UDC 004.75 **doi: 10.26906/SUNZ.2023.2.108-112**

Maksym Demydenko, Anton Fesenko

National University "Yuri Kondratyuk Poltava Polytechnic", Poltava, Ukraine

DEVELOPMENT OF A WEB API FOR AN ONLINE CAR PARTS STORE USING N-TIER ARCHITECTURE

Abstract. Modern business environment challenges, market globalization, and increasing consumer demands have significantly influenced the development of e-commerce. Digitalization of buying and selling processes has become especially important, encompassing all industries, including the automotive sector. Despite significant progress in web technology development, the issue of ensuring an efficient software structure for online stores remains relevant. Most web applications are built on a monolithic architecture, which complicates their modernization, scaling, and integration with new services. This limits business growth and reduces its efficiency. The aim of this work is to develop a Web API for an online car parts store using an N-Tier architecture. This approach ensures a clear separation of the project into logical layers, simplifying development, testing, and maintenance while enhancing the system's flexibility and scalability.

Keywords: E-commerce, web development, multi-tier architecture.

Introduction

Modern business environment challenges, market globalization, and increasing consumer demands have significantly influenced the development of e-commerce. The digitalization of buying and selling processes has become especially important, covering all industries, including the automotive sector. Despite significant progress in web technology development, the challenge of ensuring an efficient software structure for online stores remains relevant.

Relevance of the Topic. The development of e-commerce, particularly in the field of car parts sales, requires the implementation of new approaches to managing online stores. To ensure the efficient operation of such platforms, it is necessary to use flexible and scalable information systems capable of processing large volumes of data, supporting integration with various services, and adapting to rapidly changing market demands.

Many modern e-commerce platforms struggle with effective scalability and rapid adaptation to new technological trends due to the limitations of traditional architectural approaches.

A multi-tier architecture enables a clear separation of functional system components, enhancing its flexibility and scalability. Therefore, the use of such architectural approaches is relevant for creating high-performance online stores.

Analysis of Recent Research. Most publications dedicated to multi-tier (N-Tier) architecture examine its application in various software development domains, including e-commerce and enterprise systems. For instance, in study [1], the fundamental principles of N-Tier architecture, its components, and key operational aspects are discussed. The study explains how a multi-tier approach enhances scalability, simplifies system maintenance, and facilitates integration with various services.

In study [2] on the Stackify website, it is noted that N-Tier architecture differs from the MVC framework by incorporating an additional middle (business logic) layer, which streamlines communication between different system layers. The study also compares N-Tier architecture with other architectural approaches and provides real-world application examples.

An article [3] on Medium.com presents an approach to organizing multi-tier solutions to improve application performance while ensuring flexibility and adaptability to changes.

The objective of this work is to develop a Web API for an online car parts store using an N-Tier architecture.

Features of Multi-Tier Architecture in E-Commerce Application Development

The characteristics of information systems are determined not only by their functional capabilities and software tools but also by their architecture. An alternative to monolithic systems is the multi-tier (N-Tier) architecture, which divides the system into separate logical layers. This approach provides several advantages, including:

1. Modularity – Each layer is an independent project (e.g., a DLL or JAR file) that can be reused in other projects.
2. Isolated Scalability – Expanding or modifying one layer does not affect the operation of other layers.
3. Development Efficiency – Different development teams can work independently on separate layers, improving productivity.
4. Testability – Each layer can be tested individually, making it easier to detect errors and simplifying the testing of new features.

Limitations of Using Multi-Tier Architecture:

1. Cost – Multi-tier architecture requires more memory and resources to manage communication between layers.
2. Security Concerns – Strong security measures must be implemented to ensure that higher layers do not directly access lower layers but interact only through the business logic layer.
3. Reduced Performance – Additional delays in data exchange between layers may impact the overall system performance [2].

The project implementation diagram is shown in Fig. 1. The overall project consists of a website (front-end) and a server-side component (back-end), which

functions as a REST API utilizing a multi-tier architecture. It processes client requests (from the website or direct API calls) over the HTTP(HTTPS) protocol in JSON format and returns responses in the same format.

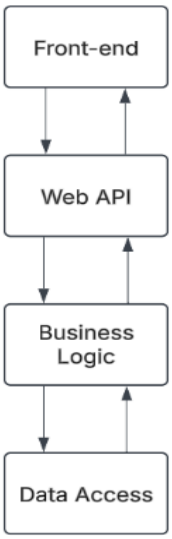

Fig. 1. Project Architecture

The system receives a request from the website and processes it down the schema to the API layer. From there, the request moves to the Business Logic (BL) layer, which checks the user's permissions to execute the request, transforms the data from the Data Transfer Object (DTO) into a fully-fledged object that represents an entity from the database, and performs any necessary calculations for required fields (for example, calculating the final price of a product based on the initial price and applying a discount).

After processing, the data is passed to the Data Access Layer (DAL), where it is stored, modified, deleted, or retrieved from the database [5].

After passing through all the layers, the request returns from the DAL layer to the BL layer, where it is transformed into the object required by the client. The processed data is then passed to the API layer, which returns the data in JSON format to the client.

The project includes the following list of data access services, as shown in Fig 2:

1. IAuthDAL – responsible for searching for users in the database by email, identifier, and recovery token, as well as providing functionality for creating and updating user records.
2. ICartItemDAL – responsible for storing items in the shopping cart in the database, as well as deleting and updating them.
3. IOnlinePaymentDAL – stores information about users' online payments and provides functionality for searching and updating records.
4. IOrderDAL – responsible for storing and updating orders.
5. IProductDAL – stores and updates products in the database. The deletion method is absent because products are not directly deleted from the database; instead, a "soft delete" technique is used. The product is marked as inactive for orders rather than being deleted.
6. IProductDetailsDAL – stores product details in the database. It serves as a base interface for services like IBrandDAL (product brands), ICategoryDAL (product categories), and ISpecialTagDAL (special tags), as they share similar logic, differing only in types.
7. IRefreshTokenDAL – stores and updates refresh tokens in the database.
8. IShoppingCartDAL – responsible for storing and updating user shopping carts.

The general database schema with which this level interacts is shown in Fig. 3. It represents the structure and relationships between various entities within the system, including users, products, orders, payments, shopping carts, and other relevant data components.

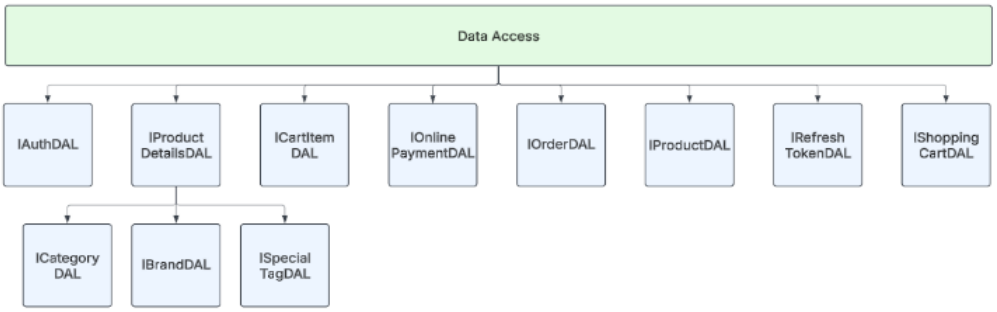

Fig. 2. Data Access Level

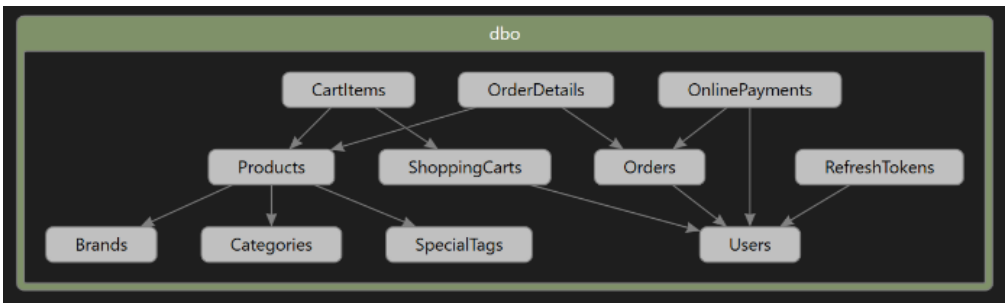

Fig. 3. Database Schema

The Business Logic (BL) layer consists of 13 services, which inherit from the interfaces shown in Fig. 4. These services are divided into primary and auxiliary categories. Primary services are designed for use by the API layer, while auxiliary services are intended for internal use within the business logic layer.

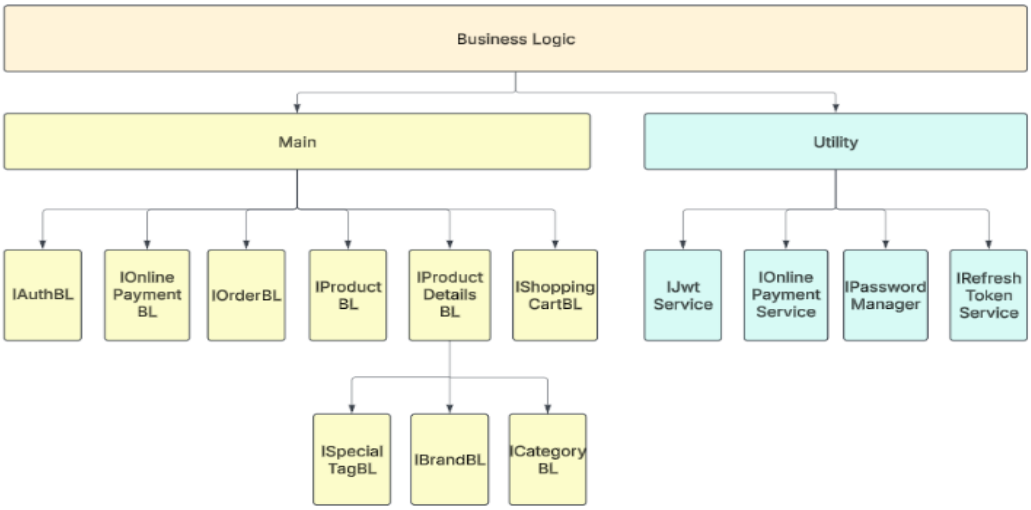

Fig. 4. Business Logic Level

The primary services include:

1. IAuthBL – responsible for user authentication and registration, role changes, and the generation of refresh tokens and JWT access tokens.

2. IOnlinePaymentBL – manages online order payments (initializing payments and canceling them in case of order cancellation).

3. IOrderBL – responsible for creating and processing orders.

4. IProductBL – responsible for creating and editing products.

5. IProductDetailsBL – Responsible for creating and editing product details. Similar to the business logic layer, it includes three more services under it: ISpecialTagBL, IBrandBL, and ICategoryBL.

IShoppingCartBL – responsible for adding and removing items from the user's shopping cart.

The auxiliary services include the following:

1. IJwtService – a service for generating access tokens in JWT format. It is called by the authentication service to generate access tokens when users log in. The current implementation of this service sets the token's expiration time to 15 minutes.

2. IOnlinePaymentService – a service for initializing online payments and canceling them. It is invoked by the main OnlinePaymentBL service. The difference between the two is that IOnlinePaymentService serves as a contract for a specific online payment service, such as Stripe, Portmone, or Privat24. OnlinePaymentBL is unaware of the specific implementation and simply interacts with the interface.

3. IPasswordManager – a password manager responsible for hashing passwords and comparing them during authentication, password changes, or user registration.

4. IRefreshTokenService – responsible for generating refresh tokens. The current implementation specifies that the refresh token's lifespan is 2 weeks.

When a user logs into the system, two tokens are issued: a JWT token with a lifespan of 15 minutes and an access token, which is stored in an HTTP-only cookie and has a lifespan of 2 weeks.

JSON Web Token (JWT) is a compact, secure format for transmitting information between parties in the form of a JSON object. It is widely used for authentication and authorization in web applications. The token consists of three parts: the header, the payload, and the signature [4]. The header defines the token type and the signing algorithm, the payload contains the data (claims), and the signature guarantees the integrity of the token, preventing it from being altered without authorization. JWT can be signed using either a symmetric or asymmetric key, making it reliable for verifying the authenticity of the data. Due to its compact nature, the token can easily be transmitted through HTTP headers or URLs. The primary advantage of JWT is its ability to work securely in stateless systems, where the server does not store session information because all the necessary information is encoded directly within the token.

The refresh token allows for the automatic renewal of the access token when its expiration time is reached, without requiring the user to log in again. This token has a longer lifespan and is stored in an HTTP-only cookie, providing added security and convenience by eliminating the need to repeatedly enter the password.

The refresh token is a unique string of random characters, 32 bytes in length. It is stored in the database, which also associates the token with a specific user and defines its expiration time. After being stored in the database, the token is sent to the API via the Business Logic (BL) layer, which then adds the token to the response in the HTTP-only cookie. This process ensures that the user's session remains valid without requiring constant re-authentication.

HTTP-only cookies are accessible only by the server and cannot be read by JavaScript, which reduces

the risk of XSS (Cross-Site Scripting) attacks. This feature enhances security by preventing malicious scripts from accessing sensitive information stored in the cookies. Additionally, it allows tokens to be automatically sent with each request without the need for manually adding them to the headers, simplifying authentication while maintaining security.

The API layer handles the reception and sending of processed requests to the client. It consists of 8 controllers in total:

1. AuthController – responsible for user registration, authentication, role changes, password changes, and access token renewal. The overall workflow and dependencies of this functionality are shown in Fig. 5.

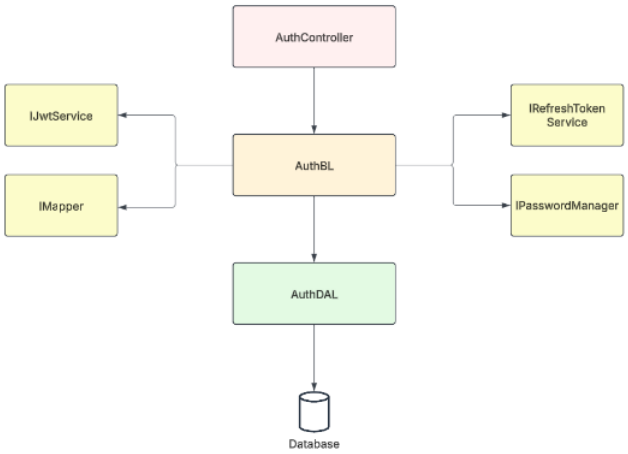

Fig. 5. Auth Component Functionality

2. BrandController – responsible for retrieving, creating, and deleting product brands. The functionality diagram is shown in Fig. 6.

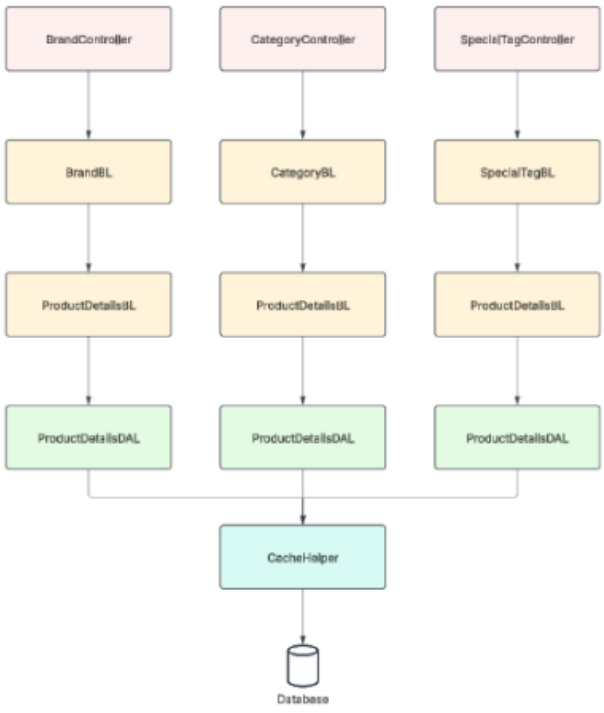

Fig. 6. Functionality of the Brand, Category and Special Tag Components

3. CategoryController – responsible for retrieving, creating, and deleting product categories. The functionality diagram is shown in Fig.6.

4. SpecialTagController – responsible for accessing, creating, and deleting special tags for products. The functionality diagram is shown in Fig. 6.

5. OnlinePaymentController – responsible for retrieving, initializing, and canceling online payments. The functionality diagram is shown in Fig. 7.

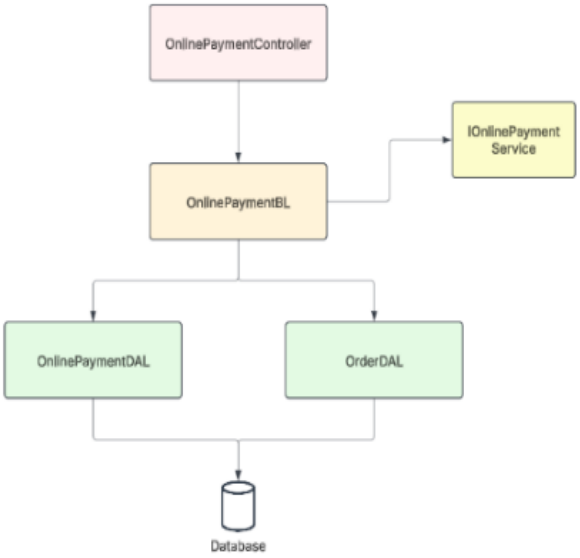

Fig. 7. Functionality of the Online Payment Component

6. OrderController – responsible for creating and updating orders. The functionality diagram is shown in Fig. 8.

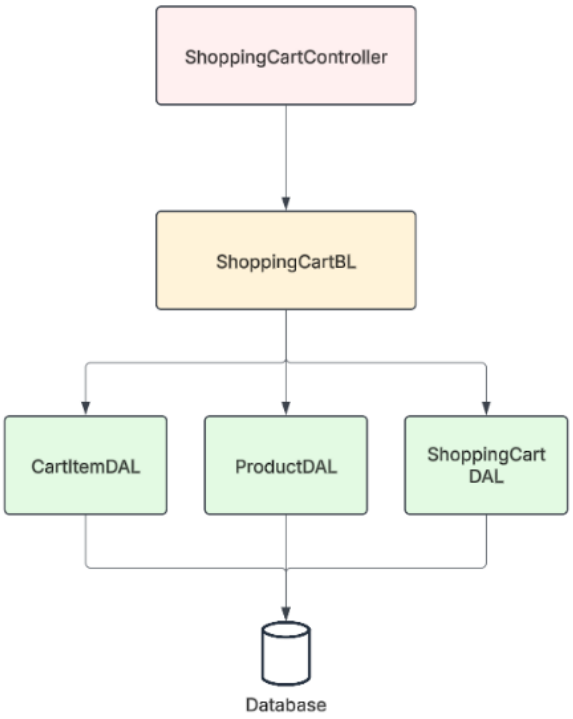

Fig. 8. Component for Order Processing

7. ProductsController – responsible for retrieving, creating, and updating products. The component is shown in Fig. 9.

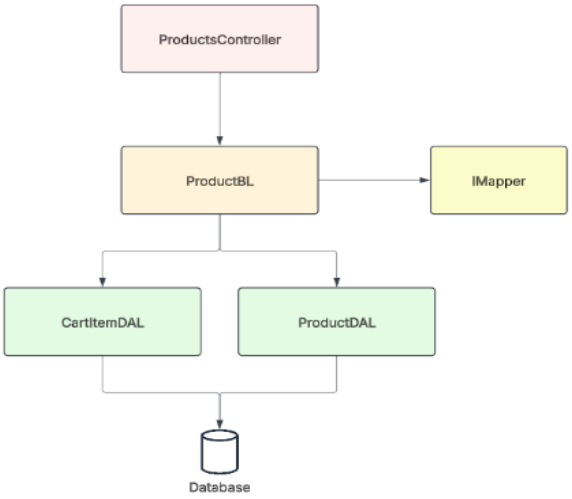

Fig. 9. Functionality of the Products Component

8. ShoppingCartController – responsible for accessing and updating the shopping cart. The component is shown in Fig. 10.

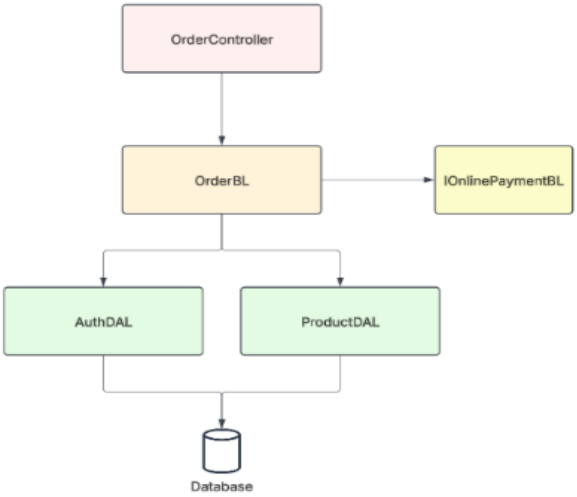

Fig. 10. Functionality of the Shopping Cart Component

Conclusion

The analysis of recent studies revealed that most publications on multi-tier (N-tier) architecture focus on its use in enterprise applications and business process management systems. Based on the findings, a prototype Web API for online stores selling auto parts has been developed using multi-tier architecture.

REFERENCES

1. Бойко Н. І. Еволюція побудови архітектур інформаційних систем. Перспективи розвитку “хмарної” архітектури». *Вісник Національного університету «Львівська політехніка», Серія: Інформаційні системи та мережі*. 2015. Вип. 832. С. 348–367. URL: https://science.lpnu.ua/sites/default/files/journal-paper/2018/jun/12945/24-348-367.pdf
2. What is N-Tier Architecture? How It Works, Examples, Tutorials, and More. URL: https://stackify.com/n-tier-architecture
3. Understanding N-Tier Architecture: Building Robust and Scalable Applications URL: https://medium.com/@segekaratas/understanding-n-tier-architecture-building-robust-and-scalable-applications-62db30a40b5
4. What is JWT? URL: https://blog.postman.com/what-is-jwt/
5. Understanding Layers, Tiers, and N-Tier Architecture in Application Development: URL: https://dev.to/3bdelrahman/understanding-layers-tiers-and-n-tier-architecture-in-application-development-1hlb

Received (Надійшла) 11.03.2025
Accepted for publication (Прийнята до друку) 07.05.2025

ВІДОМОСТІ ПРО АВТОРІВ / ABOUT THE AUTHORS

Демиденко Максим Ігорович – старший викладач кафедри комп’ютерних та інформаційних технологій і систем, Національний університет «Полтавська політехніка імені Юрія Кондратюка», Полтава, Україна;
Maksym Demydenko - Senior Lecturer, Department of Computer and Information Technologies and Systems, National University «Yuri Kondratyuk Poltava Polytechnic», Poltava, Ukraine;
e-mail: maxx74@ukr.net; ORCID Author ID: https://orcid.org/0009-0001-1374-7379
Фесенко Антон Богданович – студент кафедри комп’ютерних та інформаційних технологій і систем, Національний університет «Полтавська політехніка імені Юрія Кондратюка», Полтава, Україна;
Anton Fesenko - student of the Department of Computer and Information Technologies and Systems, National University «Yuri Kondratyuk Poltava Polytechnic», Poltava, Ukraine;
e-mail: tohafesenkolovep@gmail.com

Створення WEB API для онлайн-мазагину автотоварів з використанням архітектури N-Tier

М. І. Демиденко, А. Б. Фесенко

Анотація. Сучасні виклики бізнес-середовища, глобалізація ринків та зростання потреб споживачів суттєво вплинули на розвиток електронної комерції. Особливо важливою стала цифровізація процесів купівлі-продажу, що охопила всі галузі, зокрема й автомобільну індустрію. Незважаючи на значний прогрес у розвитку веб-технологій, проблема забезпечення ефективної структури програмних рішень для онлайн-магазинів залишається актуальною. Більшість веб-додатків побудовані на монолітній архітектурі, що ускладнює їх модернізацію, масштабування та інтеграцію з новими сервісами. Це обмежує розвиток бізнесу та знижує його продуктивність. Метою даної роботи є створення Web API для онлайн-магазину автотоварів із використанням архітектури N-Tier. Такий підхід забезпечує чітке розділення проєкту на логічні рівні, що спрощує розробку, тестування й подальшу підтримку, а також підвищує гнучкість і масштабованість системи.

Ключові слова: електронна комерція, веб-розробка, багаторівнева архітектура.