

**NATIONAL TECHNICAL UNIVERSITY OF UKRAINE
«IGOR SIKORSKY
KYIV POLYTECHNIC INSTITUTE»**

Faculty of Applied Mathematics

Department of Computer Systems Software

«Admission to the defense»

Department Head

_____ Yevgeniya SULEMA

«__» _____ 2023

Diploma project

for a bachelor's degree

**in the educational and professional program «Software Engineering of
Multimedia and Information Retrieval Systems»**

specialty 121 Software Engineering

on the topic: "Web-application for e-commerce on car Accessories"

Performer:

4th year student, group KP-94

Mohamed Islam KHELILI

Islam Khelili

Supervisor:

Associate Professor of DCSS, Ph.D., Assoc. Prof.

Dmytro NOVAK



Consultant on normative control:

Associate Professor of DCSS, Ph.D., Assoc. Prof.

Mykola ONAI

Reviewer:

Associate Professor of NUHT, Ph.D., Assoc. Prof.

Andriy MOSHENSKY

I certify that in this diploma project there are
no borrowings from the works of other
author's references.

Student *Islam Khelili*

Kyiv – 2023

National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute"
Faculty of Applied Mathematics
Department of Computer Systems Software

Level of higher education – first (bachelor's)

Specialty – 121 "Software Engineering"

Educational and professional program "Software Engineering of Multimedia and Information Retrieval Systems "

APPROVED

Department Head

_____ Yevgeniya SULEMA

" ____ " _____ 2023

TASK
for the student's diploma project

Mohamed Islam Khelili

1. The topic of the project "Web-application for e-commerce on car Accessories", the supervisor Dmytro Novak, Ph.D., associate professor, approved by the order of the university dated " ____ " _____ 20 ____ №. _____
2. The deadline for the student to submit the project " ____ " is June 2023.
3. Initial data for the project: see Technical task.
4. Contents of the explanatory note:
 - analysis of programming languages and website development technologies;
 - development of the web resource of the electronic commerce platform;
 - description of developed algorithms and routines;
 - analysis of the developed web resource.
5. List of mandatory graphic material:
 - user authorization (drawing);
 - data synchronization (drawings);
 - web resource architecture (poster);
 - structure of web application (poster);

6. Consultants of project sections

Section	Surname, initials and position of the consultant	Signature, date	
		issued the task	accepted the task
Normative control	Onai M.V., associate professor		

7. The date of issue of the task is October 31, 2022.

Calendar plan

N° z/p	The name of the stages of the diploma project	The term of execution of the stages of the project	Note
1.	Study of literature on the subject of the project	13.11.2022	
2.	Development and coordination of the technical task	25.11.2022	
3.	Development of the web resource structure	15.12.2022	
4.	Preparation of materials for the first section of the diploma project	30.12.2022	
5.	Development of page design and graphic elements	02.02.2023	
6.	Preparation of materials for the second section of the diploma project	19.02.2023	
7.	Software implementation of the web resource	10.03.2023	
8.	Web resource testing	17.03.2023	
9.	Preparation of materials for the third section of the diploma project	30.03.2023	
10.	Preparation of materials for the fourth section of the diploma project	12.04.2023	
11.	Preparation of the graphic part of the diploma project	21.04.2023	
12.	Completion of diploma project documentation	26.05.2023	

Student

Mohamed Islam KHELILI

Supervisor

Dmytro NOVAK

ABSTRACT

The final year project focuses on crafting a web application tailored for an e-commerce platform specializing in car accessories. In today's digitally-driven era, e-commerce stands out as a pivotal avenue for streamlined business transactions. With a meticulous blend of the LAMP stack and front-end technologies such as Blade, Bootstrap, and FontAwesome, coupled with the dynamic functionality of Tablescoop, our aim is to create an intuitive interface that enriches the online shopping experience for users and empowers business owners with efficient platform management tools.

Our endeavor encompasses extensive research, analysis, and the implementation of pivotal features including product catalog management, robust user authentication mechanisms, seamless shopping cart functionality, integrated payment gateways, and streamlined order processing systems. The web application boasts a responsive design ensuring optimal viewing experiences across diverse devices ranging from desktops to smartphones.

Adhering to industry best practices, we have adopted modern technologies and frameworks throughout the development lifecycle. For the frontend, HTML, CSS, and JavaScript were meticulously integrated while Python and SQLite power the backend operations. Furthermore, stringent security protocols have been implemented to safeguard user data and fortify online transactions against potential threats.

By delivering an immersive web application tailored for the e-commerce landscape, our project aims to elevate the shopping journey for customers while equipping business owners with a potent tool to expand their digital footprint and drive sales. The successful execution of this initiative promises to catalyze advancements in electronic commerce, bridging the divide between businesses and consumers in the digital realm.

Annotation

The diploma project focuses on creating a web application tailored for an e-commerce platform specializing in car accessories, utilizing the LAMP stack. In today's digital era, e-commerce has become increasingly popular as a convenient and efficient method for conducting business transactions. The objective of the web application is to provide a user-friendly interface that ensures a seamless online shopping experience for customers while facilitating effective platform management for business owners.

This project entails extensive research, analysis, and implementation of key features such as product catalog management, secure user authentication, shopping cart functionality, payment integration, and order processing. The web application is designed with a responsive layout to ensure compatibility across various devices, allowing users to effortlessly access the platform from desktops, laptops, tablets, and smartphones.

Following industry-standard methodologies, modern technologies and frameworks inherent in the LAMP stack are utilized throughout the development process. For the fronted, Blade, Bootstrap, and Font Awesome enhance the user interface, while backed operations are powered by PHP and MySQL. Additionally, stringent security measures are implemented to protect user's sensitive information and ensure secure online transactions.

The web application tailored for the e-commerce platform specializing in car accessories holds immense potential to elevate the shopping experience for consumers. By offering business owners a robust tool to expand their online presence and drive sales, this project aims to contribute significantly to the advancement of electronic commerce. It bridges the gap between businesses and consumers in the digital marketplace, empowering both parties with efficient and secure online transaction capabilities.

Faculty of Applied Mathematics
Department of Computer Systems Software

"APPROVED"

Department Head

_____ Yevgeniya SULEMA

"__" _____ 2023

WEB-APPLICATION FOR E-commerce on car Accessories

Technical task

DP.045440-02-91

"AGREED"

Supervisor:

_____ Dmytro NOVAK

Normative control:

_____ Mykola ONAI

Performer:

_____ Mohamed Islam KHELILI

2023

CONTENT

1. Name and field of application	3
2. Basis for development	3
3. Purpose of development	3
4. Requirements for the software product	3
5. Requirements for project documentation	4
6. Design stages	5
7. The order of development testing	5

1. NAME AND FIELD OF APPLICATION

Development name: Web-application for e-commerce on car Accessories.

Field of application: information technologies.

2. BASIS FOR DEVELOPMENT

The basis for the development is a diploma design task approved by the Department of Computer Systems Software of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute".

3. PURPOSE OF THE DEVELOPMENT

The development is intended for use as a web-application for e-commerce on car Accessories platform. It serves as a digital marketplace where customers can browse, search, and purchase products online.

4. SOFTWARE PRODUCT REQUIREMENTS

1. Operating System: The web-application should be compatible with various operating systems such as Windows, Linux, and macOS.
2. Web Server: The web-application should be hosted on a reliable web server with sufficient storage, bandwidth, and processing power.
3. Database: A robust database management system should be used to store and manage product, customer, and order data.
4. Payment Gateway Integration: The web-application should be integrated with a secure payment gateway to enable online transactions.
5. Content Management System: A user-friendly content management system should be used to manage and update the website's content, products, and promotions.

6. Security: The web-application should implement robust security measures such as SSL certificates, firewalls, and regular backups to protect customer data and prevent cyber-attacks.
7. Analytics: The web-application should integrate with analytics tools to track user behavior, traffic, and sales data to inform marketing and sales strategies.
8. Performance Optimization: The web application should be optimized for fast loading speed, smooth user experience, and mobile responsiveness.
9. SEO: The web application should be optimized for search engines to improve visibility and attract organic traffic.
10. Integrations: The web application should integrate with third-party applications such as shipping, inventory, and email marketing tools to streamline operations and enhance customer experience

5. REQUIREMENTS FOR PROJECT DOCUMENTATION

The following documentation would be developed in the course of project documentation:

- 1) explanatory note;
- 2) testing program and methodology;
- 3) user manual;
- 4) drawings:
 - "Database structure";
 - "Entity-relationship diagram".

6. DESIGNING STAGES

Study of literature on the topic of the work	16.11.2022
Development and coordination of the technical task	27.11.2022
Development of the structure of the web resource	15.12.2022
Development of page design and graphic elements	02.03.2023
Software implementation of the web resource	15.03.2023
Testing of the web resource	07.03.2023
Preparation of materials for the text part of the project	28.04.2023
Preparation of materials for the graphic part of the project	12.05.2023
Issuance of technical documentation of the project	25.05.2023

7. THE ORDER OF DEVELOPMENT TESTING

Testing of the developed software product is performed in accordance with the "Testing program and methodology".

**Faculty of Applied Mathematics
Department of Computer Systems Software**

"APPROVED"

Department Head

_____ Yevgeniya SULEMA

" ____ " _____ 2023

WEB-APPLICATION FOR E-commerce on car Accessories

Explanatory note

DP.045440-03-81

"AGREED"

Supervisor:

_____ Dmytro NOVAK

Normative control:

_____ Mykola ONAI

Performer:

_____ Mohamed Islam KHELILI

2023

CONTENT

LIST OF TERMS AND ABBREVIATIONS	3
INTRODUCTION	4
1. ANALYZING SOFTWARE REQUIREMENTS.....	5
1.1. Analysis of the subject area.....	5
1.2. Formulation of the problem.....	5
1.3. Analysis of the market and existing solution approaches.....	5
2. SOFTWARE MODELING AND DESIGN SECURITY.....	20
2.1. Theoretical information.....	20
2.2. Design pattern.	26
2.2. Description of the technologies used in development.	26
2.2.1. Overview of technologies	30
2.2.2. Overview of development utilities.....	20
2.3. Setting up the development environment.	26
2.3.1. Setting up the environment for service development	30
2.4. Compilation and description of the algorithm.....	20
2.5. Development of architecture.....	20
2.5.1. Overview of web application methods.....	20
2.5.2. Database development with Entity Framework Code First	26
2.5.3. Description of database table structures	30
2.6. Conclusions to the section.	49
3. ANALYSIS AND TESTING OF SOFTWARE QUALITY.....	32
3.1. Results of the test login to the system and the purchase process.....	32
3.2. Testing user registration.....	36
3.2. Testing the functionality of the administrator.....	36

3.3. Conclusions of chapter 3.....	39
CONCLUSIONS	61
LIST OF REFERENCES	62
APPENDICES	64

LIST OF TERMS AND ABBREVIATIONS

LAMP stack - An acronym representing the combination of Linux, Apache, MySQL, and PHP, used for web server infrastructure.

1. **Blade** - A templating engine used for the frontend in Laravel, enabling efficient code reuse.
2. **Bootstrap** - A popular CSS framework for building responsive and mobile-first websites.
3. **FontAwesome** - A comprehensive icon set and toolkit for web development.
4. **PHP** - A server-side scripting language used for web development.
5. **MySQL** - An open-source relational database management system.
6. **HTML** - Hypertext Markup Language, used for structuring web pages.
7. **CSS** - Cascading Style Sheets, used for styling web pages.
8. **JavaScript** - A programming language used for enhancing interactivity and functionality on web pages.
9. **SQLite** - A lightweight and serverless relational database management system.
10. **eCommerce** - Electronic commerce, the buying and selling of goods and services over the internet.
11. **UI** - User Interface, the visual elements and layout of a website or application.
12. **UX** - User Experience, the overall experience of a user when interacting with a website or application.
13. **API** - Application Programming Interface, a set of rules and protocols for building and interacting with software applications.
14. **SSL** - Secure Sockets Layer, a security protocol that ensures encrypted communication between a web server and a browser.
15. **CSRF** - Cross-Site Request Forgery, a type of attack that forces a user to execute unwanted actions on a web application in which they are authenticated.
16. **MVC** - Model-View-Controller, a software architectural pattern used for organizing code in web applications.
17. **CRUD** - Create, Read, Update, Delete, the basic operations of persistent storage in a database.
18. **RESTful** - Representational State Transfer, an architectural style for designing networked applications, commonly used for web services.
19. **AJAX** - Asynchronous JavaScript and XML, a technique for updating parts of a web page without reloading the entire page.

INTRODUCTION

E-commerce is rapidly gaining popularity as a widely accepted business model, with an increasing number of companies adopting online platforms for conducting business transactions. The focus of this project is to develop a versatile Car Accessories online store that allows customers to conveniently shop for various items. The availability of detailed product information and easy accessibility through retailer websites make online shopping an appealing and commonplace activity for consumers. Today's customers are attracted to the convenience, extensive options, competitive pricing, and user-friendly search functionality offered by online shopping platforms.

Additionally, business owners can benefit from reduced overhead costs associated with physical stores, enabling them to provide cost-effective purchasing options. The global reach of online purchasing expands the customer base, allowing businesses to cater to diverse backgrounds and enhancing overall customer value. An online store serves as a virtual platform where customers can browse through inventory, add desired items to a shopping cart, and proceed to checkout. During the checkout process, customers are prompted to provide necessary information such as billing and shipping addresses, delivery preferences, and payment details like credit card numbers or payment gateways.

1.ANALYZING SOFTWARE REQUIREMENTS

As we delve into the development of the e-commerce platform for car accessories, it's crucial to meticulously analyze the software requirements to guarantee that the application aligns with the needs of both customers and business owners. Let's break down the key aspects we need to consider:

1. **User Requirements**: We must grasp the needs and expectations of our users, encompassing both customers and business owners. This entails identifying features such as seamless navigation, efficient product search capabilities, secure payment processing, order tracking functionalities, and user-friendly account management.
2. **Functional Requirements**: These entail the specific features and functionalities that the software must deliver. This encompasses defining the system's behavior, including tasks such as displaying product listings, enabling users to add items to their cart, facilitating smooth checkout processes, and generating comprehensive order confirmations.
3. **Non-Functional Requirements**: These refer to the qualities that the software must possess, such as performance, reliability, security, and usability. For instance, our application should load swiftly, manage concurrent user traffic effectively, safeguard user data through robust encryption methods, and offer an intuitive user interface.
4. **System Requirements**: This involves delineating the hardware, software, and network infrastructure necessary to support the application. We need to specify server specifications, database requirements, and any third-party services or APIs that we'll integrate into our system.
5. **Regulatory Requirements**: Depending on the geographical location of our users and the nature of our business, we may need to adhere to legal and regulatory standards. This could entail compliance with data protection laws, online payment regulations, and accessibility guidelines.
6. **Scalability and Flexibility**: Our application should be primed for future growth and adaptable to changing requirements. This includes scalability to accommodate heightened traffic and transactions, as well as flexibility to incorporate new features and adapt to evolving business demands.
7. **Documentation**: Clear and comprehensive documentation is paramount for stakeholders to comprehend the functionality and utilization of the software. This encompasses user manuals, technical documentation for developers, and system architecture diagrams.

By meticulously analyzing these software requirements, we can ensure that our e-commerce platform for car accessories effectively meets user needs, complies with regulatory standards, and provides a reliable and user-friendly shopping experience.

1.1 Analysis of the subject area

Enhancing the Digital Marketplace for Car Accessories:

In the vibrant landscape of e-commerce, the niche of car accessories stands out as a dynamic and ever-evolving sector. The subject area encompasses a diverse array of products, ranging from practical necessities like floor mats and seat covers to high-tech gadgets such as dash cams and GPS trackers. As car enthusiasts

increasingly turn to online platforms for their automotive needs, there arises a significant opportunity to enhance the digital marketplace for car accessories.

One of the key drivers of growth in this sector is the growing affinity towards personalization and customization among car owners. With the rise of social media and influencer culture, individuals are keen on expressing their unique identities through their vehicles, fueling demand for customized accessories and aftermarket modifications. This trend underscores the importance of offering a wide selection of products and personalized shopping experiences to cater to diverse tastes and preferences.

Furthermore, the advent of new technologies, such as electric vehicles and autonomous driving systems, is reshaping the landscape of automotive accessories. As car manufacturers introduce innovative features and functionalities, there arises a need for compatible accessories and add-ons to complement these advancements. Consequently, there is a burgeoning market for cutting-edge accessories designed to enhance the performance, safety, and convenience of modern vehicles.

In addition to product innovation, the subject area also encompasses challenges related to logistics, supply chain management, and customer service. Ensuring timely delivery, managing inventory effectively, and providing exceptional customer support are critical factors that contribute to the success of e-commerce platforms in this domain. Moreover, with the increasing emphasis on sustainability and environmental consciousness, there is a growing demand for eco-friendly and ethically sourced products within the car accessories market.

As we delve deeper into the subject area, it becomes evident that the digital marketplace for car accessories presents both opportunities and challenges. By leveraging innovative technologies, adopting a customer-centric approach, and fostering partnerships with suppliers and manufacturers, e-commerce platforms can unlock the full potential of this dynamic sector. Through strategic analysis and targeted interventions, we aim to contribute to the evolution of the digital marketplace for car accessories, creating value for both consumers and businesses alike.

It's essential to recognize the significant role of consumer behavior and market trends within the subject area of car accessories e-commerce. Understanding the preferences, purchasing habits, and emerging trends of target audiences is crucial for designing effective marketing strategies and product offerings.

Consumer preferences in the realm of car accessories are often influenced by factors such as vehicle type, brand loyalty, lifestyle choices, and cultural influences. For instance, enthusiasts of off-road vehicles may gravitate towards rugged and durable accessories suited for outdoor adventures, while urban commuters may seek practical and space-saving solutions for their daily commute.

Moreover, market trends such as the rise of electric vehicles (EVs), the growing popularity of car-sharing services, and the increasing focus on sustainability are reshaping the landscape of car accessories. With the shift towards electric mobility, there is a demand for accessories tailored to EVs, such as charging solutions, range extenders, and interior enhancements designed to optimize energy efficiency.

Additionally, the emergence of smart technologies and connected car ecosystems presents new opportunities for innovation within the car accessories market. Consumers are increasingly seeking accessories equipped with smart features such as Bluetooth connectivity, voice control, and integration with mobile apps for enhanced convenience and functionality.

Furthermore, the subject area encompasses challenges related to market saturation, competition from brick-and-mortar retailers, and the proliferation of counterfeit products. E-commerce platforms must differentiate themselves through value-added services, product quality assurance, and brand integrity to build trust and credibility among consumers.

In light of these insights, our analysis underscores the importance of adopting a holistic approach to e-commerce within the car accessories sector. By staying abreast of market dynamics, leveraging consumer insights, and embracing technological advancements, we can position ourselves as leaders in the digital marketplace for car accessories. Through strategic partnerships, innovative product offerings, and exceptional customer experiences, we aim to drive growth and innovation within this dynamic and evolving industry.

1.2 Formulation of the problem

In the realm of e-commerce for car accessories, several key challenges and opportunities present themselves, necessitating a clear formulation of the problem statement. The formulation of the problem encompasses identifying the core issues or gaps in the current landscape and defining the objectives to be addressed.

Here's a structured formulation of the problem:

1. **Market Fragmentation**: The car accessories market is characterized by a multitude of sellers, platforms, and product offerings, leading to fragmentation and lack of centralized resources for consumers. This fragmentation poses challenges for consumers in finding reliable sources for quality products and trustworthy information.
2. **Product Authenticity and Quality Assurance**: With the proliferation of online marketplaces, ensuring the authenticity and quality of car accessories becomes a paramount concern. Consumers face challenges in discerning genuine products from counterfeit ones, leading to trust issues and potential dissatisfaction.
3. **Customer Experience and Personalization**: The evolving expectations of consumers demand personalized shopping experiences and seamless customer service. However, many e-commerce platforms in the car accessories sector struggle to deliver tailored experiences and efficient customer support, impacting customer satisfaction and loyalty.
4. **Logistics and Supply Chain Management**: Efficient logistics and supply chain management are critical for timely delivery and inventory management. Challenges such as inventory stockouts, long lead times, and shipping delays can hinder the smooth functioning of e-commerce operations and affect customer satisfaction.
5. **Technological Integration and Innovation**: The rapid pace of technological advancements presents both opportunities and challenges for e-commerce platforms. Integrating innovative technologies such as augmented reality (AR), artificial intelligence (AI), and predictive analytics can enhance the shopping experience and drive sales. However, the adoption of these technologies requires investment, expertise, and strategic planning.
6. **Competitive Landscape and Differentiation**: The car accessories market is highly competitive, with numerous players vying for consumer attention. E-commerce platforms must differentiate

themselves through unique value propositions, brand positioning, and innovative offerings to stand out in a crowded marketplace.

Objectives: Based on the formulation of the problem, the objectives of our project are defined as follows:

1. To create a centralized and user-friendly e-commerce platform for car accessories, addressing the market fragmentation and providing consumers with a reliable source for authentic products.
2. To implement robust quality assurance measures and authentication processes to ensure the authenticity and quality of products offered on the platform, thereby building trust and credibility among consumers.
3. To enhance the customer experience by implementing personalized shopping features, efficient customer support systems, and seamless order fulfillment processes.
4. To optimize logistics and supply chain management processes to ensure timely delivery, minimize stockouts, and improve overall operational efficiency.
5. To explore and integrate innovative technologies that enhance the shopping experience, such as AR-enabled product visualization, AI-powered product recommendations, and predictive analytics for inventory management.
6. To develop a unique value proposition and brand identity that differentiates our e-commerce platform from competitors and resonates with target consumers, thereby driving customer acquisition and retention.

By formulating the problem statement and defining clear objectives, we lay the foundation for addressing the challenges and seizing the opportunities within the e-commerce landscape for car accessories. Through strategic planning, execution, and continuous improvement, we aim to deliver a compelling and competitive e-commerce solution that meets the needs and expectations of our target audience.

1.3 Analysis of the market and existing solution approaches

In the highly dynamic market of e-commerce for car accessories, a thorough analysis of the current landscape and existing solution approaches is essential for identifying opportunities and challenges. Here's a comprehensive analysis:

1. **Market Overview:** The market for car accessories e-commerce is witnessing steady growth, fueled by factors such as the increasing adoption of online shopping, rising vehicle ownership rates, and growing consumer demand for customization and personalization. However, the market is also highly fragmented, with numerous players ranging from specialized niche retailers to general-purpose online marketplaces.
2. **Competitive Landscape:** The competitive landscape is characterized by a mix of established players and emerging startups, each vying for market share through differentiating factors such as product assortment, pricing strategy, customer service, and technological innovation. Key players include both dedicated car accessories retailers and automotive aftermarket divisions of major retailers.

3. Existing Solution Approaches :

- a. **Dedicated E-commerce Platforms** : Several dedicated e-commerce platforms specialize in car accessories, offering a wide range of products catering to diverse consumer preferences. These platforms typically focus on providing a curated selection of high-quality products, efficient logistics, and responsive customer service.
- b. **Marketplace Aggregators** : Some aggregator platforms consolidate listings from multiple sellers, offering consumers a broader selection of products while providing sellers with a centralized platform to reach a larger audience. These platforms often emphasize competitive pricing and user reviews to facilitate informed purchasing decisions.
- c. **Brick-and-Mortar Retailers with Online Presence** : Many traditional brick-and-mortar retailers in the automotive sector have expanded their reach by establishing an online presence. These retailers leverage their existing brand recognition and physical store network to attract customers while offering the convenience of online shopping and home delivery.
- d. **Customization and Personalization Services** : A growing trend in the market is the provision of customization and personalization services, where consumers can tailor their car accessories to match their individual preferences and vehicle specifications. These services may include custom-fit products, bespoke designs, and DIY installation kits.

4. Challenges and Opportunities :

- a. **Product Authenticity and Quality** : Ensuring the authenticity and quality of products remains a challenge, particularly with the proliferation of counterfeit goods in the market. Platforms that prioritize rigorous quality control measures and partner with reputable suppliers can gain a competitive edge.
- b. **Logistics and Fulfillment** : Efficient logistics and fulfillment are critical for providing a seamless shopping experience. Platforms that invest in robust logistics infrastructure, optimize supply chain management, and offer fast and reliable shipping options can enhance customer satisfaction.
- c. **Customer Experience** : Providing a personalized and user-friendly shopping experience is paramount for building customer loyalty. Platforms that leverage data analytics, AI-driven recommendations, and responsive customer support channels can differentiate themselves in the market.
- d. **Technological Innovation** : Embracing technological innovations such as AR/VR product visualization, AI-powered chatbots, and predictive analytics can enhance the shopping experience and drive sales. Platforms that stay ahead of the curve in adopting emerging technologies can gain a competitive advantage.

In conclusion, the market for car accessories e-commerce presents a mix of challenges and opportunities for players in the industry. By analyzing the market landscape and existing solution approaches, stakeholders can identify key areas for innovation, differentiation, and strategic investment to capitalize on emerging trends and meet evolving consumer demands.

2. SOFTWARE MODELING AND DESIGN SECURITY

In the development of e-commerce platforms for car accessories, software modeling and design play a crucial role in ensuring both functionality and security. Let's delve into the software tools utilized in our project and their advantages, along with how they contribute to enhancing security:

1. **Laravel Framework**: Laravel is a powerful PHP web framework known for its elegant syntax and robust features. It follows the Model-View-Controller (MVC) architectural pattern, facilitating the separation of concerns and promoting code organization. Laravel offers built-in security features such as CSRF (Cross-Site Request Forgery) protection, SQL injection prevention, and secure authentication mechanisms. By leveraging Laravel's authentication scaffolding and middleware, developers can implement user authentication and authorization with ease, mitigating the risk of unauthorized access to sensitive data.
2. **HTML and CSS**: HTML (Hypertext Markup Language) provides the structure of web pages, while CSS (Cascading Style Sheets) enhances the visual presentation and styling. When designing the frontend interface for our e-commerce platform, adherence to best practices in HTML and CSS ensures a user-friendly and intuitive shopping experience. Additionally, implementing secure coding practices such as input validation and output encoding helps prevent common vulnerabilities like cross-site scripting (XSS) and injection attacks.
3. **JavaScript**: JavaScript is a versatile scripting language used for client-side interactivity and dynamic content. In our project, JavaScript enhances the frontend functionality by enabling features such as interactive product carousels, form validation, and asynchronous data loading. However, it's crucial to implement JavaScript securely to prevent client-side attacks like DOM (Document Object Model) manipulation and unauthorized data access. Utilizing frameworks like Vue.js or React with built-in security features can help mitigate such risks.
4. **Python and Django**: While not directly utilized in the frontend development, Python and the Django framework play a crucial role in the backend architecture of our e-commerce platform. Django's built-in security features, including protection against common web vulnerabilities such as CSRF attacks, SQL injection, and clickjacking, provide a robust foundation for secure backend development. Python's readability and extensive standard library further contribute to writing secure and maintainable backend code.
5. **Database Management Systems (e.g., MySQL, PostgreSQL)**: Secure database management is essential for safeguarding sensitive user information and transaction data. Utilizing robust database management systems like MySQL or PostgreSQL with features such as encryption, role-based access control, and data validation helps ensure data integrity and confidentiality. Implementing parameterized queries and stored procedures also mitigates the risk of SQL injection attacks.
6. **Security Testing and Auditing**: In addition to secure software design and development practices, regular security testing and auditing are imperative to identify and remediate potential vulnerabilities. Conducting thorough penetration testing, vulnerability scanning, and code reviews help uncover security flaws early in the development lifecycle, allowing for timely mitigation measures to be implemented. Leveraging automated security testing tools and following industry

best practices, such as the OWASP (Open Web Application Security Project) Top 10, further strengthens the security posture of our e-commerce platform.

7. **SSL/TLS Encryption**: Implementing SSL/TLS encryption is crucial for securing data transmission between clients and servers. By using HTTPS protocol and SSL/TLS certificates, sensitive information such as login credentials, payment details, and personal data is encrypted during transit, preventing interception by malicious actors. Leveraging Laravel's built-in support for SSL/TLS and configuring web servers to enforce HTTPS ensures secure communication channels, bolstering overall data security.
7. **Content Security Policy (CSP)**: Content Security Policy is a security standard that helps mitigate cross-site scripting (XSS) attacks by specifying trusted sources of content and restricting the execution of inline scripts and unsafe practices. Integrating CSP headers into our web application's HTTP responses helps prevent malicious scripts from executing in the browser, thereby reducing the risk of XSS vulnerabilities. Leveraging Laravel's middleware functionality, developers can easily implement CSP policies and enforce strict content security measures.
8. **Two-Factor Authentication (2FA)**: Two-factor authentication adds an extra layer of security by requiring users to verify their identity using two different factors, typically something they know (e.g., password) and something they possess (e.g., mobile device). Integrating 2FA into our e-commerce platform enhances user authentication security, reducing the risk of unauthorized account access in the event of compromised credentials. Laravel provides convenient packages and libraries for implementing 2FA functionality, enabling developers to enhance user account security with minimal effort.
9. **Security Headers and Best Practices**: Adhering to security best practices and implementing security headers in HTTP responses helps mitigate various web application security risks. Headers such as X-Content-Type-Options, X-Frame-Options, and Referrer-Policy help prevent common vulnerabilities such as MIME sniffing, clickjacking, and information leakage. By configuring web servers and utilizing Laravel middleware to set appropriate security headers, our e-commerce platform can bolster its defense against potential security threats and enhance overall resilience.
10. **Session Management and CSRF Protection**: Secure session management is essential for preventing session hijacking and unauthorized access to user accounts. Implementing robust session handling mechanisms and integrating CSRF protection measures help mitigate the risk of CSRF attacks, where attackers exploit user sessions to perform unauthorized actions on behalf of authenticated users. Laravel's built-in session management features and CSRF protection middleware simplify the implementation of secure session handling and CSRF prevention, ensuring the integrity and confidentiality of user sessions.

By incorporating these advanced security measures into the software modeling and design of our e-commerce platform for car accessories, we prioritize the protection of user data, safeguard against potential threats, and uphold the highest standards of security and privacy. Through continuous vigilance, proactive security measures, and adherence to industry best practices, we aim to build a secure and trustworthy

platform that instills confidence in our users and fosters long-term success in the competitive e-commerce landscape.

In summary, software modeling and design in our e-commerce project for car accessories encompass a comprehensive approach to security across frontend and backend components. By leveraging secure coding practices, robust frameworks, and diligent security testing, we aim to build a resilient and secure platform that instills trust and confidence among users while safeguarding their sensitive information.

2.1 Theoretical information

1. **Threat Modeling:** Threat modeling is a systematic approach to identifying and mitigating potential security threats and vulnerabilities in software systems. It involves analyzing the system's architecture, identifying potential attack vectors, and assessing the impact of security breaches. By conducting threat modeling exercises early in the software development lifecycle, developers can proactively address security risks and design robust countermeasures to mitigate them.
2. **Secure Software Development Lifecycle (SDLC):** The Secure Software Development Lifecycle is a framework that integrates security practices into every phase of the software development process. It encompasses activities such as security requirements analysis, secure design and architecture, secure coding practices, security testing, and ongoing security maintenance. By adopting a Secure SDLC approach, organizations can systematically address security concerns and produce more resilient and secure software products.
3. **Security by Design Principles:** Security by Design is a design philosophy that emphasizes integrating security considerations into the architecture, design, and implementation of software systems from the outset. Key principles of Security by Design include least privilege, defense in depth, fail-safe defaults, and separation of duties. By adhering to these principles, developers can build software systems that are inherently secure and resistant to a wide range of cyber threats.
4. **Threat Modeling Techniques:** Various threat modeling techniques are used to systematically identify, prioritize, and mitigate security threats in software systems. Common techniques include STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege), DREAD (Damage, Reproducibility, Exploitability, Affected Users, Discoverability), and Attack Trees. These techniques help developers understand potential threats, assess their impact, and devise appropriate countermeasures to mitigate them effectively.
5. **Secure Coding Practices:** Secure coding practices are coding guidelines and best practices aimed at preventing common security vulnerabilities and weaknesses in software code. Examples of secure coding practices include input validation, output encoding, parameterized queries, proper error handling, and secure authentication mechanisms. By following secure coding practices, developers can minimize the risk of introducing security vulnerabilities into software applications and improve overall code quality and robustness.
6. **Security Testing Techniques:** Security testing encompasses a variety of techniques and methodologies aimed at identifying security vulnerabilities and weaknesses in software systems.

Common security testing techniques include penetration testing, vulnerability scanning, code reviews, static and dynamic analysis, and fuzz testing. By conducting comprehensive security testing throughout the software development lifecycle, organizations can identify and remediate security vulnerabilities before they can be exploited by attackers.

7. **Compliance and Regulatory Standards:** Compliance with industry standards and regulatory requirements is essential for ensuring the security and privacy of software systems, especially in highly regulated industries such as finance, healthcare, and government. Common standards and regulations include PCI DSS (Payment Card Industry Data Security Standard), HIPAA (Health Insurance Portability and Accountability Act), GDPR (General Data Protection Regulation), and ISO/IEC 27001 (Information Security Management System). Compliance with these standards helps organizations demonstrate their commitment to security and protect sensitive customer data from unauthorized access and disclosure.

By incorporating theoretical knowledge of software modeling and design security principles, techniques, and best practices into the development process, organizations can build software systems that are more resilient, secure, and trustworthy, thereby minimizing the risk of security breaches and protecting valuable assets and sensitive information from cyber threats.

2.1.1 Design Pattern Considerations with Laravel and Sanctum for Authentication:

1. **Singleton Pattern:** I can utilize the Singleton pattern to ensure that my authentication service remains consistent throughout my application. This pattern will ensure that there's only one instance of the authentication service, simplifying management and reducing unnecessary overhead.
2. **Factory Method Pattern:** I can apply the Factory Method pattern to create different types of authentication tokens or sessions based on user requirements. This pattern allows me to encapsulate the logic for creating JWT tokens or OAuth tokens dynamically, depending on the client's authentication method.
3. **Decorator Pattern:** I'll consider using the Decorator pattern to add additional functionalities or security checks to my authentication process. By creating decorator classes, I can easily enhance the authentication service with features like multi-factor authentication (MFA) or rate-limiting without modifying the core authentication logic.
4. **Repository Pattern:** I can implement the Repository pattern to abstract data access operations related to user authentication and authorization. This pattern allows me to switch between different data storage mechanisms seamlessly without impacting the authentication logic. For instance, I'll create UserRepository interfaces with methods like `findByUsername()` or `findByEmail()` for retrieving user data.
5. **Strategy Pattern:** The Strategy pattern will help me encapsulate different authentication strategies and switch between them dynamically based on user preferences or system requirements. By

defining authentication strategy interfaces and selecting the appropriate strategy during runtime, I can support various authentication methods such as password-based or token-based authentication.

6. **Observer Pattern**: I can leverage the Observer pattern to implement event-driven authentication mechanisms. By defining authentication event observers, I can trigger actions like logging, auditing, or sending notifications whenever a user logs in, logs out, or updates their authentication credentials.

By incorporating these design patterns alongside Laravel's authentication features and Sanctum for API token authentication, I can design a flexible, scalable, and secure authentication system that meets my application's requirements while adhering to best practices in software design and architecture.

2.2 Description of the technologies used in development

2.2.1 *HTML*

The proliferation of the internet ushered in a new era in terms of how people gain access to and engage with various forms of information. This, in turn, led to the development of the World Wide Web and, subsequently, websites. HTML, which stands for "Hypertext Markup Language," is a foundational language that supports the structure and content of web pages. HTML is at the center of this digital revolution. HTML is the most important component of front-end development because it offers a standardized method for organizing information and presenting it in a way that is both aesthetically pleasing and interactive.

HTML makes use of a system known as tagging, in which individual elements are specified by tags that provide information regarding their purpose and function. These tags, which are denoted by the angle brackets (>), encompass both the opening tags and the closing tags, so denoting the beginning and the end of an element. The content of a webpage is placed inside of these tags, which enables web developers to arrange and format text, graphics, links, forms, and other parts of a webpage.

HTML's capacity to generate a hierarchical structure, which can organise content in a way that is both logical and intelligible, is one of its most significant qualities. It is possible to nest elements within one another, so generating a parent-child relationship between the elements. The content may be arranged more easily with the help of this hierarchical structure, which also helps to ensure that the information is presented in a way that is consistent and easy to understand [14].

HTML elements can have attributes, which can be used to offer additional information about those elements. Developers are able to modify the functionality, appearance, and behavior of elements by specifying attributes within the opening tags of those components. Attributes are located within the opening tags of elements. A name and a value are the two components that make up an attribute.

The name identifies the particular attribute, while the value supplies the information or setting that corresponds to the name.

Over the course of its existence, HTML has developed to embrace a variety of new capabilities as well as standards. The most recent version of HTML, HTML5, has incorporated semantic elements, which are features that contain meaning and improve both the accessibility and search engine optimization of online pages. Developers are able to provide a clearer understanding of the structure and purpose of content by making use of semantic elements like `header`, `nav`, `section`, `article`, and `footer`. This is to the benefit of users as well as search engines.

Even though HTML is the building block of a webpage, it frequently works in conjunction with other markup languages like CSS (Cascading Style Sheets) and JavaScript to improve the aesthetics and functionality of webpages. The use of cascading style sheets, or CSS, to apply styles to HTML elements, such as colors, fonts, and layouts, enables customization while maintaining uniformity across different web pages. On the other side, JavaScript provides developers the ability to include interactivity and dynamic behavior into web pages, thereby producing compelling user experiences [15].

HTML is an essential language in the process of developing websites because it lays the groundwork and framework for creating websites that are both interactive and visually engaging. Developers are able to successfully organize content and present it in a manner that is friendly to users if they have a solid understanding of the fundamental ideas, components, and properties of HTML. This section's goal is to present a thorough introduction to HTML, focusing on the language's significance in front-end development and its indispensable part in sculpting the experience of using the modern web.

```
<!DOCTYPE html>
<html>

  <head>
    <title> Title here </title>
  </head>

  <body>
    Web page content goes here.
  </body>

</html>
```

Fig. 2.0. Structure of HTML

2.2.2 CSS

The use of cascading style sheets, also known as CSS, is an essential part of contemporary web development. These sheets play a vital part in improving both the aesthetic look and the layout of websites. CSS stands for "cascading style sheets," and it is a language that is used in conjunction with HTML to manage the appearance of web content. This includes the colors, fonts, spacing, and placement of the information. A website's developers are able to produce designs that are aesthetically engaging and consistent when they employ CSS. These designs can improve the user experience and reflect the website's brand identity.

The purpose of this part is to provide a comprehensive examination of CSS and the role that it plays in front-end development. We are going to go into the fundamental ideas, capabilities, and uses of CSS in order to provide a full understanding of its purpose in determining the aesthetics and functionality of web sites.

The cascading style sheet (CSS) is based on a rule-based system, which means that rules may be developed to target particular HTML elements and apply various styling features. It is possible to identify the items that need to be styled by making use of selectors such as class, ID, and element selectors. The look of these components can be modified, along with their layout, through the use of a variety of stylistic characteristics, including as color, font size, margin, padding, and background. Developers can do this.

The capability of CSS to divide a web page's display layer from its content layer is one of the most significant benefits offered by this markup language. This separation enables a clean and organized code structure, which in turn makes it much simpler to manage and update styles across a number of different pages. The developers of a website are able to create consistency throughout the entire site and make global modifications in an effective manner if the stylistic instructions are centralized in a separate CSS file. Cascading is another feature that is introduced by CSS. This allows many styles to be applied to a single element, and any conflicts are resolved according to the element's specificity and the sequence in which the styles were declared. This cascading behavior gives developers the ability to construct designs that are flexible and responsive, allowing them to adjust to a variety of screen sizes and devices. The usage of media queries, which is a component of CSS, enables the smooth and enhanced optimization of the surfing experience by enabling the customization of styles based on the features of the user's device [16].

CSS includes a variety of advanced layout capabilities that extend beyond fundamental placement. Both CSS Grid and Flexbox are highly effective layout frameworks that provide web developers with the ability to design intricate and interactive page architectures. These layout tools provide exact control over the placement and alignment of elements, which enables designers to create designs that are complex and responsive. Preprocessors for CSS, like Sass and Less, provide additional

functionalities and features that are similar to programming in order to improve the development process. They do this by introducing notions such as variables, mixins, and functions, which enables developers to design CSS code that is reusable and modular, hence increasing productivity and making the code easier to maintain.

Cascading Style Sheets (CSS) is a language that is crucial to the frontend development process since it gives developers the ability to control the visual appearance and layout of web content. Developers are able to construct websites that are aesthetically pleasing and friendly to users if they have a solid understanding of the fundamental concepts, selectors, and layout approaches that CSS offers. This section will attempt to provide a comprehensive examination of CSS, focusing on its relevance to the process of developing websites as well as its power to alter both the appearance and the functionality of online pages.

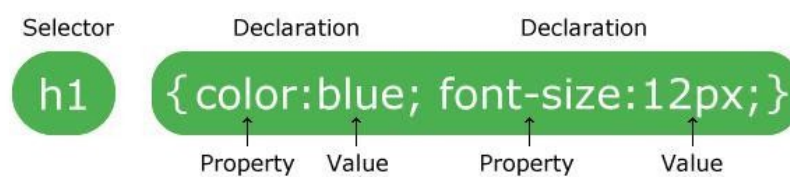


Fig. 2.1. CSS syntax

2.2.3 JavaScript

JavaScript is a programming language that is used to create dynamic web pages and add interactivity to web applications. It is a popular and powerful language that can be executed within a web page in a browser. Many big companies like Netflix, Walmart, and PayPal have built entire applications using JavaScript. With JavaScript, you can create real-time networking applications like chats and video streaming services, command-line tools, and games. Additionally, JavaScript allows you to create complex web applications that can perform tasks like sending HTTP requests and manipulating HTML and CSS code.

Many big companies like Netflix, Walmart, and PayPal have built entire applications using JavaScript. This demonstrates the power and flexibility of the language. JavaScript's popularity can be attributed to its ability to run within a web page in a browser, making it accessible to anyone with an internet connection. Furthermore, JavaScript is easy to learn, making it a great language for beginners to start with.

JavaScript was first introduced in 1995 as LiveScript by Netscape. The name was later changed to JavaScript due to the popularity of Java. Despite the name change, JavaScript has nothing in common with Java and is an independent language. Initially, JavaScript was used for spamming, annoying overlays, and pop-ups. However, it has since evolved to become a more robust language with a wide range of applications.

JavaScript is an interpreted language, meaning it is compiled at runtime rather than during development. It is designed to run in a browser, and every browser has a JavaScript engine that can

execute JavaScript code. Google's JavaScript engine, known as V8, was extracted to run JavaScript anywhere, and many non-browser environments, such as Node.js and Adobe Acrobat, also use JavaScript.

One of the main advantages of JavaScript is its ability to manipulate HTML code, CSS, and send background HTTP requests. This means that you can use JavaScript to create web pages that can respond to user interactions without reloading the entire page. To use JavaScript in HTML, you need to put your code between the `<script>` and `</script>` tags. This allows the browser to identify the JavaScript code and execute it accordingly.

In summary, JavaScript is a powerful and versatile language that has revolutionized web development. It is used to create dynamic web pages, add interactivity to web applications, and perform complex tasks like sending HTTP requests and manipulating HTML and CSS code. Despite its humble beginnings, JavaScript has become one of the most widely used programming languages in the world, and its importance is only set to increase in the future.

2.2.4 Bootstrap

Bootstrap is a front-end development framework that has become very popular among developers and designers. It's flexible, easy to use, and lets you build fully responsive websites quickly. With Bootstrap, you don't need to spend hours coding your own grid because it comes with a responsive grid system. You can easily add a range of components like navigation bars, dropdowns, progress bars, and thumbnails to your web pages. Bootstrap's website explains each piece of code in detail, making it easy for developers to understand how it works and how to use it effectively.

Bootstrap can be used with any IDE or editor, and with any server-side technology and language from ASP.NET to PHP to Ruby on Rails. Its popularity is due to the fact that it has a large community of designers and developers behind it who contribute to its development and make it even more effective. Bootstrap is hosted on GitHub, making it easy for developers to modify and contribute to its codebase.

One of the main benefits of Bootstrap is its ability to resize images automatically based on the current screen size. Additionally, there are many custom jQuery plugins available in Bootstrap. However, one of the main criticisms of frameworks like Bootstrap is their size and the amount of code they use, which can slow down the application upon first load. To solve this issue, developers have started to create templates based on Bootstrap to speed up the web development

2.2.5 PHP

PHP is a widely utilized programming language renowned for its effectiveness in backend web development. It serves as a reliable and flexible tool for developers aiming to construct robust online applications. Unlike Python, PHP boasts a long-standing history in web development, making it a staple choice for creating dynamic and scalable web solutions.

One of the key strengths of PHP lies in its simplicity and ease of use, which facilitates rapid development cycles. With PHP, developers can swiftly create backend logic for web applications without sacrificing functionality. Additionally, PHP offers a vast array of built-in functions and features tailored specifically

for web development tasks, such as handling HTTP requests, processing form data, and interacting with databases.

Moreover, PHP's extensive ecosystem of frameworks and libraries further enhances its capabilities for backend development. Frameworks like Laravel, Symfony, and CodeIgniter provide developers with powerful abstractions and tools for building complex web applications. These frameworks streamline common development tasks, including database integration, authentication, and session management, thereby accelerating the development process.

PHP's seamless integration with popular database systems, such as MySQL and PostgreSQL, makes it an ideal choice for managing dynamic content and data-driven applications. Developers can leverage PHP's robust database connectivity features to efficiently store, retrieve, and manipulate data, ensuring optimal performance and reliability.

Furthermore, PHP's compatibility with various web servers, including Apache and Nginx, ensures smooth deployment and operation across different hosting environments. This cross-platform interoperability enables developers to deploy PHP applications on a wide range of operating systems, simplifying the deployment process and maximizing compatibility.

Additionally, PHP benefits from a large and active community of developers, who contribute to its ongoing development and support. This vibrant community fosters collaboration, knowledge sharing, and continuous improvement within the PHP ecosystem, ensuring that developers have access to the latest tools, resources, and best practices.

In summary, PHP remains a highly effective and versatile programming language for backend web development. Its simplicity, extensive feature set, rich ecosystem of frameworks and libraries, and strong community support make it a preferred choice for building scalable, efficient, and maintainable web applications. Developers can harness PHP's power to create sophisticated backend systems that drive dynamic and data-driven websites with ease.

2.2.6 Laravel

Laravel is a high-level PHP web framework renowned for its extensive suite of tools and functionalities tailored to expedite website development in a streamlined and efficient manner. As an open-source web application framework, Laravel was crafted to adhere to the Model-View-Controller (MVC) architectural pattern, offering developers a structured approach to building scalable and maintainable online applications.

One of Laravel's standout advantages lies in its emphasis on productivity and code reusability, echoing the "Don't Repeat Yourself" (DRY) principle. This philosophy encourages developers to create reusable components and minimize redundant code, fostering efficiency and maintainability throughout the development process. Laravel automates common web development tasks by furnishing developers with a comprehensive set of built-in features and conventions, enabling them to focus on crafting application-specific logic rather than mundane development chores.

At the core of Laravel's features lies its robust Object-Relational Mapping (ORM) layer, which empowers developers to interact with database tables using intuitive Python objects. The ORM provides a pragmatic approach to defining database models and executing database operations without the need for composing

raw SQL queries. Compatible with a diverse range of database backends, including MySQL, PostgreSQL, and SQLite, Laravel's ORM ensures seamless database integration and data management.

Laravel's URL routing system stands as another notable feature, mapping URLs to corresponding views and allowing developers to define URL patterns and associate them with view functions or classes. This decoupling of URLs and views enhances flexibility and maintainability, facilitating easy modification and extension of application routes.

Moreover, Laravel boasts a powerful templating engine that empowers developers to craft dynamic and reusable HTML templates. The templating engine facilitates template inheritance, enabling the creation of basic templates that share elements with others, thereby reducing code duplication. Additionally, it offers control structures, filters, and template tags for manipulating and displaying data within templates.

The built-in authentication and authorization system in Laravel simplifies the implementation of secure user authentication for online applications, offering user management, registration, and login features out of the box. With fine-grained access control capabilities, developers can restrict user actions based on predefined roles and permissions, ensuring robust security measures are in place.

Laravel's administration interface streamlines content management tasks, providing administrators with a user-friendly interface to monitor and interact with application data without requiring additional code. Additionally, Laravel supports internationalization and localization, enabling developers to construct applications tailored to a global audience by facilitating text translation and localization of dates and numbers based on various locales.

Thanks to Laravel's extensibility, developers can seamlessly integrate third-party libraries and create reusable Laravel applications, referred to as "Laravel packages." The Laravel Package Index serves as a repository for a plethora of Laravel packages offering diverse functionality that can be effortlessly incorporated into projects.

In conclusion, Laravel is a feature-rich web framework written in PHP that empowers developers to build reliable, scalable, and maintainable online applications. Its focus on efficiency, code reusability, and adherence to best practices simplifies the website development process, while its built-in capabilities, including ORM, URL routing, templating, authentication, and administration, equip developers with the tools needed to create sophisticated web applications. Laravel's extensibility and vibrant community support contribute to its ongoing success and appeal in the realm of web development.

2.2.1 Overview of technologies

in my e-commerce project for car accessories, I've chosen a carefully curated stack of technologies to ensure efficiency, scalability, and security.

For the backend development, I've opted for the Laravel framework, a robust PHP framework known for its elegant syntax and comprehensive features. Laravel provides me with a solid foundation for building the backend logic of my e-commerce platform, including user authentication, product management, and order processing. With Laravel, I can leverage features like routing, middleware, and ORM to streamline development and maintain clean, organized code.

To handle authentication and API token management, I've integrated Laravel Sanctum. Sanctum provides a simple and secure way to authenticate users and issue API tokens, essential for securing access to my e-commerce platform's resources. By using Sanctum, I can implement features like user registration, login, and token-based authentication with ease, ensuring that my platform remains secure while offering a seamless user experience.

On the frontend side, I've utilized HTML, CSS, and JavaScript, along with the Blade templating engine provided by Laravel. These technologies allow me to create dynamic and interactive user interfaces for my e-commerce platform. With HTML providing the structure, CSS handling the styling, and JavaScript adding interactivity, I can deliver a modern and user-friendly shopping experience to my customers.

Additionally, I've incorporated Bootstrap and Font Awesome to expedite frontend development and enhance the visual appeal of my platform. Bootstrap provides a responsive grid system and pre-styled components, making it easier to create responsive layouts that adapt to different screen sizes. Meanwhile, Font Awesome offers a vast library of icons that I can use to enhance the visual elements of my platform, improving usability and aesthetics.

In summary, my choice of technologies for this e-commerce project reflects a balance between efficiency, security, and user experience. By leveraging the strengths of Laravel, Sanctum, HTML, CSS, and JavaScript, along with supplementary tools like Bootstrap and Font Awesome, I aim to deliver a high-quality e-commerce platform for car accessories that meets the needs of both customers and business owners.

2.2.2 Overview of development utilities.

In my e-commerce project for car accessories, I rely on a range of development utilities to streamline the development process, enhance productivity, and ensure code quality.

1. **Integrated Development Environment (IDE)**: For writing and managing code, I use a powerful IDE like Visual Studio Code or PhpStorm. These IDEs offer features such as syntax highlighting, code completion, debugging tools, and version control integration, helping me write clean, efficient code and navigate through my project with ease.
2. **Version Control System (VCS)**: I utilize Git as my version control system to track changes to my codebase and collaborate with team members effectively. By using Git, I can commit changes, create branches for feature development, merge code changes, and revert to previous versions if needed, ensuring code integrity and facilitating collaborative development.
3. **Package Managers**: Package managers like Composer for PHP and NPM (Node Package Manager) for JavaScript streamline the process of managing dependencies in my project. With Composer, I can easily install and update PHP packages required for my Laravel project, while NPM allows me to manage JavaScript libraries and tools used in frontend development.
4. **Database Management Tools**: To interact with and manage the database, I use database management tools such as phpMyAdmin or MySQL Workbench for MySQL databases and pgAdmin for PostgreSQL databases. These tools provide graphical interfaces for executing SQL queries, managing database schemas, and monitoring database performance, simplifying database administration tasks.
5. **API Development and Testing Tools**: Since my e-commerce platform includes an API for handling requests from frontend clients, I use tools like Postman or Insomnia for API development and testing. These tools allow me to design, document, test, and debug API endpoints efficiently, ensuring API functionality and reliability before integrating them into the frontend or exposing them to external clients.
6. **Continuous Integration/Continuous Deployment (CI/CD) Pipeline**: I set up a CI/CD pipeline using services like GitHub Actions or GitLab CI/CD to automate the process of building, testing, and deploying my application. With CI/CD, I can automatically trigger builds, run automated tests, and deploy updates to staging or production environments, accelerating the development cycle and ensuring code quality and reliability.
7. **Task Runners and Build Tools**: Task runners like Laravel Mix or Webpack, along with build tools like Gulp or Grunt, help optimize frontend assets, such as stylesheets, JavaScript files, and images. These tools automate tasks like compilation, minification, and bundling, optimizing frontend performance and reducing page load times for a smoother user experience.

8. **Error and Log Monitoring Tools**: For monitoring application errors and logging, I integrate tools like Sentry or Bugsnag. These tools capture and report errors in real-time, providing insights into application performance, identifying bugs, and enabling rapid debugging and issue resolution, enhancing the stability and reliability of my e-commerce platform.
9. **Error and Log Monitoring Tools**: For monitoring application errors and logging, I integrate tools like Sentry or Bugsnag. These tools capture and report errors in real-time, providing insights into application performance, identifying bugs, and enabling rapid debugging and issue resolution, enhancing the stability and reliability of my e-commerce platform.
8. **Code Quality and Static Analysis Tools**: To maintain code quality and adhere to coding standards, I utilize static analysis tools such as PHP CodeSniffer and ESLint for PHP and JavaScript respectively. These tools analyze code for syntax errors, coding style violations, and potential bugs, ensuring consistency and readability across the codebase. Additionally, I integrate tools like PHPStan and Psalm for PHP codebase to perform static code analysis and identify type-related issues, ensuring robustness and reliability in my PHP code.
9. **Documentation Tools**: Documentation is essential for maintaining codebase comprehensibility and facilitating collaboration among team members. I use documentation tools like PHPDocumentor for PHP and JSDoc for JavaScript to generate API documentation directly from code comments. These tools automate the generation of comprehensive documentation, including function signatures, parameters, return types, and usage examples, making it easier for developers to understand and use the APIs and ensuring consistency in documentation across the project.
10. **Performance Monitoring and Optimization Tools**: Monitoring application performance is critical for identifying bottlenecks and optimizing resource utilization. I leverage performance monitoring tools like New Relic or Blackfire to profile application performance, identify slow database queries, memory leaks, and CPU-intensive operations. These tools provide actionable insights and recommendations for optimizing performance, ensuring that my e-commerce platform delivers optimal speed and responsiveness to users.
11. **Collaboration and Communication Tools**: Effective collaboration and communication are key to successful project development. I use collaboration tools like Slack, Microsoft Teams, or Discord for real-time communication and collaboration among team members. Additionally, project management tools like Trello, Jira, or Asana help organize tasks, track progress, and facilitate project planning and coordination, ensuring transparency and accountability throughout the development lifecycle.
12. **Security Scanning and Vulnerability Assessment Tools**: As security is paramount for my e-commerce platform, I employ security scanning and vulnerability assessment tools like OWASP ZAP and Snyk to identify and remediate security vulnerabilities in my codebase and dependencies. These tools perform automated security scans, static and dynamic analysis, and dependency checks to detect vulnerabilities such as SQL injection, cross-site scripting (XSS), and outdated dependencies, enabling me to proactively mitigate security risks and strengthen the overall security posture of my application.

By incorporating these development utilities into my workflow, I ensure a streamlined and efficient development process, maintain high code quality and reliability, enhance application performance and security, and foster effective collaboration among team members, ultimately delivering a robust, scalable, and secure e-commerce platform for car accessories. By leveraging these development utilities effectively, I ensure a streamlined and efficient development process, maintain code quality and reliability, and deliver a high-quality e-commerce platform for car accessories that meets the needs of both users and stakeholders.

2.3 Setting up the development environment.

Installing Laravel and Blade: I begin by installing Laravel, the PHP framework that forms the backbone of AutoDelovi, along with its templating engine, Blade. Using Composer, I initiate the Laravel project and integrate Blade for dynamic templating:

```
bashCopy code
composer create-project --prefer-dist laravel/laravel AutoDelovi
cd AutoDelovi
```

1. **Initializing Git Repository:** Version control is crucial for managing code changes and facilitating collaboration. I initialize a Git repository for AutoDelovi and commit the initial codebase:

```
bashCopy code
git init
git add .
git commit -m "Initial commit"
```

2. **Integrating Bootstrap for Styling:** To enhance the visual aesthetics and responsiveness of AutoDelovi, I integrate Bootstrap, the popular CSS framework, using npm:

```
bashCopy code
npm install bootstrap
```

3. **Incorporating Font Awesome for Icons:** Font Awesome enriches AutoDelovi's user interface with a vast collection of icons. I install Font Awesome using npm to seamlessly integrate it into the project:

```
bashCopy code
npm install @fortawesome/fontawesome-free
```

4. **Setting Up Database Configuration:** A crucial aspect of AutoDelovi's development is configuring the database. I update the .env file with the database connection details, including the database type, host, port, name, username, and password:

```
makefileCopy code
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=autodelovi_db
DB_USERNAME=root
DB_PASSWORD=
```

This ensures seamless interaction between AutoDelovi and the database management system, allowing for efficient storage and retrieval of data.

5. **Creating Migration Files**: Laravel's migration feature facilitates the creation and management of database tables within the application. I generate migration files for AutoDelovi's database tables using Artisan, Laravel's command-line interface:

bashCopy code

```
php artisan make:migration create_products_table
```

```
php artisan make:migration create_categories_table
```

These migration files define the structure of the database tables required for storing product and category information, enabling smooth data management.

6. **Running Migrations**: With migration files created, I execute the migrations to create the corresponding database tables:

bashCopy code

```
php artisan migrate
```

This command executes the migration files and creates the necessary tables in the database, ensuring data integrity and consistency within AutoDelovi.

7. **Seeding Database**: To populate the database with initial data for testing and development purposes, I create seeders to insert sample data into the database tables. I generate seeders for products and categories using Artisan:

bashCopy code

```
php artisan make:seeder ProductSeeder
```

```
php artisan make:seeder CategorySeeder
```

These seeders define the sample data to be inserted into the products and categories tables, simulating realistic scenarios for testing AutoDelovi's functionality.

By completing these steps, I establish a comprehensive development environment for AutoDelovi, encompassing Laravel, Blade, Git version control, Bootstrap, and Font Awesome. With the database configured, migrations executed, and seeders created, AutoDelovi is primed for efficient development and robust testing, paving the way for the creation of a feature-rich and user-friendly e-commerce platform for car accessories. By following these steps and integrating Laravel, Blade, Git version control, Bootstrap, and Font Awesome into the development environment, I ensure that AutoDelovi is equipped with a solid foundation for efficient development and a visually captivating user experience.

2.3.1 Setting up the environment for service development

1. **Environment Configuration**: Firstly, I ensure that my development environment is properly configured. I create a `.env` file in the root directory of my project to store environment-specific configuration variables such as database credentials, API keys, and environment settings. Here's an example of how I set up my `.env` file:

makefileCopy code

```
APP_ENV=development
```

```
APP_DEBUG=true
APP_KEY=your_app_key
```

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=your_database_name
DB_USERNAME=your_database_username
DB_PASSWORD=your_database_password
```

```
API_KEY=your_api_key
```

2. **Composer for PHP Dependencies**: I use Composer, a dependency manager for PHP, to manage dependencies and autoload classes in my Laravel project. To initialize a new Laravel project and install dependencies, I run the following commands:

bashCopy code

```
composer create-project --prefer-dist laravel/laravel my-project
cd my-project
composer install
```

This command creates a new Laravel project named `my-project` and installs all required dependencies specified in the `composer.json` file.

3. **Package.json for Frontend Dependencies**: For managing frontend dependencies, I use npm (Node Package Manager) and a `package.json` file. I initialize a `package.json` file by running `npm init` and then install required packages using `npm install` or `npm ci` for clean installations.

bashCopy code

```
npm init -y
npm install bootstrap jquery popper.js
```

This command initializes a `package.json` file with default settings and installs Bootstrap, jQuery, and Popper.js as frontend dependencies. These packages are commonly used for styling and interactive features in web applications.

4. **Running Development Server**: Once dependencies are installed, I start the development server to test my application locally. In Laravel, I use the `artisan` command to start the server:

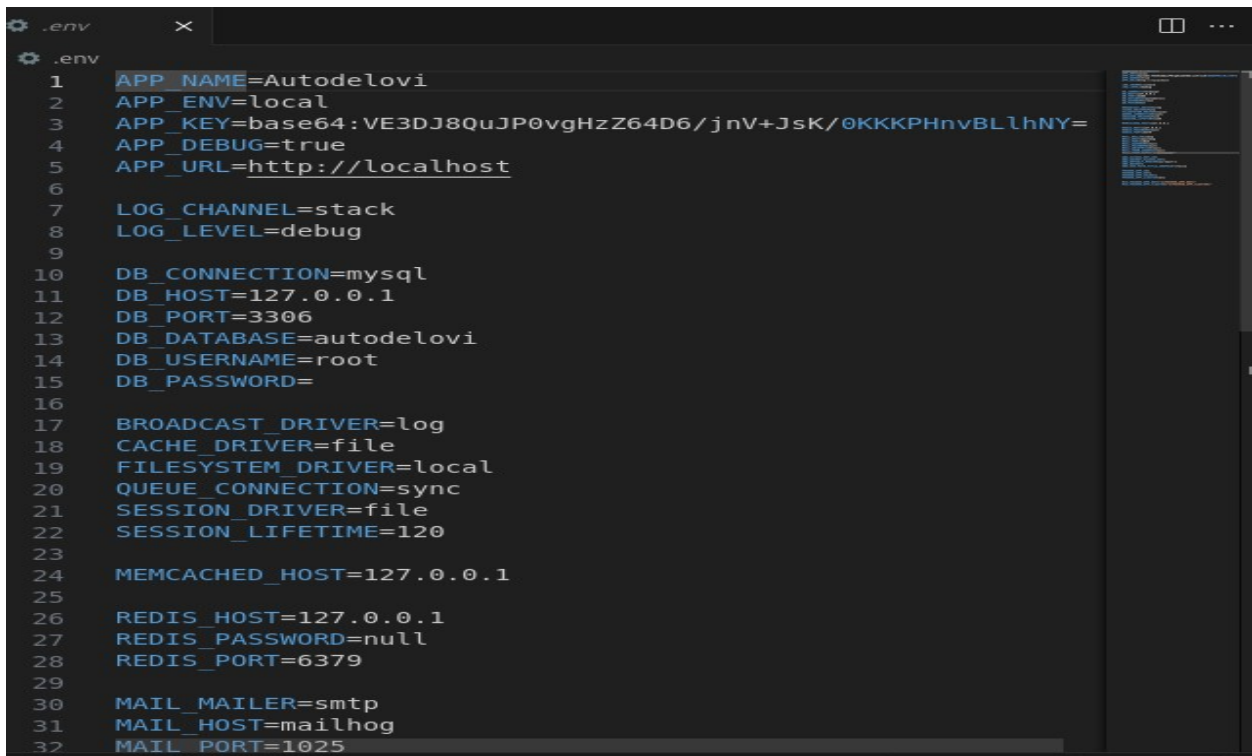
bashCopy code

```
php artisan serve
```

This command starts a development server at `http://localhost:8000`, allowing me to access my Laravel application in a web browser and test its functionality.

By setting up the environment in this manner, I ensure that my development environment is properly configured, dependencies are managed efficiently using Composer and npm, and I can run a development

server to test my application locally before deploying it to production. This streamlined setup process accelerates development and ensures consistency across development environments.

A screenshot of a code editor showing a .env file. The file contains various environment variables for a Laravel application, including app name, environment, key, debug mode, URL, logging settings, database connection details, cache, filesystem, queue, session, memcached, redis, and mail settings. The variables are listed on numbered lines from 1 to 32.

```
1 APP_NAME=Autodelovi
2 APP_ENV=local
3 APP_KEY=base64:VE3DJ8QuJP0vgHzZ64D6/jnV+JsK/0KKKPHnvBLlhNY=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8 LOG_LEVEL=debug
9
10 DB_CONNECTION=mysql
11 DB_HOST=127.0.0.1
12 DB_PORT=3306
13 DB_DATABASE=autodelovi
14 DB_USERNAME=root
15 DB_PASSWORD=
16
17 BROADCAST_DRIVER=log
18 CACHE_DRIVER=file
19 FILESYSTEM_DRIVER=local
20 QUEUE_CONNECTION=sync
21 SESSION_DRIVER=file
22 SESSION_LIFETIME=120
23
24 MEMCACHED_HOST=127.0.0.1
25
26 REDIS_HOST=127.0.0.1
27 REDIS_PASSWORD=null
28 REDIS_PORT=6379
29
30 MAIL_MAILER=smtp
31 MAIL_HOST=mailhog
32 MAIL_PORT=1025
```

Fig. 2.2 . image of .env file

2.4 .Compilation and description of the algorithm

In the context of implementing compilation and description algorithms within a Laravel-based application, it's important to understand the role of Laravel's built-in features and how they influence the development process.

1. **Utilizing Laravel's Features**: Laravel is a powerful PHP framework that comes equipped with a variety of built-in features and utilities designed to streamline the development process. These features include robust routing, expressive ORM (Object-Relational Mapping) for database interaction, powerful templating engine (Blade), authentication, and more. Leveraging these features allows developers to focus on implementing business logic rather than reinventing the wheel.
2. **Choosing Laravel for Compilation Implementation**: When considering the implementation of compilation algorithms within a Laravel application, the framework's flexibility, extensibility, and built-in utilities make it an attractive choice. Laravel's MVC (Model-View-Controller) architecture provides a structured approach to organizing code, making it easier to separate concerns and maintainability. Additionally, Laravel's support for Composer, a PHP dependency manager, simplifies the integration of third-party libraries or packages that may be required for compilation algorithms.
3. **Implementation Strategy**: To implement compilation and description algorithms within a Laravel application, I would leverage Laravel's MVC architecture to organize the code into logical components. The compilation logic would reside within the appropriate controllers or service

classes, where the source code is processed, analyzed, and transformed according to the desired algorithm. Laravel's Eloquent ORM could be utilized to interact with the database, storing and retrieving compiled code or algorithm descriptions as needed.

4. **Integration with Blade Templating Engine**: Laravel's Blade templating engine offers a convenient way to generate dynamic HTML content, making it well-suited for displaying compilation results or algorithm descriptions within views. Blade's intuitive syntax allows for seamless integration of dynamic data into HTML templates, enhancing the presentation of compilation-related information to users.
5. **Choosing Laravel for In-House Development**: The decision to choose Laravel for in-house development of compilation algorithms stems from its robust feature set, active community support, and ease of use. Laravel's comprehensive documentation, along with its vibrant ecosystem of packages and extensions, simplifies the development process and accelerates time-to-market. Moreover, Laravel's built-in testing utilities facilitate unit and integration testing of compilation algorithms, ensuring reliability and correctness.

In summary, leveraging Laravel's built-in features and MVC architecture provides a solid foundation for implementing compilation and description algorithms within a web application. By harnessing Laravel's capabilities and integrating them with custom compilation logic, developers can create efficient, maintainable, and user-friendly solutions for processing and analyzing source code.

2.5 Development of architecture

In the development of architecture for the AutoDelovi e-commerce platform, several key considerations are taken into account to ensure scalability, maintainability, and performance. Let's outline the architecture development process:

1. **Identifying Requirements**: The first step in architecture development is to gather and analyze requirements for the e-commerce platform. This involves understanding the functional and non-functional requirements, such as user authentication, product management, shopping cart functionality, payment processing, scalability, security, and performance.
2. **Selecting Technology Stack**: Based on the identified requirements and project goals, the technology stack is chosen. Since AutoDelovi is built using the Laravel framework, the architecture revolves around Laravel's ecosystem. This includes leveraging Laravel for backend development, Blade templating engine for frontend views, MySQL for database management, and integrating third-party services for additional functionalities such as payment processing and analytics.
3. **Defining System Components**: With the technology stack in place, the system components are defined. This involves breaking down the application into smaller, manageable components such as user authentication, product catalog management, shopping cart, order processing, payment integration, and administration panel. Each component is responsible for specific functionality and interacts with other components as needed.

4. **Designing Database Schema**: The database schema is designed to efficiently store and manage data related to products, users, orders, and other entities. Using Laravel's migration and ORM capabilities, database tables, relationships, and constraints are defined to ensure data integrity and optimize performance.
5. **Implementing Scalability Strategies**: Scalability is a critical aspect of architecture development, especially for e-commerce platforms expected to handle a growing number of users and transactions. Horizontal scaling strategies, such as load balancing and clustering, are implemented to distribute traffic across multiple servers. Additionally, caching mechanisms, content delivery networks (CDNs), and database optimization techniques are employed to improve performance and scalability.
6. **Ensuring Security Measures**: Security is paramount in e-commerce architecture. Measures such as HTTPS encryption, secure authentication mechanisms (e.g., OAuth, JWT), input validation, and SQL injection prevention are implemented to protect user data and prevent security breaches. Compliance with industry standards such as PCI DSS (Payment Card Industry Data Security Standard) is also ensured for handling payment information securely.
7. **Integrating Third-Party Services**: To enhance functionality and provide a seamless user experience, third-party services are integrated into the architecture. This may include payment gateways (e.g., PayPal, Stripe), shipping services, analytics tools, and marketing platforms. APIs provided by these services are utilized to enable communication and data exchange between AutoDelovi and external systems.
8. **Implementing Monitoring and Analytics**: Monitoring and analytics capabilities are integrated into the architecture to track system performance, user behavior, and business metrics. Tools such as Google Analytics, New Relic, and custom monitoring dashboards are used to collect data, analyze trends, and identify areas for optimization and improvement.

By following these steps and considerations, the architecture for AutoDelovi is developed to meet the needs of a modern e-commerce platform, ensuring reliability, scalability, security, and performance.

2.5.1 Overview of web application methods.

Web applications are software programs accessed via web browsers over the internet. They enable users to interact with data, perform tasks, and access services online. There are several methods for developing web applications, each with its own strengths, weaknesses, and suitability for different use cases. Here's an overview of some common web application methods:

1. **Traditional Server-Side Rendering (SSR)**: In traditional SSR, the server generates HTML pages dynamically and sends them to the client in response to requests. Technologies such as PHP, Ruby on Rails, and ASP.NET are commonly used for server-side rendering. SSR is suitable for content-heavy websites, where SEO (Search Engine Optimization) is crucial, as search engines can index the content easily. However, SSR may result in slower initial page loads and limited interactivity.
2. **Client-Side Rendering (CSR)**: Client-side rendering involves loading a minimal HTML page from the server and rendering the content dynamically using JavaScript on the client-side. Frameworks

like React, Vue.js, and Angular are popular choices for CSR. CSR offers a more interactive user experience and faster subsequent page loads, as only data is exchanged with the server after the initial page load. However, CSR may have SEO challenges and require careful handling of client-side routing and state management.

3. **Single Page Applications (SPA)**: SPAs are web applications that load a single HTML page initially and dynamically update the content as the user interacts with the application, without requiring full page reloads. SPAs are typically built using frameworks like React, Angular, or Vue.js. SPAs provide a smooth and responsive user experience similar to desktop applications, with fast navigation and seamless interactions. However, SPAs may have SEO challenges and require careful management of client-side state and memory.
4. **Progressive Web Applications (PWA)**: PWAs are web applications that leverage modern web technologies to provide a native app-like experience across different devices and platforms. PWAs use service workers for offline functionality, web app manifests for installation, and responsive design for adaptability. PWAs offer features such as push notifications, background sync, and offline access, making them ideal for mobile-first or offline-first applications. PWAs combine the best of web and native app experiences, providing users with fast, reliable, and engaging experiences.
5. **Serverless Architecture**: Serverless architecture abstracts server management and scaling away from the developer, allowing them to focus on writing code. Serverless applications are built using cloud services like AWS Lambda, Azure Functions, or Google Cloud Functions. Serverless architectures are highly scalable, cost-effective, and require minimal infrastructure management. They are well-suited for event-driven and microservices-based applications, where each function performs a specific task or operation.
6. **Microservices Architecture**: Microservices architecture decomposes applications into small, loosely coupled services that can be developed, deployed, and scaled independently. Each microservice is responsible for a specific business function and communicates with other services via APIs. Technologies like Docker, Kubernetes, and Spring Boot are commonly used for building and deploying microservices. Microservices offer flexibility, scalability, and resilience, but they also introduce complexities in deployment, testing, and monitoring.
7. **Hybrid Approaches**: Hybrid approaches combine elements of different web application methods to leverage the strengths of each approach. For example, a hybrid approach might involve using SSR for initial page load and CSR for subsequent interactions, or combining SPAs with server-side rendering for improved SEO and performance.

When it comes to user experience and interactivity, the choice of web application method can significantly impact how users interact with the application. Single-page applications (SPAs) offer a seamless and fluid user experience, where content is dynamically loaded without page refreshes, leading to a more immersive experience akin to desktop applications. On the other hand, traditional server-side rendering (SSR) may provide a more straightforward browsing experience initially, but may feel less responsive during subsequent interactions.

Performance considerations are crucial, especially in today's fast-paced digital landscape where users expect near-instantaneous responses. Client-side rendering (CSR) and SPAs excel in providing fast and responsive

interfaces by offloading rendering tasks to the client's browser. However, they may require careful optimization to minimize JavaScript bundle sizes and optimize resource loading. Conversely, server-side rendering (SSR) can offer faster initial page loads by delivering pre-rendered HTML content, but may incur additional latency for subsequent interactions.

Search engine optimization (SEO) plays a pivotal role in driving organic traffic to web applications. Traditional SSR approaches are favored for SEO purposes because search engine crawlers can easily parse and index statically generated HTML pages. However, modern CSR and SPA techniques can also achieve good SEO results with proper implementation. Techniques such as server-side rendering of critical content, pre-rendering for search engine bots, and optimizing meta tags and structured data can enhance the discoverability of SPAs and CSR applications.

Scalability and maintenance considerations are essential for ensuring the long-term success of web applications. Microservices architecture, serverless computing, and containerization technologies like Docker and Kubernetes offer scalable deployment options for modern web applications. Additionally, automation tools and continuous integration/continuous deployment (CI/CD) pipelines streamline the development, testing, and deployment processes, reducing maintenance overhead and enhancing developer productivity.

In summary, choosing the right web application method involves weighing various factors such as user experience, performance, SEO, scalability, and maintenance requirements. Each method has its own set of trade-offs and considerations, and the optimal approach depends on the specific needs and objectives of the project. By carefully evaluating these factors and selecting the appropriate method, developers can build web applications that deliver exceptional user experiences and meet business objectives effectively.

2.5.2 Database development with Entity Framework Code First

In the context of my project, implementing Entity Framework Code First methodology offers several advantages for developing the database schema.

Firstly, defining entities in C# classes closely aligns with the object-oriented nature of my application. For instance, I can easily represent entities like `Product`, `Purchase`, `SoldProduct`, `Cart`, `Brand`, `Vehicle`, `User`, `Comment`, and `Category` as classes, with properties mapping to database columns.

This approach enhances code readability and maintainability, as the structure of the database entities mirrors the application's domain model.

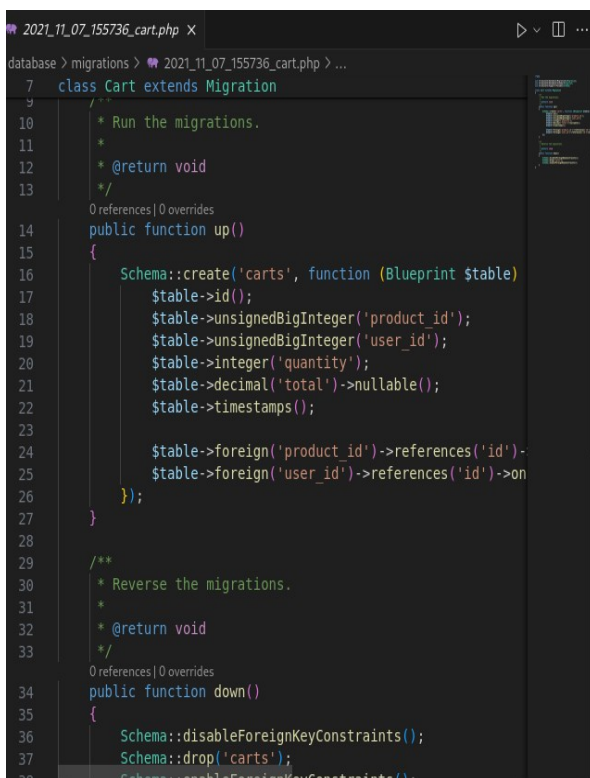
Secondly, establishing relationships between entities using navigation properties simplifies the representation of complex data relationships. For instance, I can define relationships such as one-to-many between `Product` and `Brand`, `Purchase` and `Product`, `SoldProduct` and `Product`, and many-to-many between `Product` and `Category`. This allows for intuitive navigation and querying of related entities within the application code.

The automatic generation of the database schema based on entity definitions streamlines the database setup process. When running the application for the first time, Entity Framework Code First generates the necessary tables (`products`, `purchases`, `sold_products`, `carts`, `brands`, `vehicles`,

users, comments, categories, and the default Laravel tables), eliminating the need for manual database creation and management.

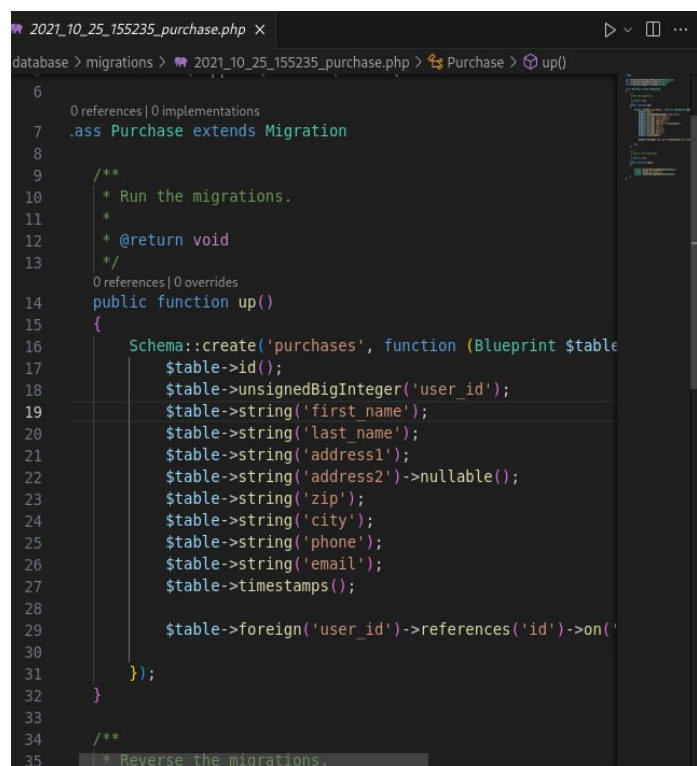
Furthermore, the use of migrations facilitates schema evolution over time as the project requirements change. By creating and applying migrations, I can easily update the database schema without losing existing data. This ensures that the database remains in sync with the evolving application logic, enabling seamless integration of new features and functionalities.

Overall, Entity Framework Code First offers a robust and flexible approach to database development, aligning closely with the object-oriented principles of my project. By leveraging this methodology, I can efficiently manage the database schema, define complex relationships, and adapt to evolving requirements, thereby enhancing the scalability and maintainability of my application.



```
2021_10_07_155736_cart.php X
database > migrations > 2021_10_07_155736_cart.php > ...
7 class Cart extends Migration
8 /**
9  * Run the migrations.
10  *
11  * @return void
12  */
13 0 references|0 overrides
14 public function up()
15 {
16     Schema::create('carts', function (Blueprint $table)
17     {
18         $table->id();
19         $table->unsignedBigInteger('product_id');
20         $table->unsignedBigInteger('user_id');
21         $table->integer('quantity');
22         $table->decimal('total')->nullable();
23         $table->timestamps();
24
25         $table->foreign('product_id')->references('id')->on('products');
26         $table->foreign('user_id')->references('id')->on('users');
27     });
28 }
29 /**
30  * Reverse the migrations.
31  *
32  * @return void
33  */
34 0 references|0 overrides
35 public function down()
36 {
37     Schema::dropIfExists('carts');
38     Schema::disableForeignKeyConstraints();
39 }
```

Fig. 2.3.migration for carts



```
2021_10_25_155235_purchase.php X
database > migrations > 2021_10_25_155235_purchase.php > Purchase > up()
6
7 class Purchase extends Migration
8 /**
9  * Run the migrations.
10  *
11  * @return void
12  */
13 0 references|0 overrides
14 public function up()
15 {
16     Schema::create('purchases', function (Blueprint $table)
17     {
18         $table->id();
19         $table->unsignedBigInteger('user_id');
20         $table->string('first_name');
21         $table->string('last_name');
22         $table->string('address1');
23         $table->string('address2')->nullable();
24         $table->string('zip');
25         $table->string('city');
26         $table->string('phone');
27         $table->string('email');
28         $table->timestamps();
29
30         $table->foreign('user_id')->references('id')->on('users');
31     });
32 }
33 /**
34  * Reverse the migrations.
35  */
```

Fig. 2.4. migration for purchase

```

2021_10_22_155632_product.php X
database > migrations > 2021_10_22_155632_product.php > Product
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 class Product extends Migration
8 {
9     /**
10      * Run the migrations.
11      *
12      * @return void
13      */
14     public function up()
15     {
16         Schema::create('products', function (Blueprint $table) {
17             $table->id();
18             $table->string('name');
19             $table->string('description');
20             $table->string('price');
21             $table->string('image');
22             $table->unsignedBigInteger('category_id');
23             $table->unsignedBigInteger('brand_id');
24             $table->enum('vehicle_type', [ 'Motor', 'Auto', ' ']);
25             $table->timestamps();
26
27             $table->foreign('category_id')->references('id')
28             $table->foreign('brand_id')->references('id')->on('brands');
29         });
30     }
31 }

```

Fig. 2.5. migration for products

```

2021_10_22_155556_category.php X
database > migrations > 2021_10_22_155556_category.php > Category
3 Illuminate\Database\Migrations\Migration;
4 Illuminate\Database\Schema\Blueprint;
5 Illuminate\Support\Facades\Schema;
6
7 class Category extends Migration
8 {
9     /**
10      * Run the migrations.
11      *
12      * @return void
13      */
14     public function up()
15     {
16         Schema::create('categories', function (Blueprint $table) {
17             $table->id();
18             $table->unsignedBigInteger('category_id')->nullable();
19             $table->string('name');
20             $table->string('image');
21             $table->timestamps();
22         });
23     }
24
25     /**
26      * Reverse the migrations.
27      *
28      * @return void
29      */
30     public function down()
31     {

```

Fig. 2.6. migration for categories

2.5.3 Description of database table structures

2.5.3.1 Use Case Diagram

The diagrams in this work were created using Visual Paradigm and The Unified Modeling Language (UML), which is a modeling language used to visualize systems. The process of modeling begins with case modeling, which helps define the functional requirements of the system .

One of the diagrams used is the Use Case Diagram, which is helpful for planning the usage of the system. It consists of the system boundary, actors, use cases, and relationships (associations). Actors are external entities that interact with the system but are not directly a part of it. For example, a user is considered an actor since they use the system, but they are not part of the system itself.

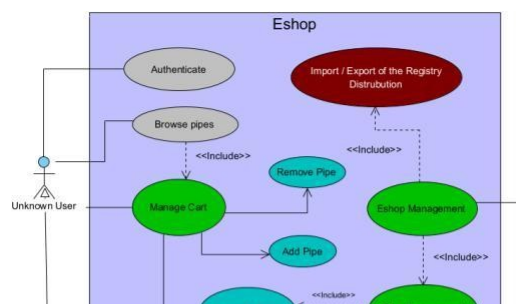


Fig. 2.7. Use Case Diagram of the implemented e-shop

Use cases describe the behavior of the system when an actor performs an action. These use cases are within the system boundary, which represents the system itself. Anything within the boundary is considered internal to the system, while anything outside of it is external. Additionally, associations represent the relationships between the components. Figure 3.1 displays the Use Case Diagram for the system in this study

2.5.3.2 Database diagram

In order to handle the data effectively, it needs to be stored in a database. For this project, PostgreSQL, a relational database, has been chosen. To model the database can be utilized and it is not a part of the UML, it is widely used for designing data structures and database systems [22]. It plays a similar role to the Class Diagram but specifically focuses on database design. It is highly favored as it helps avoid design errors, such as creating tables that contain unnecessary information. The Database Diagram for the e-commerce platform can be observed in Figure 3.2.

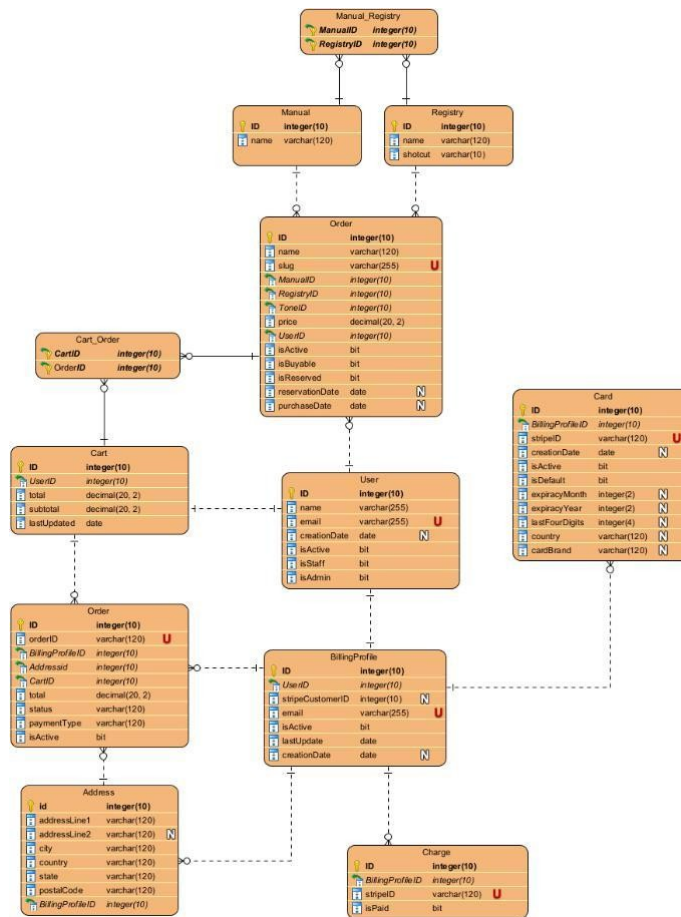


Fig. 2.8. Database diagram of the implemented e-shop

2.5.3.3 Project architecture

The project structure is a key aspect in understanding the project and facilitating navigation. Fortunately, Django provides a well-defined project structure that organizes each app into separate Python packages. This structured approach enables efficient collaboration, even for individuals who are not familiar with the project. In the provided directory structure, the project is named "my_project_with_apps" and consists of two apps: "my_empty_app" and "cart_app." The project directory contains three packages, including the ones for the apps and a package with the same name as the project. The package with the project name contains different files compared to the app folders. Notably, the settings.py file within this package contains crucial configurations for the website, such as the list of apps being used. It is essential to add apps to the settings.py file for them to function correctly. Moreover, settings.py includes information about the database, allowed hosts, and other important settings.

```

my_project_with_apps
├── my_empty_app
│   ├── migrations
│   │   └── __init__.py
│   ├── admin.py
│   ├── apps.py
│   └── ...
  
```


Fig. 2.9. Project architecture

Additionally, there is a significant contrast between "my_empty_app" and "cart_app." The "cart_app" features a template directory that houses the HTML files related to the app. For instance, in code snippet, the view representing the shopping cart can be observed.

2.5.3.4 Immediate Payments using Stripe

Once the user has reviewed all the provided information and proceeded to make a payment, the website initiates the payment process through Stripe, a payment gateway. Based on Stripe's response, the user is then redirected to either the success or fail endpoint. If the response indicates a successful transaction, the products are marked as purchased, and website users can view the name of the adoptive user in the details of the purchased products.

Furthermore, Stripe generates a unique Stripe ID for each user, which is utilized to securely store and process credit card information for payment execution. It is important to note that the credit card information is exclusively stored within Stripe's database and is not accessible to the website administrator, except for limited details such as the last four digits of the card, card brand, expiration month, and year. These details are only used to assist the user in confirming the card being used for the transaction.

2.5.3.5 Delayed Payment Methods

The checkout process and payment via Stripe, the website offers users the option to reserve products using different payment methods. One of the supported methods is a money transfer to the adoption account, which can be done online or offline.

It's important to note that money transfers between accounts may take up to three days, and there may be limitations on the number of API calls per day. To avoid potential issues, a transaction check is performed every 24 hours. The API is used to retrieve account transactions, allowing access to up to 24 months of transaction history. However, for the purposes of the e-shop's check frequency, the transaction history is reduced to 30 days.

To schedule API calls at specific times, a task scheduler is required. While Django does not provide a built-in task scheduler, an external scheduler such as Django-cron can be used. Django-cron allows developers to run Django/Python code on a recurring basis by creating management commands. These commands need to be added to the cron, which is the UNIX system scheduler responsible for executing commands or scripts at predetermined times.

It's worth mentioning that the API requires the admin to log into the account using a link provided during the setup phase. API has access to the account information for 90 days. Some admins may prefer to perform the transaction check independently to avoid the need to regain access every 90 days [23].

In addition to online payments, the admin also handles offline payments, such as cash payments. This requires admin access to the Admin Panel, which provides the ability to make changes to the adoption website. The Admin Panel, depicted in Figure 3.4, grants access to all database-related information and allows the admin to modify it as needed.

The database table structures are designed to reflect the various entities and their relationships within the application. Let's delve into the structure of each table and explore the relationships between them:

1. Products Table:

- This table represents the products available in the e-commerce platform.
- It includes attributes such as product name, description, price, and image URL.
- Each product may belong to a specific brand, which establishes a one-to-many relationship between the `Products` and `Brands` tables.

2. Purchases Table:

- The purchases table records transactions where users buy products.
- It includes attributes such as purchase ID, product ID, user ID, purchase date, and quantity.

- Each purchase is associated with a specific user and product, establishing one-to-many relationships with the `Users` and `Products` tables.

3. `SoldProducts Table`:

- This table tracks the products that have been sold.
- It includes attributes such as sale ID, product ID, sale date, and quantity.
- Each sold product corresponds to a specific product entry, establishing a one-to-many relationship with the `Products` table.

4. `Cart Table`:

- The cart table stores the products that users have added to their shopping carts.
- It includes attributes such as cart ID, product ID, user ID, and quantity.
- Each cart entry is associated with a specific user and product, establishing one-to-many relationships with the `Users` and `Products` tables.

5. `Brands Table`:

- This table contains information about the brands associated with products.
- It includes attributes such as brand ID and brand name.
- Each brand may have multiple products associated with it, establishing a one-to-many relationship with the `Products` table.

6. `Vehicle Table`:

- The vehicle table stores information about vehicles or car accessories.
- It includes attributes such as vehicle ID, model, year, and description.
- Each vehicle entry may be associated with one or more products, establishing a one-to-many relationship with the `Products` table.

7. `Users Table`:

- This table stores user information, including login credentials and personal details.
- It includes attributes such as user ID, username, email, password hash, and role.
- Users may have various interactions within the system, such as making purchases or adding items to their cart.

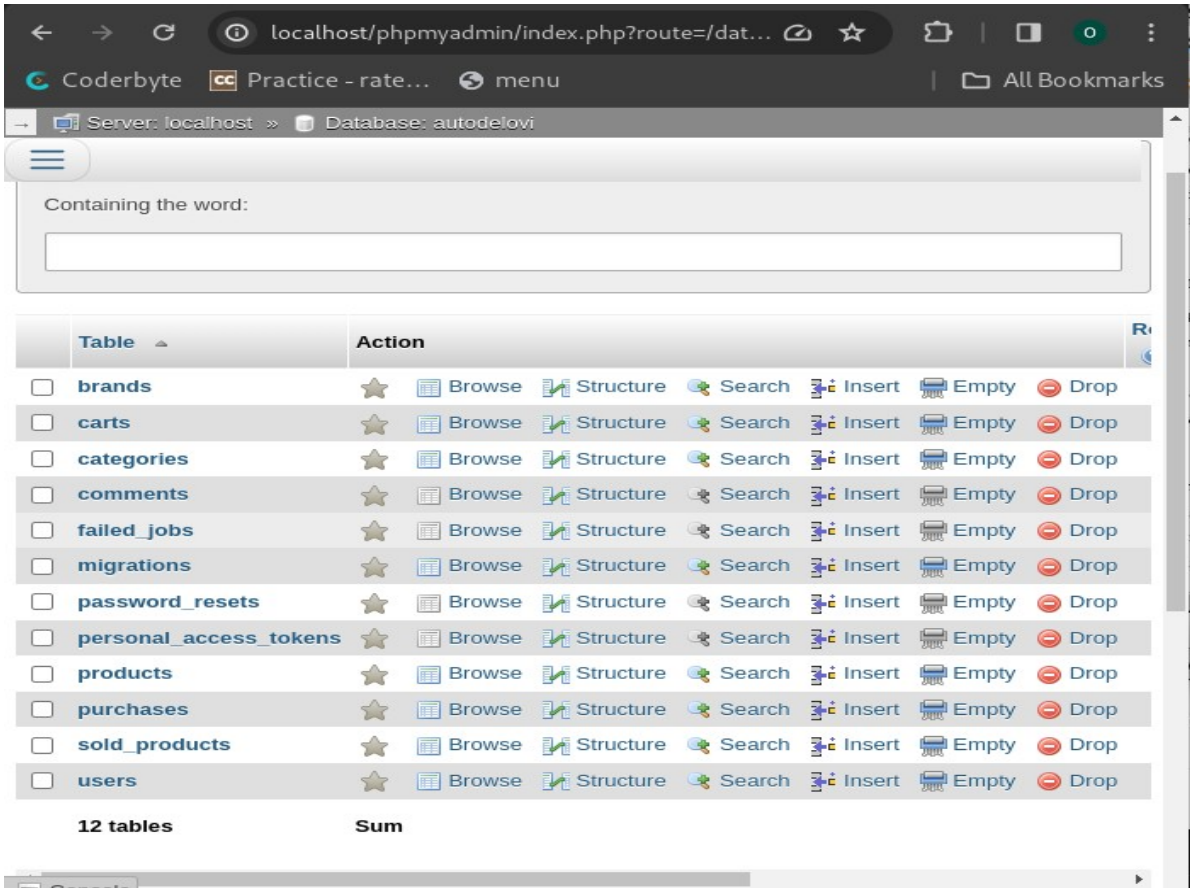
8. `Comments Table`:

- The comments table allows users to leave comments on products.
- It includes attributes such as comment ID, user ID, product ID, comment text, and timestamp.
- Each comment is associated with a specific user and product, establishing one-to-many relationships with the `Users` and `Products` tables.

9. `Categories Table`:

- This table categorizes products into different categories.
- It includes attributes such as category ID and category name.
- Products can belong to one or more categories, establishing a many-to-many relationship between the `Products` and `Categories` tables.

These tables collectively form the database schema for the e-commerce platform, capturing the various entities, their attributes, and the relationships between them. This relational structure allows for efficient data storage, retrieval, and manipulation, enabling seamless interactions within the application.



The screenshot shows the phpMyAdmin interface for the 'autodelovi' database. At the top, there's a search bar with the text 'Containing the word:'. Below it, a table lists 12 tables: brands, carts, categories, comments, failed_jobs, migrations, password_resets, personal_access_tokens, products, purchases, sold_products, and users. Each table has a star icon and a set of action icons: Browse, Structure, Search, Insert, Empty, and Drop. At the bottom of the table, it says '12 tables' and 'Sum'.

Table	Action
<input type="checkbox"/> brands	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> carts	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> categories	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> comments	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> failed_jobs	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> migrations	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> password_resets	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> personal_access_tokens	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> products	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> purchases	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> sold_products	★ Browse Structure Search Insert Empty Drop
<input type="checkbox"/> users	★ Browse Structure Search Insert Empty Drop
12 tables	Sum

Fig. 2.10. all tables in autodevilo

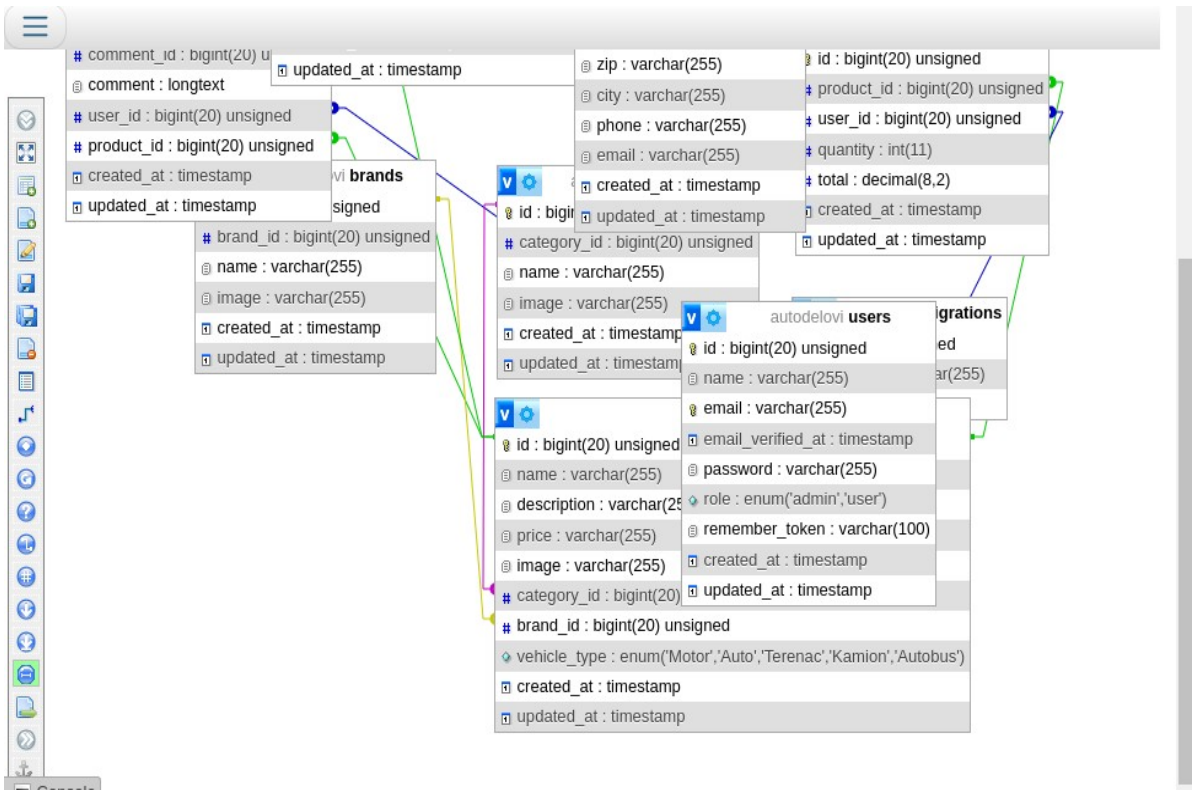


Fig. 2.11. database design

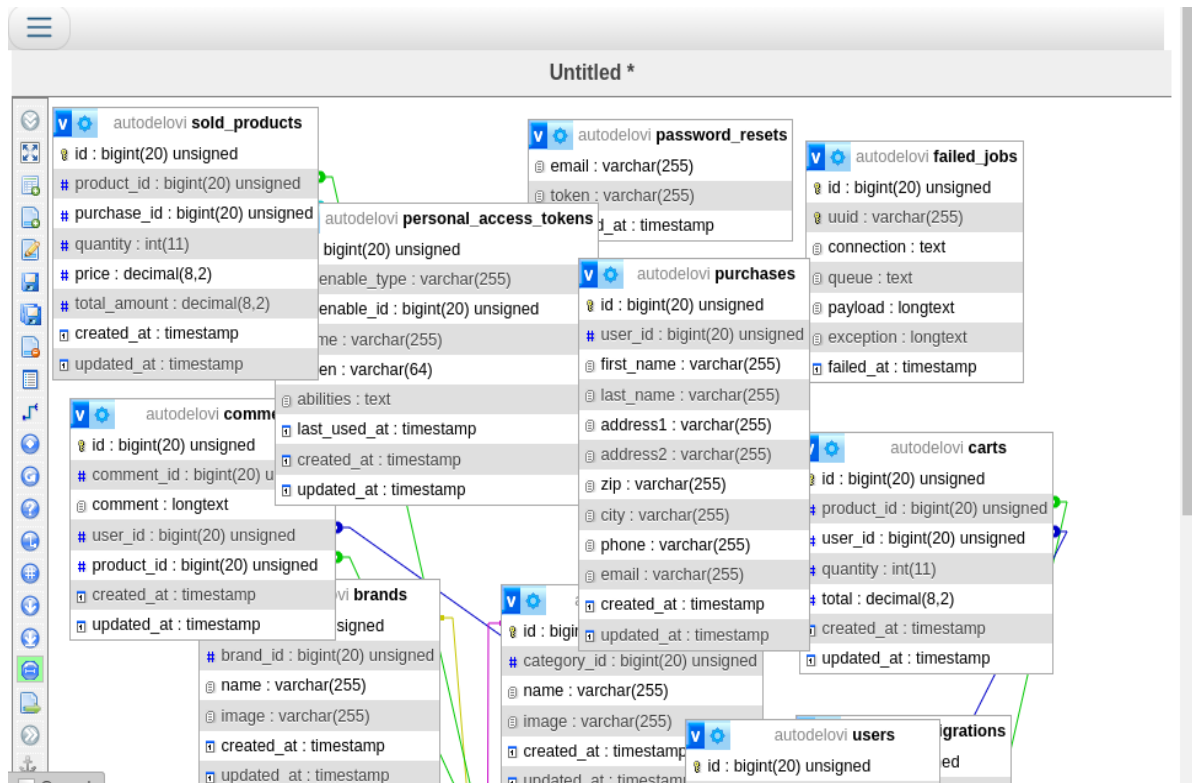


Fig. 2.12. database design

2.6 Conclusions to the section

In the evaluation of client-side tools for our project, HTML and CSS emerged as pivotal components in web development. HTML forms the foundation, providing the structure and content of web pages, while CSS enhances the visual presentation and styling, crucial for crafting an attractive and user-friendly frontend interface.

Turning to server-side tools, our focus rested on PHP and the Laravel framework. PHP, renowned for its versatility and widespread adoption, serves as the backbone for server-side development in our project. Its simplicity, readability, and extensive libraries make it a preferred choice for web application development. Laravel, a high-level PHP web framework, complements PHP by offering a comprehensive suite of tools and functionalities tailored for building robust and scalable web applications.

The synergy between PHP and Laravel yields a potent combination for server-side development in our project. Together, they enable efficient data processing, server management, and seamless integration with databases, ensuring the smooth functioning of our web applications.

The analysis of both client-side and server-side tools underscores their synergistic roles in our project's web application development. While HTML and CSS govern the frontend presentation and user experience, PHP and Laravel orchestrate backend processing and server-side functionalities. By leveraging these tools effectively, we can craft sophisticated and interactive web applications that align with the requirements of our e-commerce platform for car accessories.

3. ANALYSIS AND TESTING OF SOFTWARE QUALITY

1. OBJECT OF TESTS

The web resource of the Center for Electronic Education, which is a website created on the LAMP Stack platform using PHP and Laravel framework and PHP Unit for testing.

2. PURPOSE OF TESTING

During the testing process, the following should be checked:

- 1) functional effectiveness of the web application components;
- 2) seamless integration and interaction with the product database;
- 3) compatibility of data exchange formats and protocols within the e-commerce ecosystem;
- 4) adequate data security levels to protect sensitive user and transaction information;
- 5) usability and user-friendliness of the web application;
- 6) Compliance of the web application design with the specifications outlined in the technical requirements.

3. METHODS OF TESTING

Testing is performed using a combination of White Box and Black Box testing methodologies. Both the source code and the operational aspects of the software product are evaluated for their adherence to functional requirements. The testing procedure encompasses "system testing" levels.

The following methods are employed:

- 1) functional testing, specifically at the Critical Path Testing level (baseline testing);
- 2) performance testing of the software, including Stability testing and Load testing;
- 3) user Interface (UI) testing.

4. MEASURES AND PROCEDURE OF TESTING

Testing is conducted using testing tools such as Jest and Cypress.

The functionality of the web application is assessed through:

- 1) dynamic manual testing - by entering limit and unacceptable values in editable fields;
- 2) dynamic manual testing for compliance with functional requirements;
- 3) static code analysis;
- 4) cross-browser compatibility testing;
- 5) stress testing for maximum load conditions;
- 6) resilience testing under varying conditions;
- 7) usability testing;
- 8) user Interface testing.

3.1 Results of the test login to the system and the purchase process

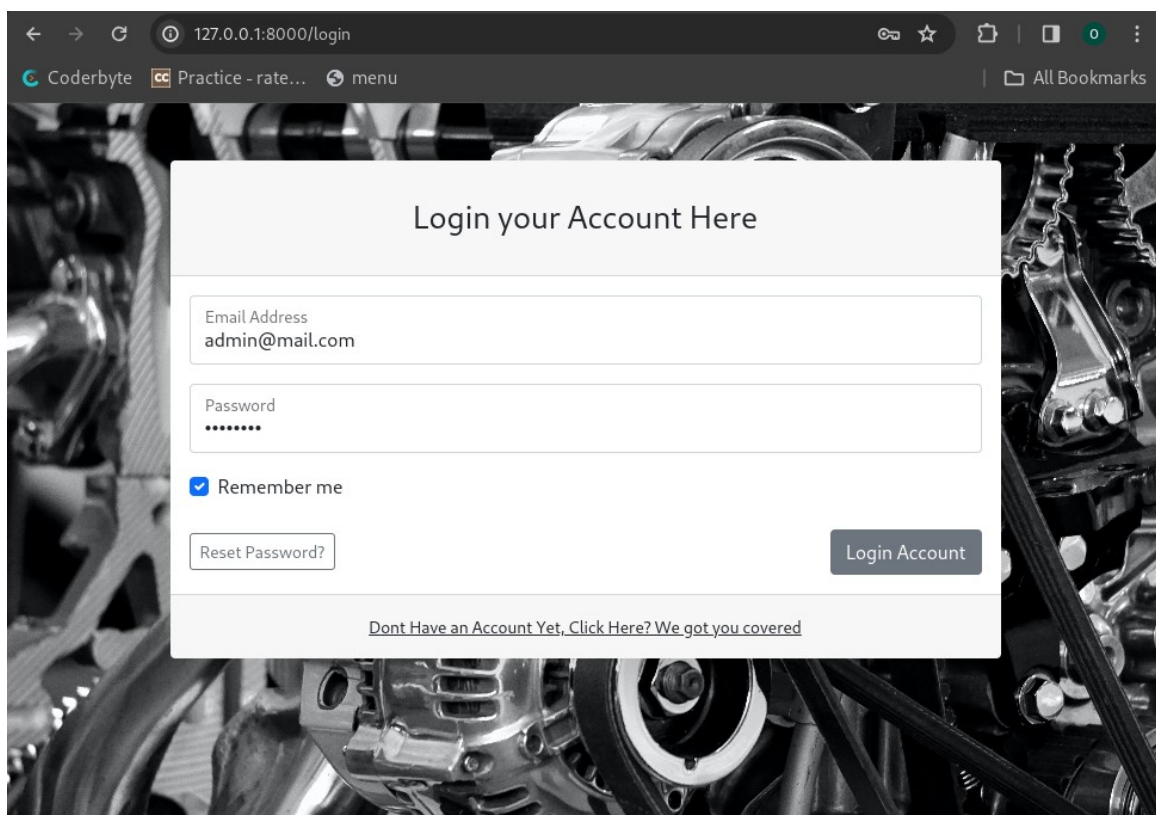


Fig: 3.0 logging in page with admin credentials

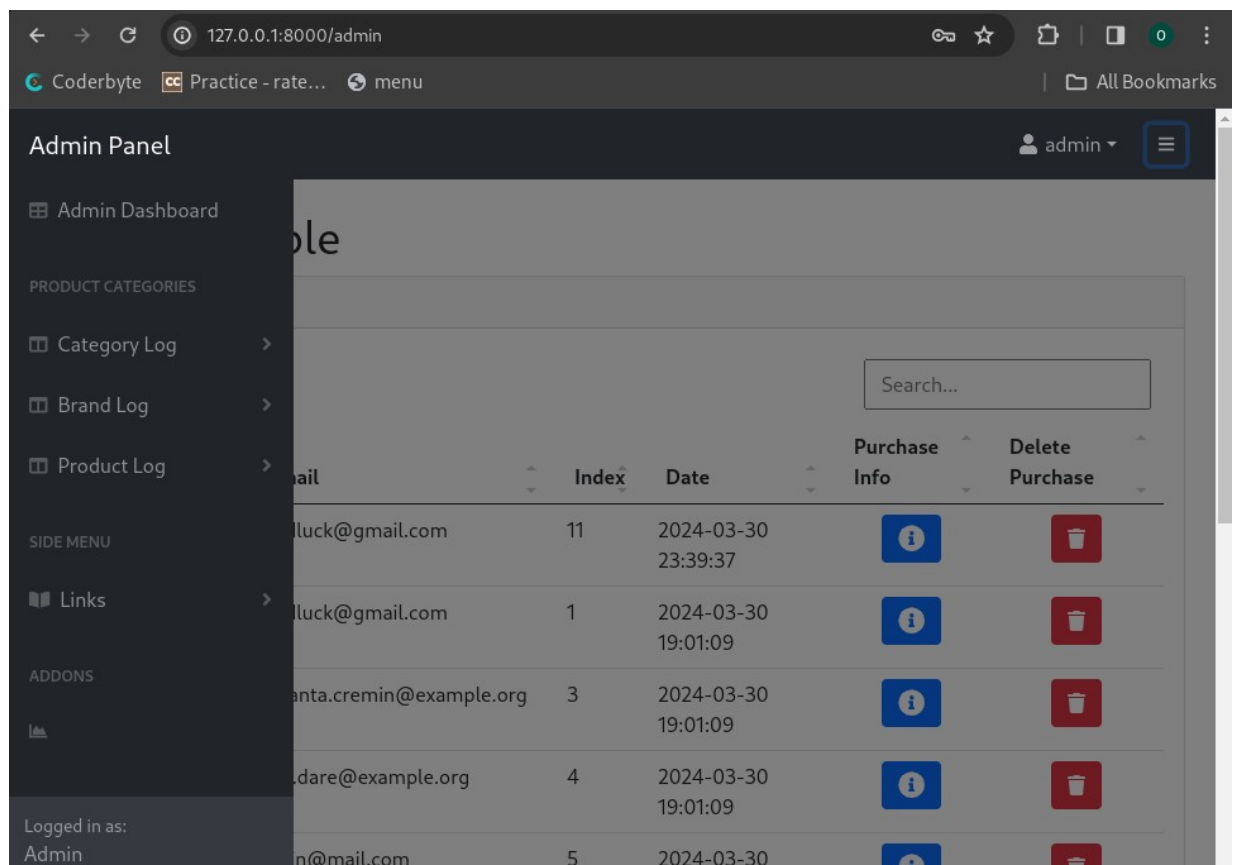


Fig: 3.1 logged in to admin dashboard

3.2 Testing user registration

The screenshot shows a web browser at the address 127.0.0.1:8000/register. The page features a registration form titled "Fill the Form" with a background image of a car engine. The form contains fields for "First Name", "Last Name", "Email Address", "Password", and "Password Confirmation". A "Register with us" button is at the bottom, along with a link for users who already have an account.

Fill the Form

First Name: goodluck

Last Name: goodluck

Email Address: goodluck@gmail.com

Password:

Password Confirmation:

[Register with us](#)

[Already Have an Account? Click here](#)

Fig: 3.2 Registration page filled with user details



Review Our Top Products:



Fig: 3.3 Logged in User(goodluck@gmail.com)

3.3 Testing the functionality of the administrator

A screenshot of an Admin Panel interface. The page title is 'Admin Panel' and the user is logged in as 'admin'. The main section is titled 'Create Products'. It contains several input fields and dropdown menus for creating a new product. The fields are: Name (VX-380 Benz), Description (this is the new benz model for 2024), Price (1000000), Category (odio), Brand (minus), and Vehicle (Auto). There is also a label 'Image Of Product:' at the bottom. The browser's address bar shows the URL '127.0.0.1:8000/admin/product/create'.

Fig: 3.4 create new product

127.0.0.1:8000/admin/product/create

Coderbyte Practice - rate... menu All Bookmarks

Admin Panel admin


1000000

Category
odio

Brand
minus

Vehicle
Auto

Image Of Product:



Choose File download.jpeg

Create Product

Fig: 3.5 creating new product 2

127.0.0.1:8000/admin/product/create

Coderbyte Practice - rate... menu All Bookmarks

Admin Panel admin

Product was created successfully!

Create Products

Name
Name..

Description
Description..

Price
100

Category
praesentium

Brand
nostrum

Vehicle
Autobus

Fig: 3.6 successful submission of products

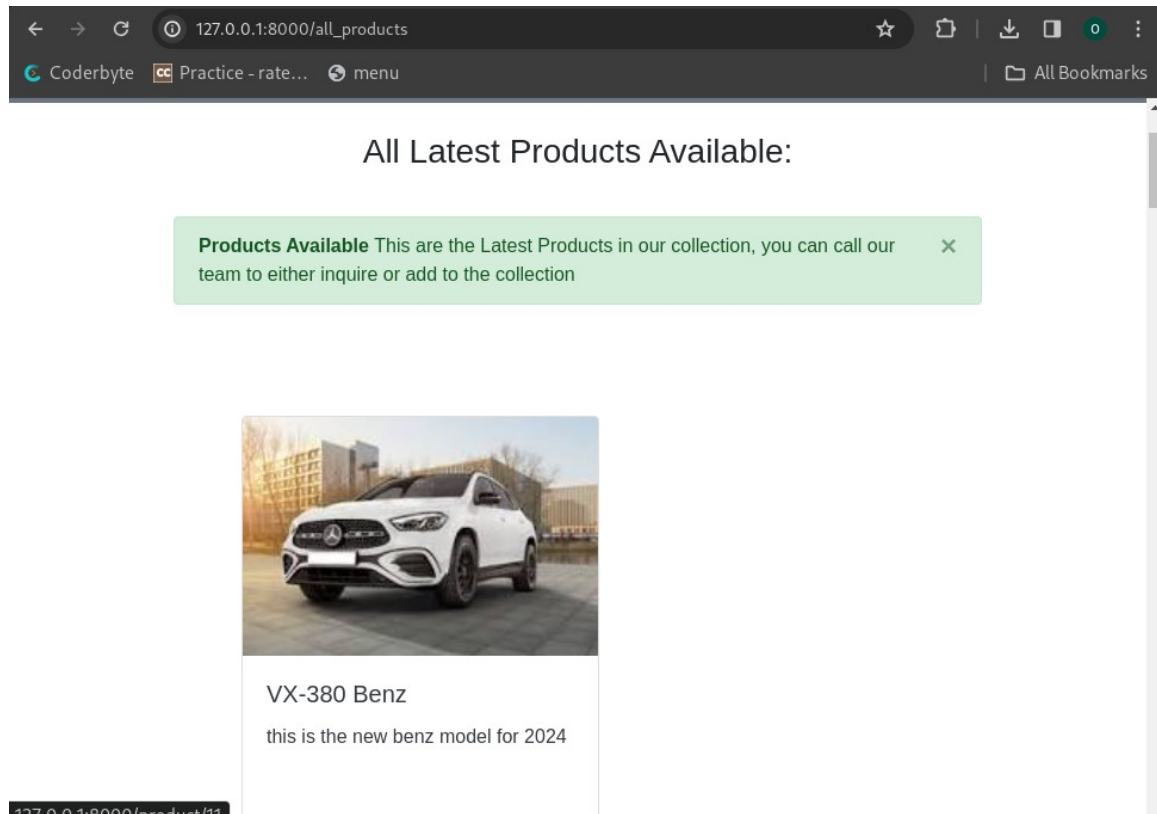


Fig: 3.7 view of newly added product

3.4 Conclusions to the section

The development of a web-application for an electronic commerce platform involves various aspects such as design and implementation, as well as data synchronization.

The design and implementation phase focuses on creating an intuitive and visually appealing user interface for the web-application. This includes front-end development using tools like HTML, CSS, and JavaScript to design the layout, colors, fonts, and other visual elements of the application. The front-end developers are responsible for ensuring that the website is user-friendly and provides a seamless navigation experience. On the back-end, server-side tools like Python and frameworks like Django are used to handle data synchronization and communication between the client and server. These tools enable the web-application to interact with databases, process user requests, and perform necessary calculations or operations.

Data synchronization plays a crucial role in ensuring that the information displayed on the web-application remains up-to-date and consistent across different devices and platforms. This involves handling user inputs, updating the database, and propagating changes to all relevant components of the application. Tools like APIs and task schedulers are utilized to automate and manage these synchronization processes effectively.

The development of a web-application for an electronic commerce platform requires a combination of front-end and back-end development tools, effective data synchronization techniques, and a well-designed user interface. The successful implementation of these elements ensures a smooth and efficient user experience while interacting with the electronic commerce platform.

CONCLUSIONS

The web application for the electronic commerce platform was created with a user-friendly interface that enables seamless online shopping experiences for customers and facilitates effective management of the platform for business owners. Throughout the project, a combination of HTML, CSS, and JavaScript was used for frontend development, while Python and SQLite were employed for backend implementation.

The choice of HTML, CSS, and JavaScript for frontend development proved to be effective in creating a user-friendly and visually appealing interface. These technologies allowed for the seamless integration of various design elements and the implementation of interactive features, enhancing the overall user experience.

On the backend, Python and SQLite were selected as the primary technologies. Python provided a solid foundation for handling the application's business logic and server-side operations, while SQLite proved suitable for managing and storing the application's data.

The successful integration of these technologies resulted in a well-rounded web application for the electronic commerce platform. The application offers essential features such as product catalog management, shopping cart functionality, payment integration, and order processing. These features collectively contribute to a seamless and convenient online shopping experience for customers.

LIST OF REFERENCES

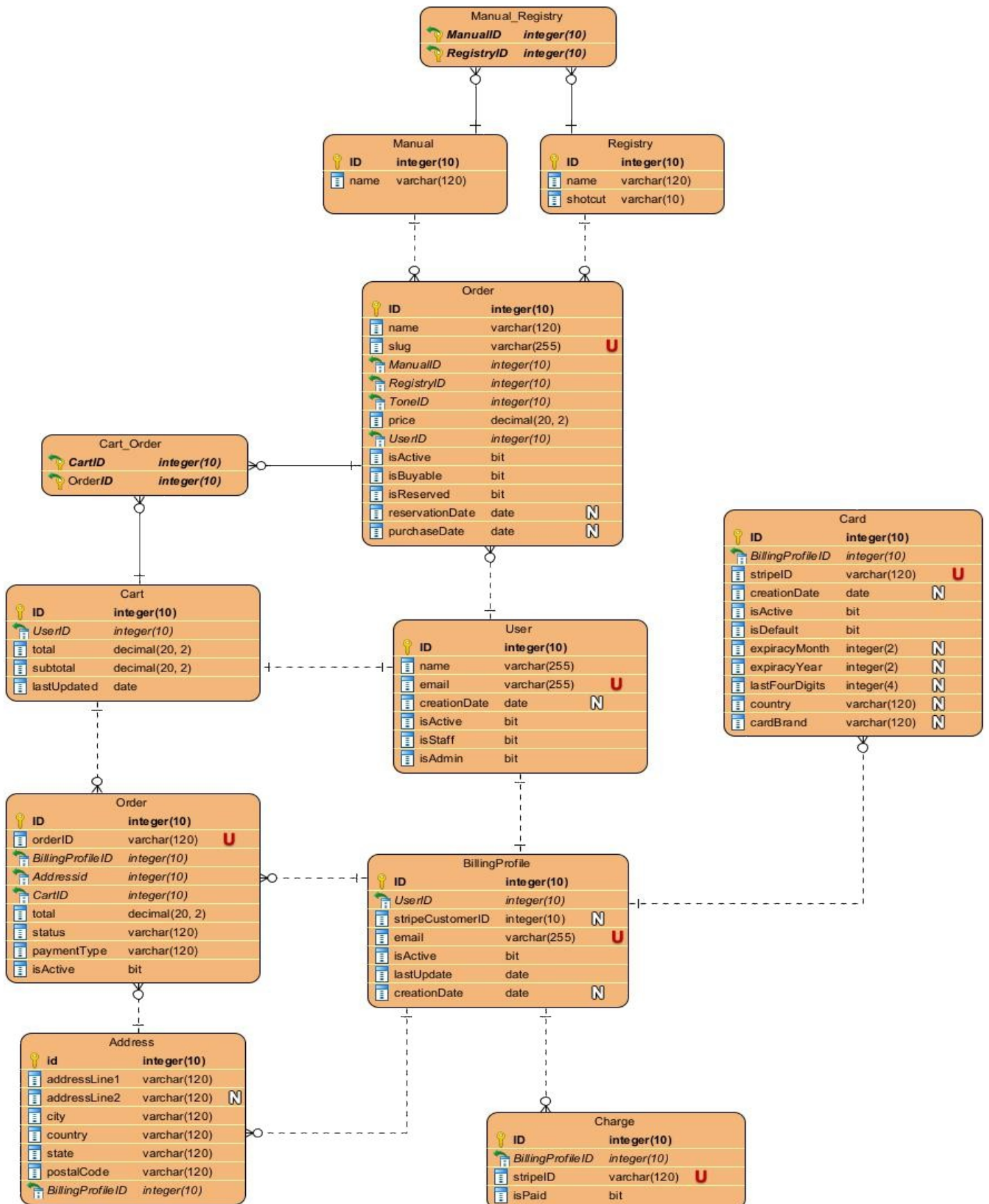
1. Vincent, W. S. Django for Beginners: Build Websites with Python and Django. Welcometocode, 2020. – 292 p.
2. George, N. Build a Website with Django 3: A Complete Introduction to Django
3. GNW Independent Publishing, 2021. – 262 p.
3. Pinkham, A. Web Development in Python with Django: Building Backend Web Applications and APIs with Django. Pearson Education, Limited, 2019. – 197 p.
4. Bendoraitis, A., Kronika, J. Django 3 Web Development Cookbook: Actionable Solutions to Common Problems in Python Web Development, 4th Edition. Packt Publishing, Limited, 2020. – 608 p.
5. Uzayr, S. B. Mastering Django: A Beginner's Guide. CRC Press LLC, 2022. – 302 p.

6. Ravindran, A. Django Design Patterns and Best Practices: Easily Build Maintainable Websites with Powerful and Relevant Django Design Patterns. Packt Publishing, Limited, 2015. – 222 p.
7. Smith, D. Python Machine Learning: The Crash Course for Beginners to Programming and Deep Learning, Artificial Intelligence, Neural Networks and Data Science. Scikit Learn, Tensorflow, Pandas and Numpy. Independently published, 2019. – 142 p.
8. Korper, S. The E-commerce book: Building the E-empire. San Diego : Academic Press, 2000. – 284 p.
9. Moore A. Create your own online store in a weekend. 2014. – 176 p.
10. J M. R. Electronic commerce. 4th ed. New York, NY : Aspen Publishers, 2011. – 894 p.
11. Mourya S. K. E-Commerce. Alpha Science International, Limited, 2015. – 264 p.
12. Electronic Commerce 2008 (Electronic Commerce) / P. Marshall et al. Prentice Hall, 2007. – 1008 p.
13. Stewart, S. Python Programming: Python Programming for Beginners, Python Programming for Intermediates. Platinum Press LLC, 2019. – 180 p.
13. Bennett, J., Overland, B. Advanced Python Programming. Pearson Technology Group Canada, 2019. – 672 p.
14. Liang, Y. D. Programming. Pearson Education, Limited, 2012. – 768 p.
15. Crispin, L. Testing Extreme Programming. Boston : Addison-Wesley, 2003. – 306 p.
17. Software Testing Techniques: Finding the Defects that Matter (Programming Series) / G. Miller et al. Charles River Media, 2004. – 362 p.

18. Python [Electronic resource]. – Access mode:
<https://docs.python.org/3/tutorial/>
19. Python web framework Django [Electronic resource]. – Access mode:
<https://www.djangoproject.com/>
20. E-commerce Website using Django [Electronic resource]. – Access mode:
<https://www.geeksforgeeks.org/e-commerce-website-using-django/>
21. LBTest: A Learning-Based Testing Tool for Reactive Systems [Electronic resource]. – Access mode:
<https://www.semanticscholar.org/paper/LBTest%3A-A-Learning-Based-Testing-Tool-for-Reactive-Meinke-Sindhu/c3be7eafa7e203690df3606b6279e8e2b5de2470>
22. Testing Safety Critical Avionics Software Using LBTest [Electronic resource].
– Access mode: <https://www.semanticscholar.org/paper/Testing-Safety-Critical-Avionics-Software-Using-Stenlund/0822e41416154fd2301f0dbadd30f41bfc1c2582>
23. Testing in Django [Electronic resource]. – Access mode:
<https://realpython.com/testing-in-django-part-1-best-practices-and-examples/>

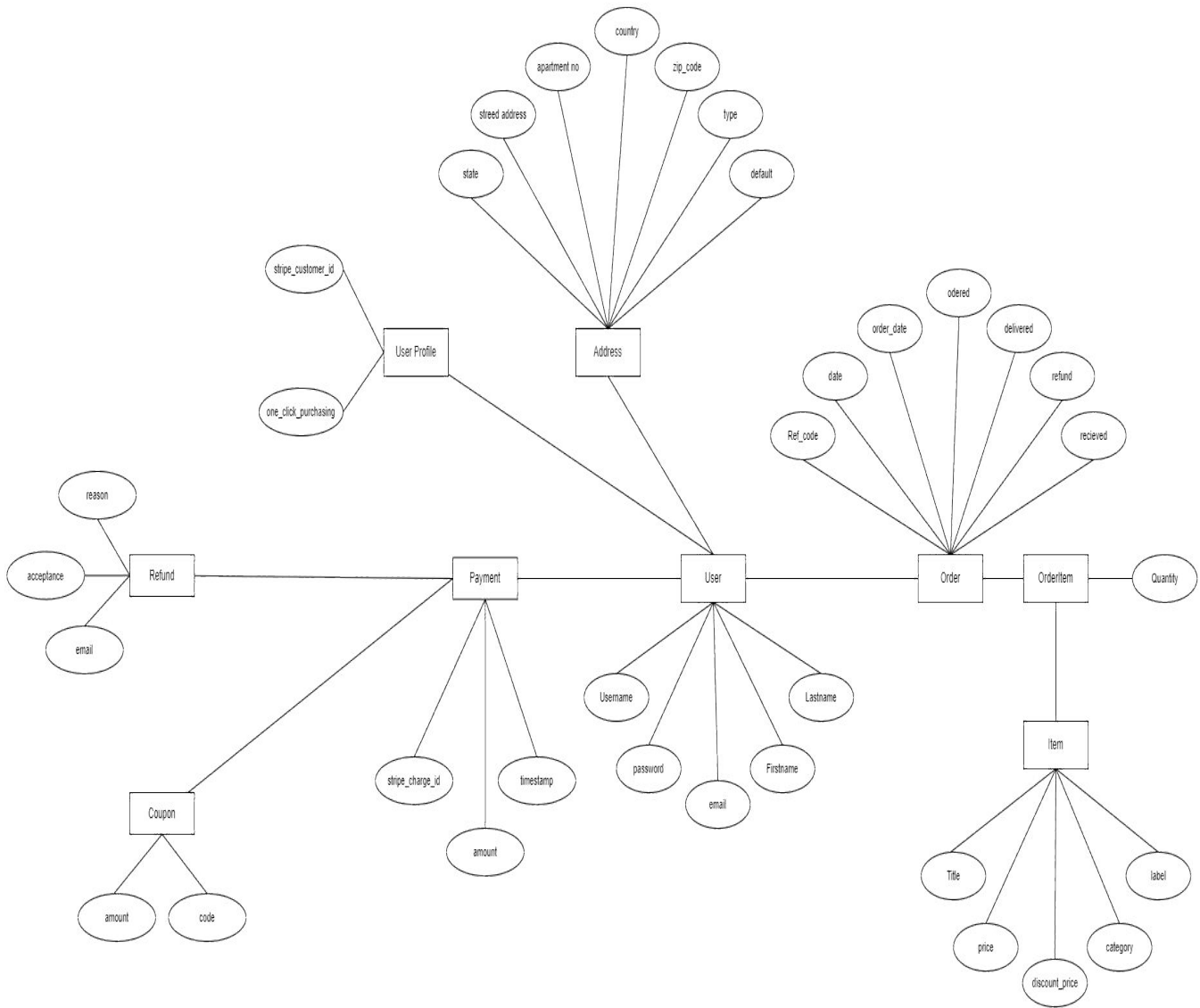
APPENDICES

Appendix 1 Database structure



Appendix 2

Entity-relationship diagram



Appendix 3

Copy of presentation



NATIONAL TECHNICAL UNIVERSITY OF UKRAINE
«IGOR SIKORSKY
KYIV POLYTECHNIC INSTITUTE»
FACULTY OF APPLIED MATHEMATICS



DEPARTMENT OF COMPUTER SYSTEMS SOFTWARE

WEB-APPLICATION FOR E-COMMERCE ON CAR ACCESSORIES PLATFORM

Performed by: Mohamed Islam KHELILI

Supervisor: Associate Professor of DCSS, Ph.D., Assoc. Prof.

Dmytro NOVAK

Kyiv - 2023

RELEVANCE





THE PURPOSE OF THE DIPLOMA PROJECT



The purpose of the diploma project is to develop a web-application for e-commerce on car Accessories platform, with the goal of providing businesses with a comprehensive online platform to showcase their products and enabling customers to engage in secure and efficient online shopping experiences. By creating an intuitive user interface, incorporating robust security measures, and implementing features such as product catalog management, streamlined checkout processes, the project seeks to enhance the overall car commerce ecosystem, fostering growth and innovation in the industry while catering to the evolving demands of businesses and customers.

3/21



TASKS OF THE DIPLOMA PROJECT



1. Analyze existing software solutions.
2. Determine the requirements for the software being developed.
3. Analyze and choose software implementation tools.
4. Develop a web-application for e-commerce on car Accessories platform according to the requirements.
5. Test the developed software and analyze the results.
6. Formulate ways of further development of the project .

4/21



CHALLENGES

1. **Security:** Keeping customer information and transactions secure is important to prevent data breaches and fraud.
2. **Scalability:** As the user base and transactions increase, the application should be able to handle the growing demand without slowing down.
3. **User Experience:** Designing an easy-to-use interface that is visually appealing and works well on different devices is crucial for customer satisfaction.
4. **Integration:** Connecting with payment gateways, shipping providers, and other systems requires smooth integration to ensure seamless operations.
5. **Performance:** Optimizing the application's speed and responsiveness is essential to avoid slow loading times and delays during transactions.



ANALOGUES



6/21



REQUIREMENTS FOR THE DEVELOPED SOFTWARE



Functional:

- User information
- Product Listings and Catalog Management
- Shopping Cart and Order Management
- Payment Integration
- Inventory Management



REQUIREMENTS FOR THE DEVELOPED SOFTWARE



Non-functional:

- User Reviews and Ratings
- Security
- Responsive Design
- Analytics and Reporting
- Scalability and Performance



REQUIREMENTS FOR THE DEVELOPED SOFTWARE



Non-functional:

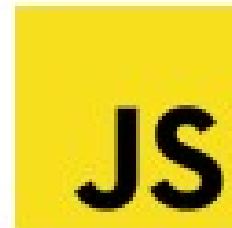
- User Reviews and Ratings
- Security
- Responsive Design
- Analytics and Reporting
- Scalability and Performance



SELECTION OF DEVELOPMENT TECHNOLOGIES



Backend



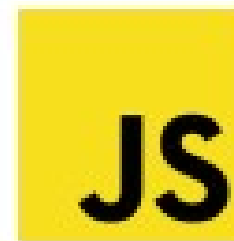
9/21



SELECTION OF DEVELOPMENT TECHNOLOGIES



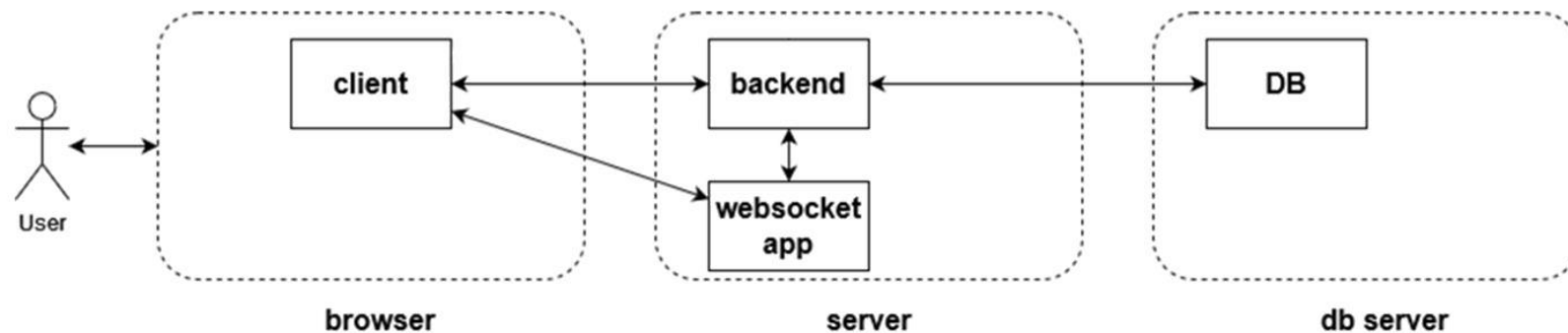
Frontend



10/21

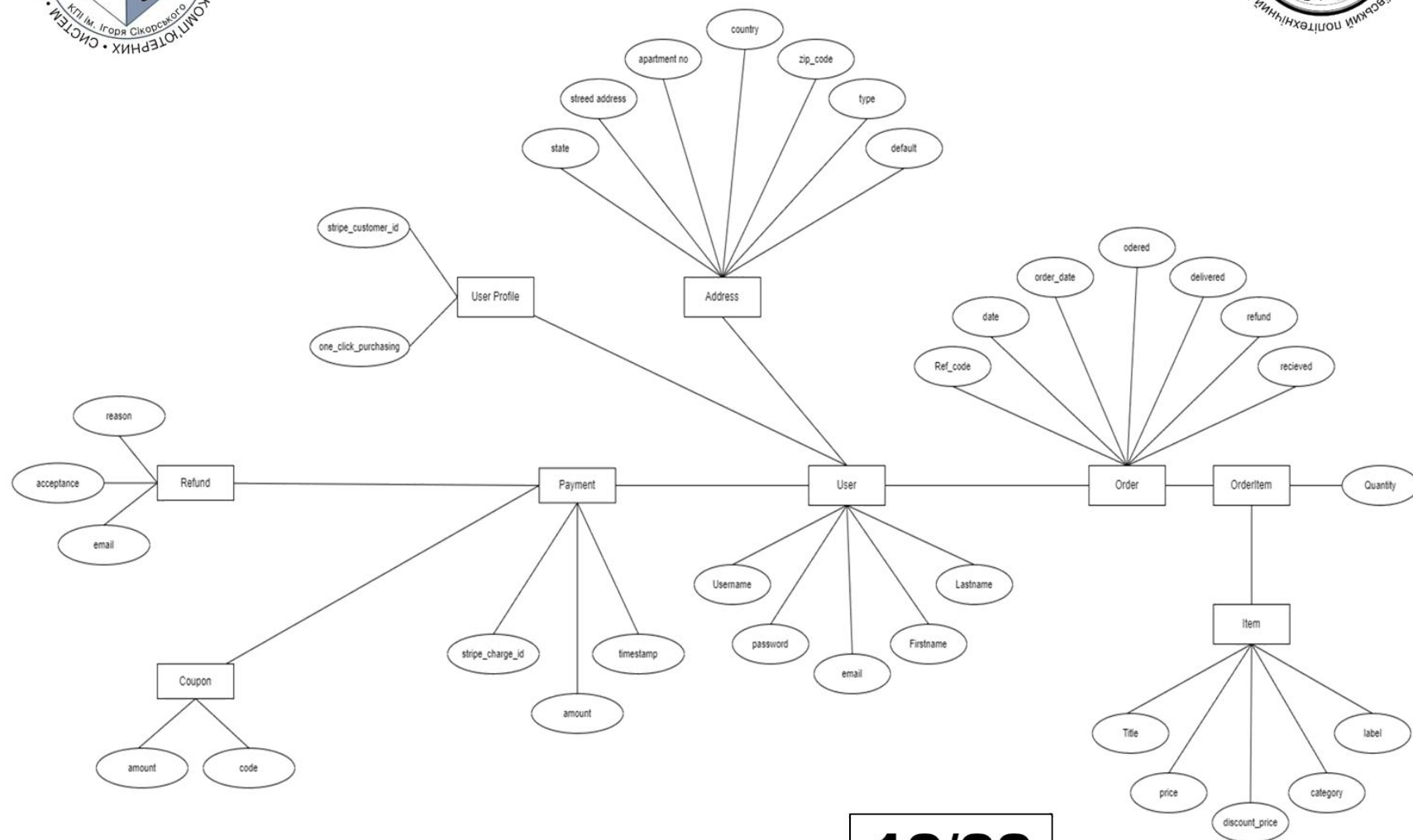


SOFTWARE ARCHITECTURE





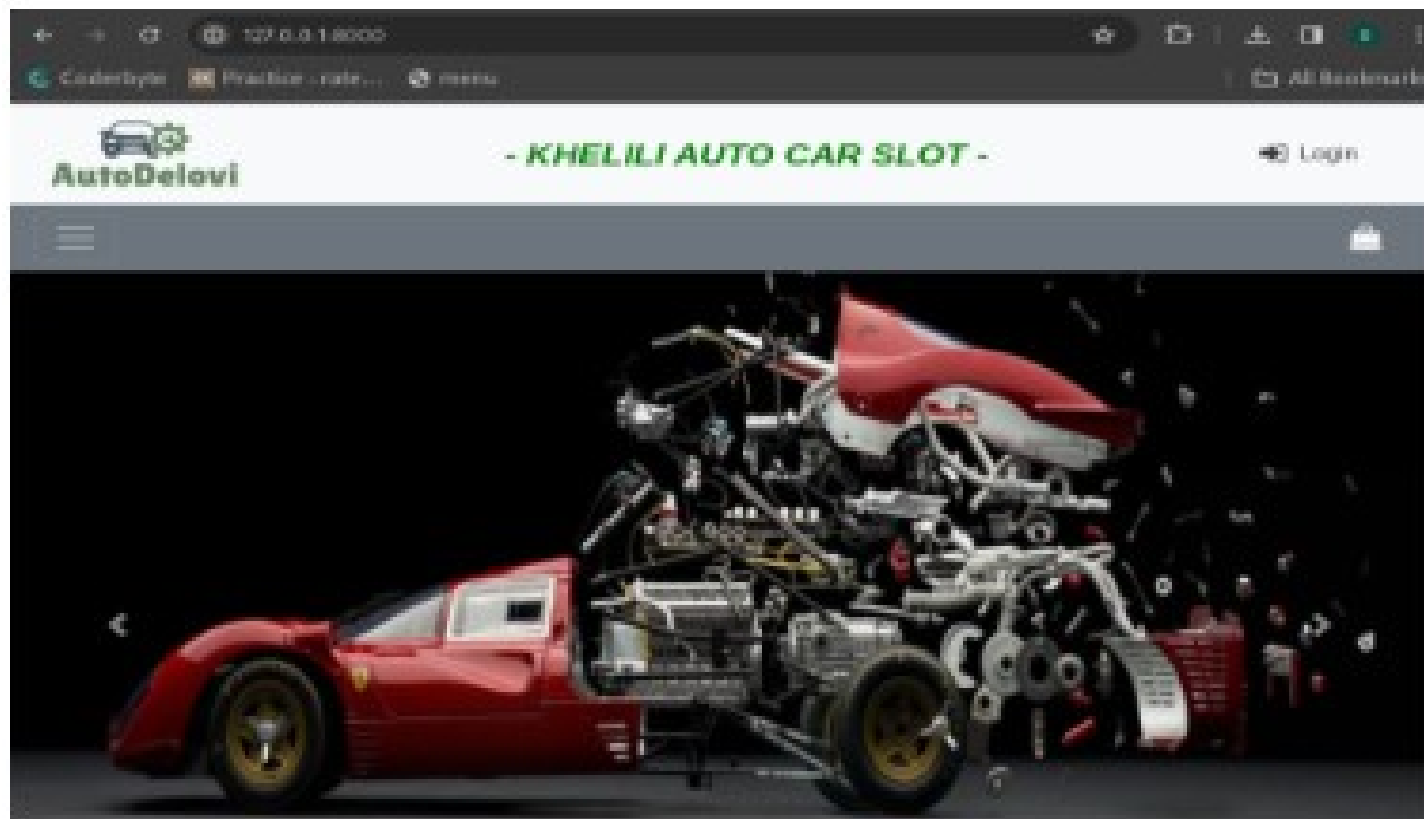
STRUCTURE OF ENTITY-RELATIONSHIP DIAGRAM



12/22



USER LANDING PAGE

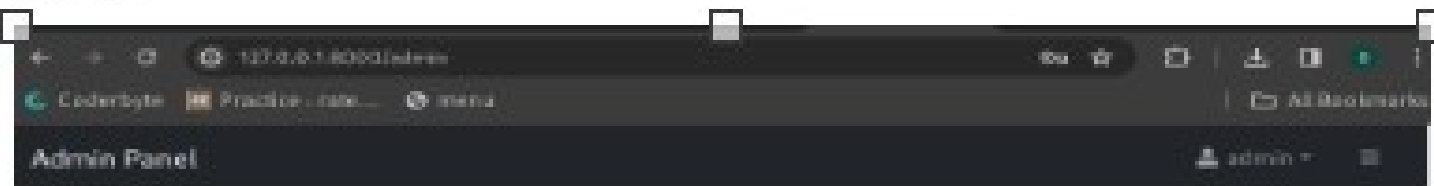


Carousel

13/21



ADMIN SEARCH PAGE



Purchase table

All Purchases

10 entries per page

Search...

Search within table

First Name	E-mail	Index	Date	Purchase Info	Delete Purchase
goodluck	goodluck@gmail.com	1	2024-03-30 23:39:37		
goodluck	goodluck@gmail.com	1	2024-03-30 12:01:09		
Armand Muller	samanta.cremins@example.org	3	2024-03-30 19:01:09		
Emanuel Bosco MD	joshie.dane@example.org	4	2024-03-30 12:01:09		
admin	admin@gmail.com	5	2024-03-30		

Product cards

14/21



CHECKOUT FORM

User information (first name, last name , address)



All Carts Details

Name of Product	Name of Buyer	Price	Quantity	Product Quantity	Destroy cart
ut	goodluck	10.32	1	<input type="text"/>	<input type="text"/>
son	goodluck	23.2	1	<input type="text"/>	<input type="text"/>

Total: \$33.52

Purchase

Products Click to continue purchase

127.0.0.1:8000/product/purchase/complete14

15/21



PURCHASE FORM

127.0.0.1:8000/product/purchaseform94

Coderbyte Practice - rate... menu All Bookmarks

 - **KHELILI AUTO CAR SLOT** - Log Out

Menu Shopping Cart 2

Fill Purchase Form

<input type="text" value="samuel"/>	<input type="text" value="James"/>
<input type="text" value="no 12 od"/>	<input type="text" value="no 12 oof"/>
<input type="text" value="call"/>	<input type="text" value="09393..."/>
<input type="text" value="samuel@gmail.com"/>	<input type="text" value="902322132"/>
<input type="button" value="Submit !!"/>	



SUCCESSFUL PAYMENT PAGE



Fill Purchase Form

<input type="text" value="First Name.."/>	<input type="text" value="Last Name.."/>
<input type="text" value="address one"/>	<input type="text" value="second address invalid .."/>
<input type="text" value="City.."/>	<input type="text" value="09993.."/>
<input type="text" value="Email.."/>	<input type="text" value="Phone Number.."/>

17/21



ADVANTAGES OF THE DEVELOPED SOFTWARE



- The developed software offers several advantages over traditional e-commerce web-applications. One of the main advantages is the intuitive navigation and search features, which allow customers to quickly find the products they need and complete their purchases with minimal hassle.
- The software also offers a seamless shopping experience across all devices, including desktops, laptops, tablets, and smartphones. Additionally, the secure payment processing ensures that customer information is protected at all times.



WAYS OF FURTHER DEVELOPMENT OF THE PROJECT

- Improve the user interface
- Add new features
- Integrate commerce with social media
- Enhance the search functionality
- Implement a loyalty program
- Optimize performance



CONCLUSIONS

The web application for the electronic commerce platform was created with a user-friendly interface that enables seamless online shopping experiences for customers and facilitates effective management of the platform for business owners. Throughout the project, a combination of HTML, CSS, and JavaScript was used for frontend development, while Python and SQLite were employed for backend implementation.

The choice of HTML, CSS, and JavaScript for frontend development proved to be effective in creating a user-friendly and visually appealing interface. These technologies allowed for the seamless integration of various design elements and the implementation of interactive features, enhancing the overall user experience.

On the backend, Laravel and SQLite were selected as the primary technologies. PHP provided a solid foundation for handling the application's business logic and server-side operations, while SQLite proved suitable for managing and storing the application's data.

The successful integration of these technologies resulted in a well-rounded web application for the electronic commerce platform. The application offers essential features such as product catalog management, shopping cart functionality, payment integration, and order processing. These features collectively contribute to a seamless and convenient online shopping experience for customers.



Thank you for attention!

Appendix 4

Program listing

Base file (.env)

```
APP_NAME=Autodelovi

APP_ENV=local
APP_KEY=base64:VE3DJ8QuJP0vgHzZ64D6/jnV+JsK/0KKKPHnvBLlhNY=
APP_DEBUG=true
APP_URL=http://localhost

LOG_CHANNEL=stack
LOG_LEVEL=debug

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=autodelovi
DB_USERNAME=root
DB_PASSWORD=

BROADCAST_DRIVER=log
CACHE_DRIVER=file
FILESYSTEM_DRIVER=local
QUEUE_CONNECTION=sync
SESSION_DRIVER=file
SESSION_LIFETIME=120

MEMCACHED_HOST=127.0.0.1

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

MAIL_MAILER=smtp
MAIL_HOST=mailhog
MAIL_PORT=1025
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS=null
MAIL_FROM_NAME="${APP_NAME}"

AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_DEFAULT_REGION=us-east-1
AWS_BUCKET=
AWS_USE_PATH_STYLE_ENDPOINT=false

PUSHER_APP_ID=
PUSHER_APP_KEY=
PUSHER_APP_SECRET=
PUSHER_APP_CLUSTER=mt1

MIX_PUSHER_APP_KEY="${PUSHER_APP_KEY}"
MIX_PUSHER_APP_CLUSTER="${PUSHER_APP_CLUSTER}"
```

basefile (compose.json)

```
{
  "name": "laravel/laravel",
  "type": "project",
  "description": "The Laravel Framework.",
  "keywords": ["framework", "laravel"],
  "license": "MIT",
  "require": {
    "php": "^7.3|^8.0",
    "fruitcake/laravel-cors": "^2.0",
    "guzzlehttp/guzzle": "^7.0.1",
    "laravel/framework": "^8.54",
    "laravel/sanctum": "^2.11",
    "laravel/tinker": "^2.5",
    "laravel/ui": "^3.3",
    "willvincent/laravel-rateable": "^2.3"
  },
  "require-dev": {
    "facade/ignition": "^2.5",
    "fakerphp/faker": "^1.23",
    "laravel/sail": "^1.0.1",
    "mockery/mockery": "^1.4.2",
    "nunomaduro/collision": "^5.0",
    "phpunit/phpunit": "^9.3.3"
  },
  "autoload": {
    "psr-4": {
      "App\\": "app/",
      "Database\\Factories\\": "database/factories/",
      "Database\\Seeders\\": "database/seeders/"
    }
  },
  "autoload-dev": {
    "psr-4": {
      "Tests\\": "tests/"
    }
  },
  "scripts": {
    "post-autoload-dump": [
      "Illuminate\\Foundation\\ComposerScripts::postAutoloadDump",
      "@php artisan package:discover --ansi"
    ],
    "post-update-cmd": [
      "@php artisan vendor:publish --tag=laravel-assets --ansi"
    ],
    "post-root-package-install": [
      "@php -r \"file_exists('.env') || copy('.env.example', '.env');\""
    ],
    "post-create-project-cmd": [
      "@php artisan key:generate --ansi"
    ]
  },
  "extra": {
    "laravel": {
      "dont-discover": []
    }
  },
  "config": {
    "optimize-autoloader": true,
    "preferred-install": "dist",
    "sort-packages": true
  }
}
```

```

    },
    "minimum-stability": "dev",
    "prefer-stable": true
}

```

Base File(package.json)

```

{
  "private": true,
  "scripts": {
    "dev": "npm run development",
    "development": "mix",
    "watch": "mix watch",
    "watch-poll": "mix watch -- --watch-options-poll=1000",
    "hot": "mix watch --hot",
    "prod": "npm run production",
    "production": "mix --production"
  },
  "devDependencies": {
    "axios": "^0.21",
    "bootstrap": "^4.6.0",
    "jquery": "^3.6",
    "laravel-mix": "^6.0.6",
    "lodash": "^4.17.19",
    "popper.js": "^1.16.1",
    "postcss": "^8.1.14",
    "resolve-url-loader": "^4.0.0",
    "sass": "^1.32.11",
    "sass-loader": "^11.0.1"
  }
}

```

Admin Controller

```

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Purchase;

class AdminController extends Controller
{
    public function index(Purchase $purchase)
    {
        return view('admin.index', compact('purchase'));
    }
}

```

Brand Controller

```

<?php

namespace App\Http\Controllers;

use App\Models\Category;
use Illuminate\Http\Request;
use App\Models\Brand;
use Illuminate\Support\Facades\DB;

```

```

class BrandController extends Controller
{
    public function index(Brand $brands)
    {
        $brands = Brand::all();
        $categories = Category::all();
        return view('brand.index', compact('brands', 'categories'));
    }

    public function create()
    {
        return view('brand.create');
    }

    public function store(Request $request)
    {
        $request->validate([
            'name' => 'required',
            'image' => 'required'
        ]);

        $data = $request->all();

        $imageName = time() . '.' . $request->image->extension();
        $request->image->move(public_path('/storage/app/public/uploads'));
        $data['image'] = $imageName;

        Brand::create($data);

        return redirect()->back()->with(['message' => 'Brand Created Successfully ', 'alert' => 'alert-success']);
    }

    public function show()
    {
        $brands = Brand::all();
        return view('brand.show', compact('brands', ));
    }

    public function destroy(Brand $brand)
    {
        $brand->delete();
        return redirect()->back()->with(['message' => 'Brand Deleted Successfully !!', 'alert' => 'alert-success']);
    }
}

```

Cart Controller

```
<?php
```

```
namespace App\Http\Controllers;
```

```

use Illuminate\Http\Request;
use App\Models\Product;
use App\Models\Cart;
use App\Models\Category;
use Illuminate\Support\Facades\Auth;

```

```

class CartController extends Controller
{
    /**
     * Display a listing of the resource.
     */
}

```

```

* @return \Illuminate\Http\Response
*/
public function index()
{
    $categories = Category::all();
    $carts = Cart::with(['user', 'product'])->get();
    return view('cart.index', compact('carts', 'categories'));
}

```

```

/**
 * Show the form for creating a new resource.
 *
 * @return \Illuminate\Http\Response
 */
public function create()
{
    //
}

```

```

/**
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    try {
        // Create a new cart item for the authenticated user
        auth()->user()->cart()->create($request->all());
    }
}

```

```

// Flash a success message
session()->flash('success', 'Product added to cart successfully.');
```

```

    } catch (\Exception $e) {
        // If an error occurs, flash an error message
        session()->flash('error', 'Error occurred while adding product to cart.');
```

```

    }
}

```

```

// Redirect back to the previous page
return redirect()->back();
}

```

```

/**
 * Display the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function show($id)
{
    //
}

```

```

/**
 * Show the form for editing the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function edit($id)
{
    //
}

```

```
{  
//  
}
```

```
/**  
 * Update the specified resource in storage.  
 */  
@param \Illuminate\Http\Request $request  
@param int $id  
@return \Illuminate\Http\Response  
 */  
public function update(Request $request, Cart $cart)  
{  
    $request->merge(['total' => $cart->product->price * $request->get('quantity')]);  
    $cart->update($request->all());  
    return redirect()->back();  
}
```

```
/**  
 * Remove the specified resource from storage.  
 */  
@param int $id  
@return \Illuminate\Http\Response  
 */  
public function destroy(Cart $cart)  
{  
    $cart->delete();  
    return redirect()->back()->with(['message' => 'Uspesno ste uklonili produkt!', 'alert' => 'alert-success']);  
}
```

Category Controllers

```
<?php
```

```
namespace App\Http\Controllers;
```

```
use Illuminate\Http\Request;  
use App\Models\Category;  
use Illuminate\Support\Facades\DB;
```

```
class CategoryController extends Controller  
{  
    public function index()  
    {  
    }  
}
```

```
public function create()  
{  
    return view('categories.create');  
}
```

```
public function store(Request $request)  
{  
    $request->validate([  
        'name' => 'required',  
        'image' => ['required', 'image']  
    ]);  
}
```

```
$data = $request->all();
```

```

$imageName = time() . '.' . $request->image->extension();
$request->image->move(public_path('/storage/app/public/uploads'), $imageName);

$data['image'] = $imageName;

Category::create($data);

return redirect()->back()->with(['message' => 'Category created successfully!', 'alert' => 'alert-success']);
}

public function show()
{
    $categories = Category::all();
    return view('categories.show', compact('categories'));
}

public function destroy(Category $category)
{
    $category->delete();
    return redirect()->back()->with(['message' => 'Deleted Successfully ', 'alert' => 'alert-success']);
}
}

```

Comment Controller

```

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use \Illuminate\Support\Facades\Auth;
use App\Models\Comment;
use App\Models\Product;

class CommentController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $commentns = Comment::latest();
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param \Illuminate\Http\Request $request
     * @return \Illuminate\Http\Response
     */
}

```



```

*/
public function store(Request $request)
{
    $product = Product::findOrFail($request->product_id);
    Comment::create([
        'comment' => $request->comment,
        'user_id' => Auth::id(),
        'product_id' => $product->id,
    ]);

    return redirect()->back()->with(['message' => 'Uspesno ste narucili produkt!', 'alert' => 'alert-success']);
}

```

```

/**
 * Display the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function show($id)
{
    //
}

```

```

/**
 * Show the form for editing the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function edit($id)
{
    //
}

```

```

/**
 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
    //
}

```

```

/**
 * Remove the specified resource from storage.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function destroy(Comment $comment)
{
    $comment->delete();
    return redirect()->back();
}

```

Home Controller

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Product;
use Illuminate\Support\Facades\DB;
use App\Models\Category;
use App\Models\Comment;
use App\Models\SoldProduct;

class HomeController extends Controller
{
    public function index(Product $product, SoldProduct $product id)
    {
        $comments = Comment::latest()->paginate(3);
        $categories = Category::all();
        $products = DB::table('products')->orderBy('created_at', 'DESC')->paginate(4);

        $popularProducts = Product::get()->sortByDesc(
            function ($p) {
                return $p->solds->sum('qty');
            }
        )->take(4);

        return view('home.index', compact('products', 'categories', 'comments', 'popularProducts', 'product id'));
    }
}
```

Product Controller

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Product;
use App\Models\Category;
use App\Models\Brand;
use Illuminate\Support\Facades\DB;
use App\Models\User;
use App\Models\Rating;
use App\Models\Comment;
use App\Models\Purchase;
use Illuminate\Support\Facades\Auth;

class ProductController extends Controller
{
    public function index(Product $product, Category $category, Comment $comment, Brand $brand)
    {
        $categories = Category::all();
        $vehicle_type = ['Motor', 'Auto', 'Terenac', 'Kamion', 'Autobus'];
        $recomendedProducts = DB::table('products')->where('brand_id', $product->brand->id)->take(4)->get();

        return view('product.index', compact('product', 'brand', 'category', 'categories', 'comment',
            'recomendedProducts', 'vehicle_type'));
    }
}
```

```
public function show()
{
    $products = Product::all();

    return view('product.show', compact('products'));
}
```

```
public function create()
{
    $categories = Category::all();
    $brands = Brand::all();
    $vehicle_type = ['Motor', 'Auto', 'Terenac', 'Kamion', 'Autobus'];

    return view('product.create', compact('categories', 'brands', 'vehicle_type'));
}
```

```
public function store(Request $request)
{
    $request->validate([
        'name' => 'required',
        'description' => 'required',
        'price' => 'required',
        'brand_id' => 'required',
        'category_id' => 'required',
        'vehicle_type' => "",
        'image' => ['required', 'image'],
    ]);
```

```
$data = $request->all();
$imageName = time() . '.' . $request->image->extension();
$request->image->move(public_path('/storage/app/public/uploads'), $imageName);
$data['image'] = $imageName;
```

```
Product::create($data);
return redirect()->back()->with(['message' => 'Product was created successfully !', 'alert' => 'alert-success']);
}
```

```
public function edit($id)
{
    $product = DB::table('products')->where('id', $id)->first();
    $categories = Category::all();
    $brands = Brand::all();
    $vehicle_type = ['Motor', 'Auto', 'Terenac', 'Kamion', 'Autobus'];

    return view('product.edit', compact('product', 'categories', 'brands', 'vehicle_type'));
}
```

```
public function update(Request $request, Product $product)
{
    $this->authorize('update', $product->id);
    $request->validate([
        'name' => 'required',
        'description' => 'required',
        'price' => 'required',
    ]);
```

```
return view('product.update')->with(['message' => 'Uspesno ste azurilali produkt!', 'alert' => 'alert-success']);
}
```

```
public function destroy(Product $product)
{
}
```

```
$product->delete();
return redirect()->back()->with(['message' => 'Uspesno ste obrisali product!', 'alert' => 'alert-success']);
}
```

```
public function addToCart($id)
{
    $product = Product::findOrFail($id);
    $cart = session()->get('cart', []);
```

```
if (isset($cart[$id])) {
    $cart[$id]['quantity']++;
} else {
    $cart[$id] = [
        "name" => $product->name,
        "quantity" => 1,
        "price" => $product->price,
        "image" => $product->image
    ];
}
session()->put('cart', $cart);
```

```
return redirect()->back()->with('success', 'Produkt je uspesno dodat u korpi');
}
```

```
public function allProducts()
{
    $products = Product::latest()->get();
    $categories = Category::all();
```

```
return view('product.products', compact('products', 'categories'));
}
```

```
public function products_by_category(Category $category, Product $product)
{
    $categories = Category::all();
    $productsByCategory = Product::where('category_id', $category->id)->latest()->get();
    return view('product.products_by_category', compact('productsByCategory', 'categories'));
}
```

```
public function products_by_brand(Brand $brand, Product $product)
{
    $categories = Category::all();
    $productsByBrand = Product::where('brand_id', $brand->id)->latest()->get();
    return view('product.products_by_brands', compact('categories', 'productsByBrand', 'brand'));
}
```

```
public function products_by_search()
{
    $categories = Category::all();
```

```
$search_text = $_GET['query'];
$products = Product::where('name', 'LIKE', '%' . $search_text . '%')->with('category')->get();
return view('product.productsBySearch', compact('products', 'categories'));
}
```

```
public function products_by_vehicle_type(Product $product)
{
    $categories = Category::all();
    $productsByVehicleType = Product::where('vehicle_type', $product->vehicle_type)->latest()->get();
```

```
return view('product.products_by_vehicle_type', compact('productsByVehicleType', 'categories'));
}
```

```
public function allProductsAdmin()
{
    $products = Product::all();
    return view('product.allProductAdmin', compact('products'));
}
```

Purchase Controller

```
<?php
```

```
namespace App\Http\Controllers;
```

```
use Illuminate\Http\Request;
use App\Models\Product;
use App\Models\Category;
use App\Models\Purchase;
use App\Models\Cart;
use App\Models\User;
use Illuminate\Support\Facades\DB;
```

```
class PurchaseController extends Controller
```

```
{
    public function index()
    {
```

```
}
```

```
public function show()
{
    $purchases = Purchase::latest()->paginate(10);
    return view('purchase.show', compact('purchases'));
}
```

```
public function create(Product $product)
{
    $categories = Category::all();
    return view('purchase.index', compact('product', 'categories'));
}
```

```
public function store(Request $request)
{
    $request->validate([
        'user_id' => 'required',
        'first_name' => 'required',
        'last_name' => 'required',
        'address1' => 'required',
        'address2' => '',
        'city' => 'required',
        'zip' => 'required',
        'phone' => 'required',
        'email' => 'required',
    ]);
```

```
$purchase = auth()->user()->purchase()->create($request->all());
```

```
foreach (auth()->user()->cart as $product) {  
$purchase->products()->create([  
'product_id' => $product->product_id,  
'quantity' => $product->quantity,  
'price' => $product->product->price,  
'total_amount' => $product->total  
]);  
$product->delete();  
}
```

```
return redirect()->back()->with(['message' => 'Purchase completed Successfully !', 'alert' => 'alert-success']);  
}  
public function purchaseInfo(Purchase $purchase, User $user)  
{  
return view('purchase.purchase_info', compact('purchase', 'user'));  
}
```

```
public function destroy(Purchase $purchase)  
{  
$purchase->delete();  
return redirect()->back()->with(['message' => 'Purchase Deleted Successfully ', 'alert' => 'alert-success']);  
}  
}
```