

SerVICE

Name:Nicoara Tomas

Group: VI

Table of Contents

[Deliverable 1 3](#_Toc64843130)

[Project Specification 3](#_Toc64843131)

[Functional Requirements 3](#_Toc64843132)

[Use Case Model 3](#_Toc64843133)

[Use Cases Identification 3](#_Toc64843134)

[UML Use Case Diagrams 3](#_Toc64843135)

[Supplementary Specification 3](#_Toc64843136)

[Non-functional Requirements 3](#_Toc64843137)

[Design Constraints 3](#_Toc64843138)

[Glossary 3](#_Toc64843139)

[Deliverable 2 3](#_Toc64843140)

[Domain Model 3](#_Toc64843141)

[Architectural Design 4](#_Toc64843142)

[Conceptual Architecture 4](#_Toc64843143)

[Package Design 4](#_Toc64843144)

[Component and Deployment Diagram 4](#_Toc64843145)

[Deliverable 3 4](#_Toc64843146)

[Design Model 4](#_Toc64843147)

[Dynamic Behavior 4](#_Toc64843148)

[Class Diagram 4](#_Toc64843149)

[Data Model 4](#_Toc64843150)

[System Testing 4](#_Toc64843151)

[Future Improvements 4](#_Toc64843152)

[Conclusion 4](#_Toc64843153)

[Bibliography 4](#_Toc64843154)

# Deliverable 1

## Project Specification

This project is a web application that allows users to sell services. It consists of four main entities: User, Service, Category, and Order.

## Functional Requirements

1.User registration and login: Users must be able to register for an account with the web app using their email address and create a username and password. They should be able to log in and out of the app using their credentials.

2.Service creation and editing: Users should be able to create new services to sell on the web app, including adding a title, description, price, and category. They should also be able to edit and delete their existing services.

3.Service ordering and payment: Buyers should be able to browse and search for services on the web app, and place orders for the services they want to purchase. They should be able to pay for services using a secure payment gateway.

4.Service categorization and filtering: Services should be organized into categories, and buyers should be able to filter services by category when browsing and searching.

5.Order management: Sellers should be able to manage their orders, including viewing order details and marking orders as complete. Buyers should be able to view their order history and track the status of their current orders.

## Use Case Model 1

### Use Cases Identification

Use-Case: *Create Account*

Level: User Goal

Primary Actor: User

Main success scenario: The user creates an account by providing a valid email address, a unique username, and a password.

Extensions: If the email address or username is already in use, the system displays an error message.

Use-Case: *Create Service*

Level: User Goal

Primary Actor: User

Main success scenario: The user creates a service by providing a title, a description, a price, and one or more categories.

Extensions: If the user is not logged in, the system redirects them to the login page.

Use-Case: *Searching for Services*

Level: User Goal

Primary Actor: Buyer

Main success scenario:

The buyer enters a keyword or category to search for services.

The system displays a list of services matching the search criteria.

The buyer selects a service from the list.

The system displays detailed information about the service.

Extensions:

If no services match the search criteria, the system displays a message indicating that no results were found.

Use-Case: *Buying Service*

Level: User Goal

Primary Actor: Buyer

Main success scenario:

The buyer selects a service they wish to purchase.

The system displays the details of the selected service, including the price.

The buyer confirms that they wish to purchase the service.

The system prompts the buyer to enter their payment details.

The buyer enters their payment details and confirms the purchase.

The system confirms that the purchase was successful and updates the buyer's order history.

Extensions:

If the buyer's payment is declined, the system displays an error message and prompts the buyer to enter a different payment method.

If the service is no longer available (e.g. it has been deleted by the seller), the system displays an error message and prompts the buyer to select a different service.  
  
Use-Case: *Update Service*

Level: User Goal

Primary Actor: Service Provider

Main Success Scenario:

The Service Provider selects a Service they wish to update.

The Service Provider makes the desired updates to the Service's information (e.g., title, description, price, categories).

The Service Provider saves the changes.

The Service information is updated in the system and is now displayed correctly on the Service Provider's profile and on search results.

Extensions:

If the Service Provider makes a mistake while updating the Service information, they can choose to cancel the update without saving the changes.

Use-Case: *Delete Service*

Level: User Goal

Primary Actor: Service Provider

Main Success Scenario:

The Service Provider selects a Service they wish to delete.

The Service Provider confirms that they wish to delete the Service.

The Service information is removed from the system and is no longer displayed on the Service Provider's profile or on search results.

Extensions:

If the Service Provider changes their mind about deleting the Service, they can choose to cancel the deletion before confirming it.

Use-Case: *Delete Account*

Level: User Goal

Primary Actor: User

Main Success Scenario:

The User selects the "Delete Account" option.

The User is prompted to confirm that they wish to permanently delete their account.

If the User confirms, all of their information (including Services and Orders) is deleted from the system.

The User is logged out and redirected to the homepage.

Extensions:

If the User changes their mind about deleting their account, they can choose to cancel the deletion before confirming it.

If the User has pending orders or services, they will be unable to delete their account until those are completed or transferred to another user. In this case, they will receive an appropriate error message.

### UML Use Case Diagrams

Diagram

Description automatically generated

## Supplementary Specification

### Non-functional Requirements

1.Performance: The application should be able to handle a large number of users and services without significant delays or downtime. This is important because the success of the application depends on its ability to provide a fast and reliable service to its users.

2.Security: The application should be designed with security in mind, to protect user data and prevent unauthorized access or malicious attacks. This is important because users will be providing sensitive information such as their email address and payment details.

3.Usability: The application should be easy to use and navigate, with clear and intuitive interfaces. This is important because users will not use an application that is difficult or confusing to use.

4.Compatibility: The application should be compatible with a wide range of devices and browsers, to ensure that users can access it from anywhere. This is important because users may access the application from different devices and browsers.

### Design Constraints

1.ASP.NET Core 7: The web application must be built using ASP.NET Core 7 framework. This framework is a modern, cross-platform framework for building web applications and is suitable for this project's requirements.

2.Microsoft SQL Server Express: The database for the web application must use Microsoft SQL Server Express. This is a free and lightweight version of the popular SQL Server database management system and is suitable for this project's requirements.

3.Front-end Technologies: The front-end of the web application must be developed using Blazor, CSS, HTML, and Razor. Blazor is a modern and open-source web framework for building web apps using C# and HTML. CSS and HTML are industry-standard web technologies for styling and structuring web pages, respectively. Razor is a markup syntax used to create dynamic web content in .NET.

4.Coding standards: The code developed for this project must follow the established coding standards and best practices for the ASP.NET Core and Blazor frameworks. These standards will ensure that the code is maintainable, scalable, and easily understandable by other developers who may work on the project in the future.

5.Testing: The web application must be thoroughly tested to ensure that it is free from defects and meets the functional and non-functional requirements. The testing process should include unit testing, integration testing, and end-to-end testing.

6.Deployment: The web application must be deployed on a suitable hosting platform that supports ASP.NET Core 7 and Microsoft SQL Server Express. This may include cloud-based hosting platforms such as Microsoft Azure, Amazon Web Services, or Google Cloud Platform, or on-premises hosting solutions.

## Glossary

User: A person who uses the application to sell or order services.

Service: A task or job that a user offers to perform for a fee.

Category: A classification of services based on their type or subject.

Order: A request by a buyer to purchase a service from a seller.

## Deliverable 2

## Domain Model

Name: **BaseEntity**

Attributes: id | createdDate | updatedDate | externalId

Associations: an abstract Entity

Name: **User**

Attributes: username | email | password | firstName | lastName | userType | services | orders + [BaseEntity Attributes]

Associations: a user can be a service provider or buyer; a user can have many services (if service provider) and many orders (if buyer)

Name: **Service**

Attributes: title | description | price | category | provider | orders + [BaseEntity Attributes]

Associations: a service is offered by a service provider; a service can have many orders

Name: **Order**

Attributes: buyer | service | status | orderDate | completionDate + [BaseEntity Attributes]

Associations: an order is placed by a buyer for a specific service and is associated with a service provider

Name: **Category**

Attributes: name | description + [BaseEntity Attributes]

Associations: a category can have many services

## Architectural Design

### Conceptual Architecture

• The architectural style used for the SerVICE platform is REST (Representational State Transfer), and the architectural pattern is a combination of MVC (Model-View-Controller) and MVVM (Model-View-ViewModel).

• The REST architectural style is well-suited for designing web-based systems that are scalable, flexible, and easy to integrate with other systems and services. REST defines a set of constraints and principles for creating stateless, client-server systems that communicate using a uniform interface.

• The MVC pattern is a widely used pattern in web development that separates the application into three main components: the Model, the View, and the Controller. The Model represents the data and the business logic of the application, the View represents the user interface, and the Controller handles user input and updates the Model and View accordingly.

• In addition to MVC, the MVVM pattern is used on the client-side for better separation of concerns and improved maintainability. The ViewModel acts as an intermediary between the View and the Model, handling data bindings and UI logic.

• For the SerVICE platform, the REST architecture is used to create a set of resources that can be accessed and manipulated using HTTP requests. For example, the system has resources for users, services, orders, and categories. Clients can access these resources using HTTP requests, and the server can respond with the appropriate representations of the resources.

**+ High-Level System Architecture**

• Frontend: The frontend of the SerVICE platform is built using Blazor, a modern web framework for building single-page applications using C# , CSS, HTML (Blazor). The frontend follows the MVVM pattern for improved separation of concerns and maintainability.

• Backend: The backend is built using ASP.NET Core, a high-performance, cross-platform framework for building modern, cloud-based, connected applications. The backend follows the MVC pattern and exposes a set of RESTful APIs for the frontend to consume.

• Database: The data storage layer is implemented using Microsoft SQL Server, a robust and scalable relational database management system. The database stores information about users, services, orders, and categories.

• Authentication and Authorization: The platform uses ASP.NET Core Identity for handling user registration, authentication, and authorization.

• Communication: For real-time communication between users, the platform leverages SignalR, a library for adding real-time web functionality to applications.

• Overall, the SerVICE platform is designed to be easy to use, reliable, and scalable, with the ability to handle large volumes of traffic and transactions. The combination of REST, MVC, MVVM, Blazor, and modern web development technologies ensures a flexible, maintainable, and high-performance system.

**Package Design**

Diagram

Description automatically generated*A picture containing text, parking

Description automatically generated*

### Component and Deployment Diagram

Diagram, table

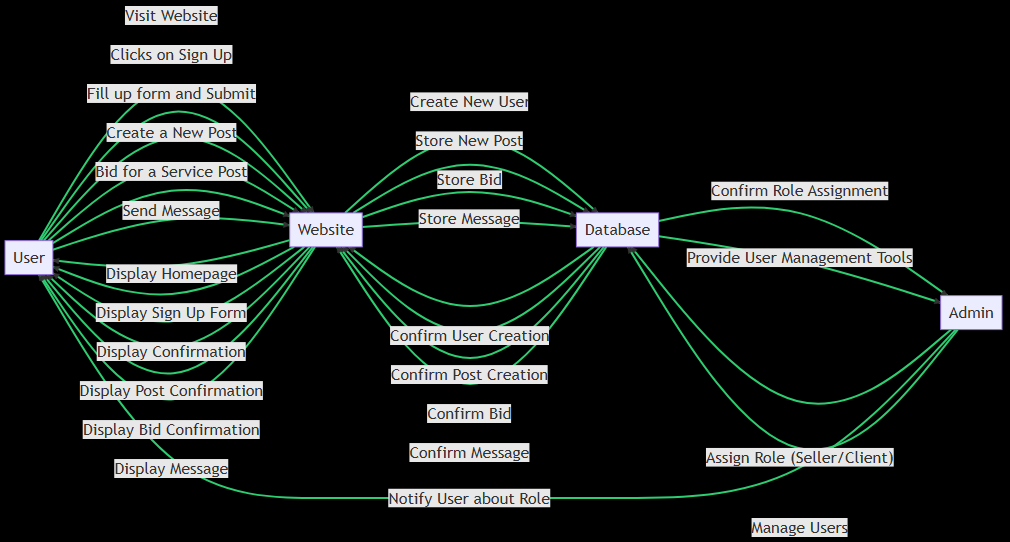
Description automatically generated with medium confidence

# Deliverable 3

## Design Model

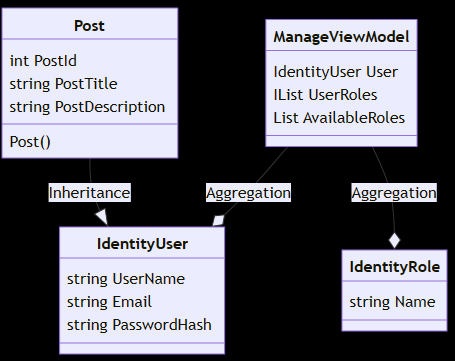
### Dynamic Behavior





### 

### Class Diagram

**

As for the GoF (Gang of Four) design patterns, the Factory Method pattern could be a good fit here. The Factory Method pattern provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created. This could be useful in your case for creating different types of Post objects depending on the user's role.

## Data Model

A screenshot of a computer

Description automatically generated with low confidence

# System Testing

Unit Testing: This involves testing individual components of the software to ensure they work as expected.

Test Cases:

Create a User and check if it's correctly stored in the database.

Create a Post and check if it's correctly stored in the database.

Create a Bid and check if it's correctly stored in the database.

Create a Message and check if it's correctly stored in the database.

Integration Testing: This involves testing the interaction between different components of the software.

Test Cases:

Check if a User can create a Post and if it's correctly linked to the user.

Check if a User can create a Bid for a Post and if it's correctly linked to both the user and the post.

Check if a User can send a Message and if it's correctly linked to the user.

Functional Testing: This involves testing the functionality of the software to ensure it works as expected.

Test Cases:

Check if a user can sign up and log in successfully.

Check if a seller can create a post successfully.

Check if a client can bid for a service post successfully.

Check if users can send and receive messages in real-time chat.

System Testing: This involves testing the software as a whole to ensure it works as expected.

Test Cases:

Check if the system can handle multiple users creating posts, bidding for posts, and sending messages at the same time.

Check if the system can handle large amounts of data, such as a large number of posts, bids, and messages.

Acceptance Testing: This involves testing the software to ensure it meets the requirements and expectations of the users.

Test Cases:

Check if the users find the system easy to use and navigate.

Check if the users are satisfied with the functionality of the system, such as creating posts, bidding for posts, and using the real-time chat.

# Future Improvements

Custom Usernames and Profile Pictures: Allow users to set custom usernames and upload profile pictures. This will make the platform more personalized and user-friendly.

Real-Time Chat between Client and Seller: Implement a real-time chat feature using SignalR or a similar technology. This will allow clients and sellers to communicate directly, discuss details about the service, negotiate prices, etc.

Ratings and Reviews: Implement a feature that allows clients to rate and review sellers and their services. This will help other clients make informed decisions when choosing a seller.

Search and Filter: Implement a search feature that allows clients to search for specific services. Additionally, implement filters that allow clients to filter services based on various criteria such as price, rating, location, etc.

Notification System: Implement a notification system that notifies users about important events such as new bids, messages, etc.

Secure Payment System: Implement a secure payment system that allows clients to pay for services directly through the platform. This will make transactions more convenient and secure for both clients and sellers.

Admin Dashboard: Implement an admin dashboard that allows the admin to manage all users, posts, bids, etc. This will make it easier for the admin to manage the platform.

Service Categories: Allow sellers to categorize their services. This will make it easier for clients to find the services they are looking for.

Bid History: Allow clients to view their bid history. This will help clients keep track of their bids and the services they have bid on.

# Conclusion

In conclusion, the "SerVice" web application is a comprehensive platform that facilitates the selling and bidding of services. It incorporates several key features such as custom usernames and profile pictures, real-time chat, ratings and reviews, search and filter capabilities, a secure payment system, and an admin dashboard for effective user management.

The application's design and functionality have been carefully planned and tested to ensure a seamless user experience. The use of UML diagrams and GoF patterns has helped to structure and optimize the system's architecture.

The application also includes several advanced features such as service categories, bid history, and user verification to enhance usability and security. These features, combined with a robust testing strategy, ensure that the application is reliable, secure, and user-friendly.

Overall, the "SerVice" application is a well-designed, feature-rich platform that effectively meets the needs of both service sellers and clients. It represents a significant contribution to the online service marketplace.

# Bibliography

Microsoft. (2023). ASP.NET Core SignalR. Retrieved from https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-7.0

Microsoft. (2023). Introduction to ASP.NET Core. Retrieved from https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0

Microsoft. (2023). Migrate from ASP.NET Core 6.0 to 7.0. Retrieved from https://docs.microsoft.com/en-us/aspnet/core/migration/60-to-70?view=aspnetcore-7.0&tabs=visual-studio

Microsoft. (2023). Migrate from ASP.NET Core 5.0 to 6.0. Retrieved from https://docs.microsoft.com/en-us/aspnet/core/migration/50-to-60?view=aspnetcore-7.0&tabs=visual-studio

Microsoft. (2023). .NET Framework. Retrieved from https://docs.microsoft.com/en-us/dotnet/framework/

Freeman, E., Robson, E., Bates, B., & Sierra, K. (2004). Head First Design Patterns. O'Reilly Media.

Martin, R. C. (2002). Agile Software Development, Principles, Patterns, and Practices. Prentice Hall.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.