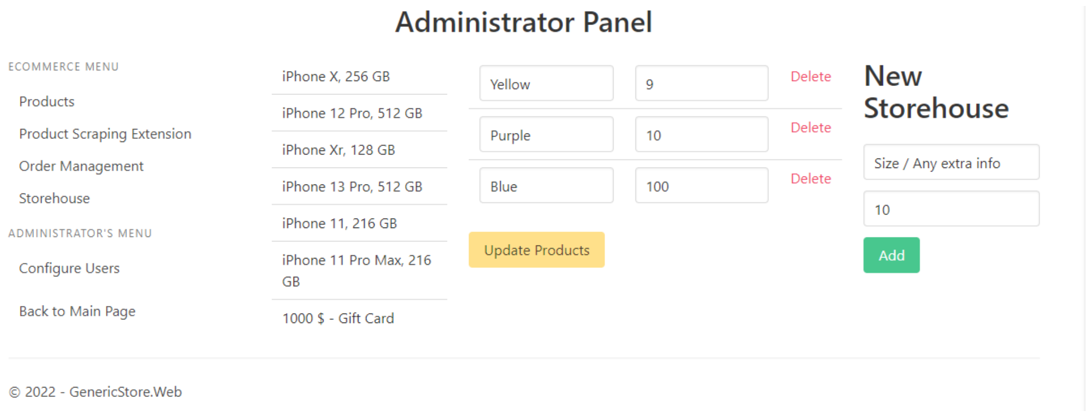
The functionality of the operational order processing conveyor framework is depicted in Fig. 5.14. It allows for a three-stage order processing process, including the statuses of pending, packaging, and order shipment. The page displays the order code and the customer’s email.



tbyzDqUTEGHs - illia@gmail.com
v4cyy2aDZgcf - illia@gmail.com

Fig. 5.14. Working order processing pipeline frame page

The inventory management functionality is illustrated in Fig. 5.15. It lists the inventory of created items, and clicking on one of them opens a dynamic panel for adding the quantity of items with characteristics to the inventory.


Administrator Panel

ECOMMERCE MENU
Products
Product Scraping Extension
Order Management
Storehouse
ADMINISTRATOR'S MENU
Configure Users
Back to Main Page

iPhone X, 256 GB
iPhone 12 Pro, 512 GB
iPhone Xr, 128 GB
iPhone 13 Pro, 512 GB
iPhone 11, 216 GB
iPhone 11 Pro Max, 216 GB
1000 $ - Gift Card

Yellow	9	Delete
Purple	10	Delete
Blue	100	Delete

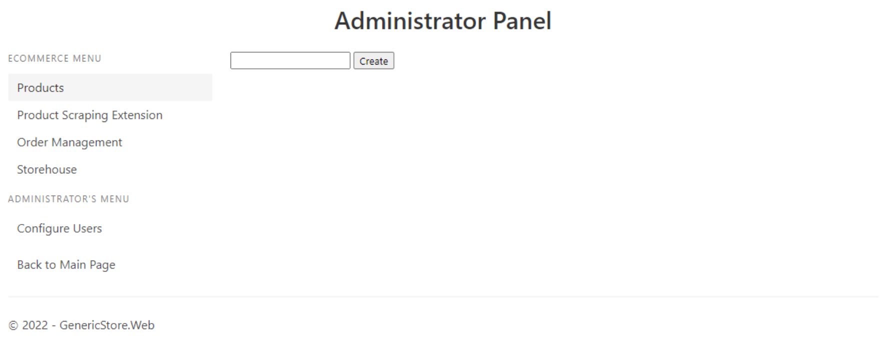
Update Products

New Storehouse
Size / Any extra info
10
Add

© 2022 - GenericStore.Web

Fig. 5.15. Warehouse management page

The second sub-panel, “Administrator’s Menu” is depicted in Fig. 5.10. It consists of a page for creating a new user with the role of a manager, shown in Fig. 5.16. The button for returning to the main application page is displayed in Fig. 5.10.


Administrator Panel

ECOMMERCE MENU
Products
Product Scraping Extension
Order Management
Storehouse
ADMINISTRATOR'S MENU
Configure Users
Back to Main Page

Create

© 2022 - GenericStore.Web

Fig. 5.16. Warehouse management page

Returning to Fig. 5.1, you can observe that some products have icons of yellow and red colors. These icons, in the case of yellow, indicate that the product is running low in stock, with fewer than 11 units remaining.

If the icon color is red, it signifies that the product is out of stock. These icons can be seen in Fig. 5.17.

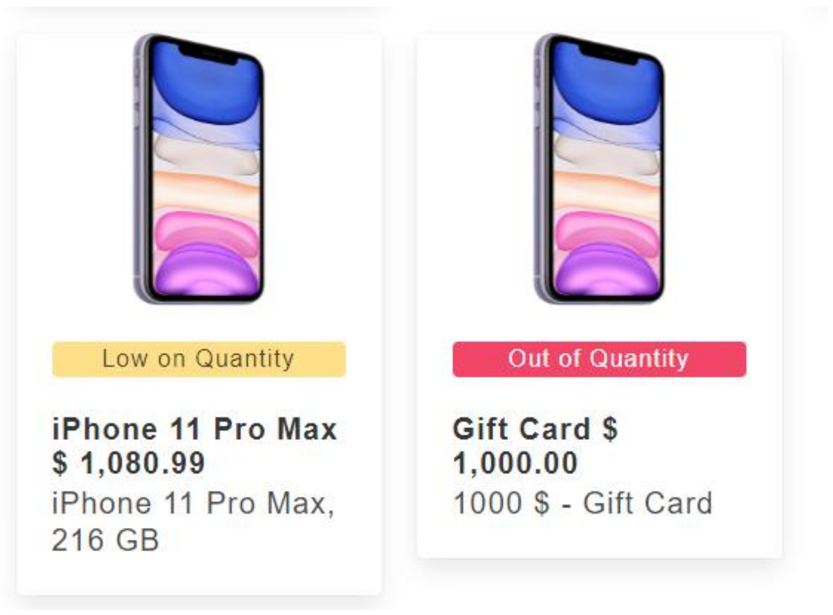


Fig. 5.17. Icons of the status of goods in the warehouse

Additionally, while filling the shopping cart with items, the cart menu generates a dynamic sum of all added products. The dynamic change in the cart's total can be observed in Fig. 5.18.

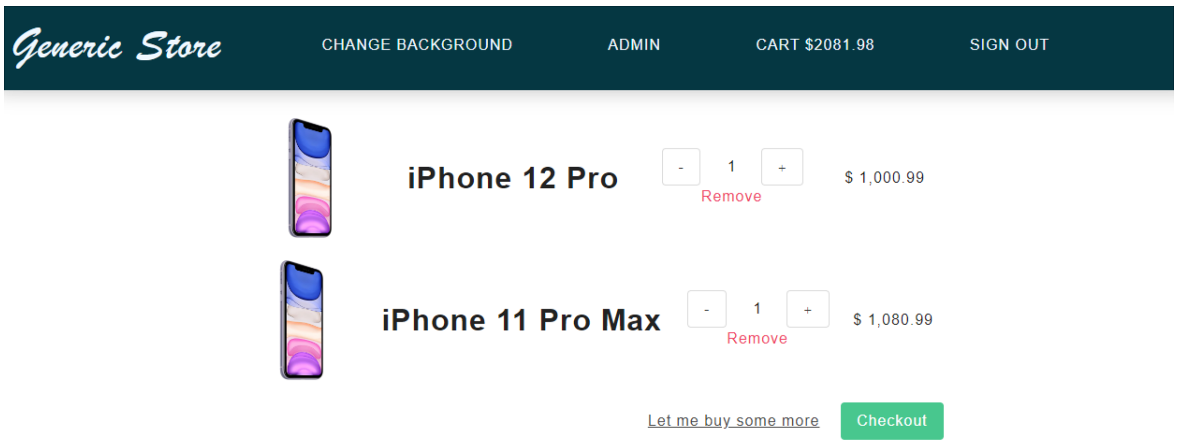


Fig. 5.18. Icons of product status in stock

Conclusions on the section

The user guide provides comprehensive instructions for navigating the dynamic e-commerce application. Users, both clients and administrators should meet system requirements, including a device with internet access. Installation is straightforward, requiring the launch of the provided link.

Upon accessing the application, users encounter a user-friendly interface. For clients, functionalities encompass product viewing, cart management, and order processes. Administrators enjoy additional capabilities, such as product and inventory management.

Visual cues, like colored icons, convey stock statuses efficiently. In summary, the guide systematically explains how to use and manage the e-commerce platform, ensuring a seamless experience for all users.

CONCLUSIONS

During the development of this diploma project, several improvements were considered and implemented for the e-commerce web application using web scraping with an optimized product reading process, addressing various shortcomings in existing implementations and introducing several innovations.

The introduced innovations include the automation of product creation processes through web data scraping, the restructuring of the order processing conveyor framework, and the implementation of a dynamic user interface.

One of the key features of this project is the ability to automatically generate product lists and descriptions obtained from internet stores for the application's administrator. Additionally, a functional order processing framework and dynamic user interface were developed.

In the "Analysis of software requirements" section, a comprehensive analysis of the subject area, software requirements based on successful IT projects, and their significance in this project are presented.

In the "Software modeling and design" section, the architecture of the web application and its database are designed and examined from a software perspective. It also presents class libraries, classes, and methods that define the application's logic.

The "Quality analysis and software testing" section analyzes application quality indicators, describes testing processes, and reviews a test case for the web application.

In the "Software implementation and maintenance" section, the deployment process and usage of the web application are explained.

In the "User guide" section, the step-by-step instructions for navigating and utilizing the dynamic e-commerce application are detailed.

The result of this diploma project is a fully functioning e-commerce web application using web scraping with an optimized product reading process, which includes several innovations that address the shortcomings of existing solutions. This application can be easily expanded and supplemented with new unique features to meet various application needs.

REFERENCES

1. CMS [Electronic resource]. – 2023. – Mode of access to the resource: https://www.techtarget.com/searchcontentmanagement/definition/content-management-system-CMS.
2. E-commerce platform [Electronic resource]. – 2023. – Mode of access to the resource: https://sendpulse.com/support/glossary/ecommerce-platform.
3. Web scraping [Electronic resource]. – 2023. – Mode of access to the resource: https://www.zyte.com/learn/what-is-web-scraping/.
4. Camunda [Electronic resource]. – 2023. – Mode of access to the resource: https://docs.camunda.org/manual/7.17/.
5. CMS statistics [Electronic resource]. – 2023. – Mode of access to the resource: https://www.envisagedigital.co.uk/wordpress-market-share/.
6. WordPress stats [Electronic resource]. – 2023. – Mode of access to the resource: https://kinsta.com/knowledgebase/what-is-wordpress/.
7. WordPress [Electronic resource]. – 2023. – Mode of access to the resource: https://www.namecheap.com/wordpress/what-is-wordpress/.
8. Drupal [Electronic resource]. – 2023. – Mode of access to the resource: https://www.drupal.org/about.
9. Joomla [Electronic resource]. – 2023. – Mode of access to the resource: https://www.techopedia.com/definition/3276/joomla.
10. Magento [Electronic resource]. – 2023. – Mode of access to the resource: https://www.commonplaces.com/blog/what-is-magento/.
11. CMS comparison [Electronic resource]. – 2023. – Mode of access to the resource: https://blog.templatetoaster.com/open-source-cms/.
12. Software development process [Electronic resource]. – 2023. – Mode of access to the resource: https://www.techopedia.com/definition/16431/software-development.
13. SDLC [Electronic resource]. – 2023. – Mode of access to the resource: https://www.indeed.com/career-advice/career-development/what-is-software-development.
14. Programming language C# [Electronic resource]. – 2023. – Mode of access to the

resource: https://www.techopedia.com/definition/26272/c-sharp.

15. ASP.NET Core framework [Electronic resource]. – 2023. – Mode of access to the resource: https://www.tutorialsteacher.com/core/aspnet-core-introduction.
16. MVC template [Electronic resource]. – 2023. – Mode of access to the resource: https://www.tutorialspoint.com/mvc_framework.htm.
17. AWS service [Electronic resource]. – 2023. – Mode of access to the resource: https://www.techopedia.com/definition/26426/amazon-web-services-aws.
18. Docker platform [Electronic resource]. – 2023. – Mode of access to the resource: https://docs.docker.com/get-started/overview/.
19. Stripe service [Electronic resource]. – 2023. – Mode of access to the resource: https://www.nerdwallet.com/article/small-business/what-is-stripe.
20. Bulma framework [Electronic resource]. – 2023. – Mode of access to the resource: https://devmountain.com/blog/why-bulma-css-could-be-your-new-favorite-framework.
21. Vue.js framework [Electronic resource]. – 2023. – Mode of access to the resource: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/.
22. Microsoft SQL Server [Electronic resource]. – 2023. – Mode of access to the resource: https://www.sqlservertutorial.net/getting-started/what-is-sql-server/.
23. HTML Agility Pack library [Electronic resource]. – 2023. – Mode of access to the resource: https://html-agility-pack.net/.
24. Subject area of the database [Electronic resource]. – 2023. – Mode of access to the resource: https://www.orbitanalytics.com/subject-area-modeling/.
25. Concept and description of ER diagram [Electronic resource]. – 2023. – Mode of access to the resource: https://www.smartdraw.com/entity-relationship-diagram/.
26. Database schema [Electronic resource]. – 2023. – Mode of access to the resource: https://www.tutorialspoint.com/dbms/dbms_data_schemas.htm.
27. UML software diagrams [Electronic resource]. – 2023. – Mode of access to the resource: https://www.tutorialspoint.com/uml/uml_standard_diagrams.htm.
28. Use case diagram [Electronic resource]. – 2023. – Mode of access to the resource: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-

use-case-diagram/.

29. Class diagram [Electronic resource]. – 2023. – Mode of access to the resource: https://www.microtool.de/en/knowledge-base/what-is-a-class-diagram/.
30. Sequence diagram [Electronic resource]. – 2023. – Mode of access to the resource: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/.
31. State diagram [Electronic resource]. – 2023. – Mode of access to the resource: https://www.techopedia.com/definition/16446/state-diagram.
32. Component diagram [Electronic resource]. – 2023. – Mode of access to the resource: https://www.lucidchart.com/pages/uml-component-diagram.
33. Software quality [Electronic resource]. – 2023. – Mode of access to the resource: https://asq.org/quality-resources/software-quality.
34. Software testing [Electronic resource]. – 2023. – Mode of access to the resource: https://u-tor.com/topic/software-quality-defined-and-measure.
35. xUnit [Electronic resource]. – 2023. – Mode of access to the resource: https://xunit.net/.
36. Moq [Electronic resource]. – 2023. – Mode of access to the resource: https://docs.microsoft.com/en-us/shows/visual-studio-toolbox/unit-testing-moq-framework.
37. Types of tests [Electronic resource]. – 2023. – Mode of access to the resource: https://www.kaizenko.com/what-is-the-testing-pyramid/.
38. Concept and description of unit testing [Electronic resource]. – 2023. – Mode of access to the resource: https://www.guru99.com/unit-testing-guide.html.
39. Concept and description of integration testing [Electronic resource]. – 2023. – Mode of access to the resource: https://www.guru99.com/integration-testing.html.
40. Concept and description of system testing [Electronic resource]. – 2023. – Mode of access to the resource: https://www.guru99.com/system-testing.html.
41. Concept and description of functional testing [Electronic resource]. – 2023. – Mode of access to the resource: https://www.guru99.com/functional-testing.html.
42. Concept and description of acceptance testing [Electronic resource]. – 2023. –

Mode of access to the resource: https://www.panaya.com/blog/testing/what-is-uat-testing/.

43. Concept and description of UI testing [Electronic resource]. – 2023. – Mode of access to the resource: https://www.guru99.com/gui-testing.html.
44. Concept and detailed description of software deployment methods [Electronic resource]. – 2023. – Mode of access to the resource: https://thenewstack.io/deployment-strategies/.

APPENDIX A

Listing of some parts of the app source code

GenericStore.Entities/ClientInformation.cs

```
using System;

namespace GenericStore.Entities
{
    public class ClientInformation
    {
        public string Name { get; set; }
        public string Surname { get; set; }
        public string Email { get; set; }
        public string PhoneNumber { get; set; }

        public string RequiredAddress { get; set; }
        public string OptionalAddress { get; set; }
        public string City { get; set; }
        public string PostalCode { get; set; }
        public DateTime OrderDate { get; set; }

    }
}
```

GenericStore.Entities/Order.cs

```
using System;
using System.Collections.Generic;
using GenericStore.Entities.Enums;
```

```
namespace GenericStore.Entities
{
    public class Order
    {
        public int Id { get; set; }
        public string OrderReference { get; set; }
        public string StripeTokenReference { get; set; }

        public string Name { get; set; }
        public string Surname { get; set; }
        public string Email { get; set; }
        public string PhoneNumber { get; set; }

        public string RequiredAddress { get; set; }
        public string OptionalAddress { get; set; }

        public string City { get; set; }
        public string PostalCode { get; set; }
        public DateTime OrderDate { get; set; }

        public OrderStatus Status { get; set; }
        public ICollection<OrderStorehouse> OrderStorehouses { get; set; }
    }
}
```

GenericStore.Entities/OrderStorehouse.cs

```
namespace GenericStore.Entities
{
```

```
public class OrderStorehouse
    {
        public int OrderId { get; set; }
        public Order Order { get; set; }

        public int StorehouseId { get; set; }
        public Storehouse Storehouse { get; set; }

        public int Quantity { get; set; }
    }
}
```

GenericStore.Entities/Product.cs

```
using System.Collections.Generic;

namespace GenericStore.Entities
{
    public class Product
    {
        public int Id { get; set; }
        public string Naming { get; set; }
        public string Specification { get; set; }
        public decimal Price { get; set; }

        public ICollection<Storehouse> Storehouses { get; set; }
    }
}
```

GenericStore.Entities/ProductInCart.cs

```
namespace GenericStore.Entities
{
    public class ProductInCart
    {
        public int ProductId { get; set; }
        public string ProductName { get; set; }
        public int StorehouseId { get; set; }
        public decimal Price { get; set; }
        public int Quantity { get; set; }
    }
}
```

GenericStore.Entities/Storehouse.cs

```
using System.Collections.Generic;

namespace GenericStore.Entities
{
    public class Storehouse
    {
        public int Id { get; set; }
        public string Description { get; set; }
        public int Quantity { get; set; }

        public int ProductId { get; set; }
        public Product Product { get; set; }


        public ICollection<OrderStorehouse> OrderStorehouses { get; set; }
    }
}
```

GenericStore.Entities/StorehouseOnHold.cs

```
using System;
namespace GenericStore.Entities
{
    public class StorehouseOnHold
    {
        public int Id { get; set; }

        public string SessionTokenId { get; set; }

        public int StorehouseId { get; set; }
        public Storehouse Storehouse { get; set; }

        public int Quantity { get; set; }
        public DateTime DateOfExpiry { get; set; }
    }
}
```

GenericStore.UnitTests/BLTests/CartServices/AddClientInformationTest.cs

```
using AutoFixture;
using GenericStore.BL.IMPL.CartServices;
using GenericStore.DAL.ABSTRACT;
using GenericStore.Entities;
using GenericStore.Models.CartDTO;
using Moq;
using Xunit;

namespace GenericStore.UnitTests.BLTests.CartServices
{
    public class AddClientInformationTest
    {
        private Mock<ISessionControlManager> mockedRepository;
        private AddClientInformation service;
        private Fixture fixture;

        public AddClientInformationTest()
        {
            mockedRepository = new Mock<ISessionControlManager>();
            service = new AddClientInformation(mockedRepository.Object);
            fixture = new Fixture();
        }

        [Fact]
        public void DoAction_EntityCreated()
        {
            var dto = fixture.Create<ClientInformationDTO>();
            service.DoAction(dto);
            mockedRepository.Verify(x =>
x.AddClientInformation(It.IsAny<ClientInformation>()));
```

```
        }
    }
}
```

GenericStore.Models/OrdersDTO/CreateOrderRequestDTO.cs

```
using System;
using System.Collections.Generic;
namespace GenericStore.Models.OrdersDTO
{
    public class CreateOrderRequestDTO
    {
        public string StripeTokenReference { get; set; }
        public string SessionId { get; set; }

        public string Name { get; set; }
        public string Surname { get; set; }
        public string Email { get; set; }
        public string PhoneNumber { get; set; }

        public string RequiredAddress { get; set; }
        public string OptionalAddress { get; set; }
        public string City { get; set; }
        public string PostalCode { get; set; }
        public DateTime OrderDate { get; set; }

        public List<StorehouseDTO> Storehouses { get; set; }
    }
}
```

GenericStore.Models/OrdersDTO/GetOrderResponseDTO.cs

```
using System;
using System.Collections.Generic;

namespace GenericStore.Models.OrdersDTO
{
    public class GetOrderResponseDTO
    {
        public string OrderReference { get; set; }

        public string Name { get; set; }
        public string Surname { get; set; }
        public string Email { get; set; }
        public string PhoneNumber { get; set; }

        public string RequiredAddress { get; set; }
        public string OptionalAddress { get; set; }
        public string City { get; set; }
        public string PostalCode { get; set; }
        public DateTime OrderDate { get; set; }

        public IEnumerable<ProductDTO> Products { get; set; }

        public string TotalPrice { get; set; }
    }
}
```

GenericStore.Models/OrdersDTO/ProductDTO.cs

```
namespace GenericStore.Models.OrdersDTO
{
   public class ProductDTO
   {
      public string Naming { get; set; }
      public string Specification { get; set; }
      public string Price { get; set; }
      public int Quantity { get; set; }
      public string StorehouseDescription { get; set; }
   }
}
```

GenericStore.Models/OrdersDTO/StorehouseDTO.cs

```
namespace GenericStore.Models.OrdersDTO
{
   public class StorehouseDTO
   {
      public int StorehouseId { get; set; }
      public int Quantity { get; set; }
   }
}
```

APPENDIX B

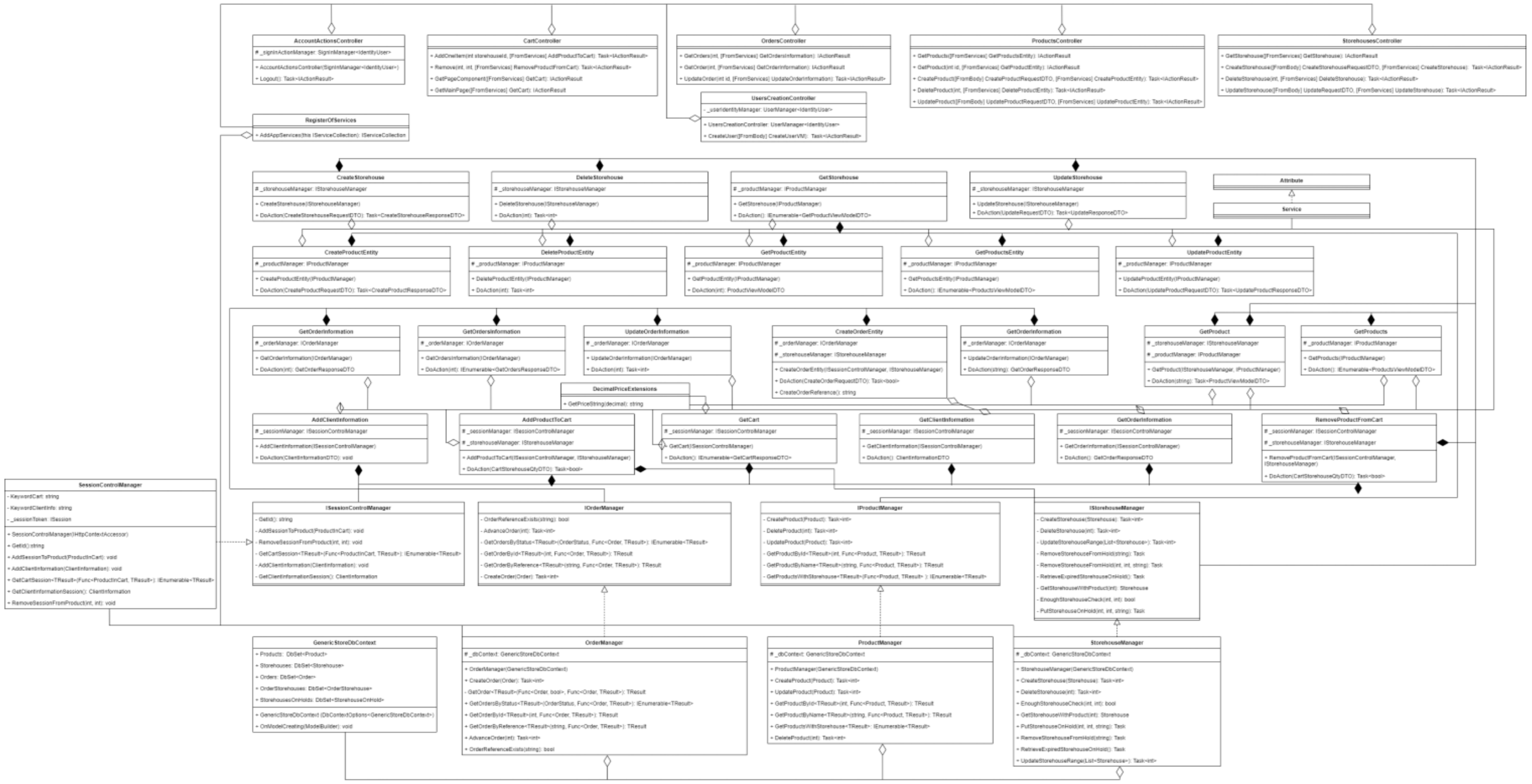


Fig. B.1. Structural scheme of software classes