< Art gallery>

Analysis and Design Document

Student: Tusa Laurentiu

**Group: 30431**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <12/04/23> | <1.0> | <Initial specification> | <Tusa Laurentiu> |
| <2/05/23> | <1.2> | <Design and data model> | <Tusa Laurentiu> |
| <16/05/23> | <2.0> | <Implementation> | <Tusa Laurentiu> |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 5

2.1 Conceptual Architecture 5

2.2 Package Design 6

2.3 Component and Deployment Diagrams 6

III. Elaboration – Iteration 1.2 6

1. Design Model 7

1.1 Dynamic Behavior 7

1.2 Class Design 9

2. Data Model 10

3. Unit Testing 11

IV. Elaboration – Iteration 2 11

1. Architectural Design Refinement 11

2. Design Model Refinement 12

V. Construction and Transition 14

1. System Testing 14

2. Future improvements 15

VI. Bibliography 16

# Project Specification

The goal of this project is to develop an art gallery application. Here are some key specifications that could be included in an Art gallery project:

1. User Authentication and Authorization: The application should have a user authentication and authorization system that allows users to create accounts, login and logout, and access different parts of the website based on their roles.
2. Gallery Management: The application should allow gallery administrators to manage the gallery, including adding new art pieces, editing existing ones, and removing them from the gallery.
3. Search and Filter: Users should be able to search and filter art pieces by different attributes such as artist name, art style, medium, size, price, etc.
4. Art Piece Details Page: Each art piece should have a dedicated details page with a high-quality image, detailed description, artist information, price, and related pieces.
5. Responsive Design: The application should be designed with responsive web design principles to ensure that it works well on different screen sizes and devices, including desktops, laptops, tablets, and smartphones.
6. Security: The application should be built with security in mind.

# 

# Elaboration – Iteration 1.1

# Domain Model

Domain model:

1. User: Represents a user of the application, with properties such as Id, Name, Email, Password.

2. Admin: Represents the admin who creates art pieces, with properties such as Id, Name, Email, ContactInformation, Password and manage the application.

3. ArtPiece: Represents an art piece that is displayed in the gallery, with properties such as Id, Title, DescriptionAndSize, ImageUrl, Price, TypeId.

4. Type: Represents a type or genre of art, with properties such as Id and Name.

5. Order: Represents an order placed by a user, with properties such as Id, UserId, TotalPrice.

Conceptual class diagrams:

O imagine care conține grafic

Descriere generată automat

# Architectural Design

## Conceptual Architecture

Based on the requirements of the application, a three-tier architecture with a layered design pattern should provide an adequate design.

The three tiers are:

1. Presentation Tier: This tier handles the user interface of the application, including the web pages and user interactions. In this architecture, the presentation tier can be implemented using ASP.NET Core 6 web application with Razor Pages or MVC.
2. Application Tier: This tier handles the business logic and workflow of the application. It is responsible for processing user requests and generating appropriate responses. In this architecture, the application tier can be implemented using a combination of services and controllers, following the Dependency Injection (DI) pattern.
3. Data Tier: This tier handles the storage and retrieval of data used by the application. It includes the database, data access objects (DAOs), and any other necessary data-related components. In this architecture, the data tier can be implemented using Entity Framework Core to access a local database.

For the layered design pattern, the application tier can be further divided into three layers:

1. Presentation Layer: This layer handles the communication between the presentation tier and the application tier. It receives user requests and sends them to the appropriate controller or service.
2. Business Layer: This layer contains the core business logic of the application, including validation, data transformation, and processing. It receives requests from the presentation layer, performs the necessary actions, and returns responses.
3. Data Access Layer: This layer handles the communication between the application tier and the data tier. It uses Entity Framework Core to perform database operations such as CRUD (Create, Read, Update, Delete) and querying.

The Dependency Injection pattern can be used to manage dependencies between components in the application, making it easy to replace or swap out components as needed. It also makes the application easier to test and maintain.

Overall, the three-tier architecture with a layered design pattern is a good fit for this Art gallery application because it separates the different concerns of the application and provides a clear and modular structure for development. It also allows for easy scaling and maintenance of the application.

## Package Design

*O imagine care conține diagramă

Descriere generată automat*

## Component and Deployment Diagrams

Component diagram:

O imagine care conține diagramă

Descriere generată automat

Deployment diagram:

O imagine care conține diagramă

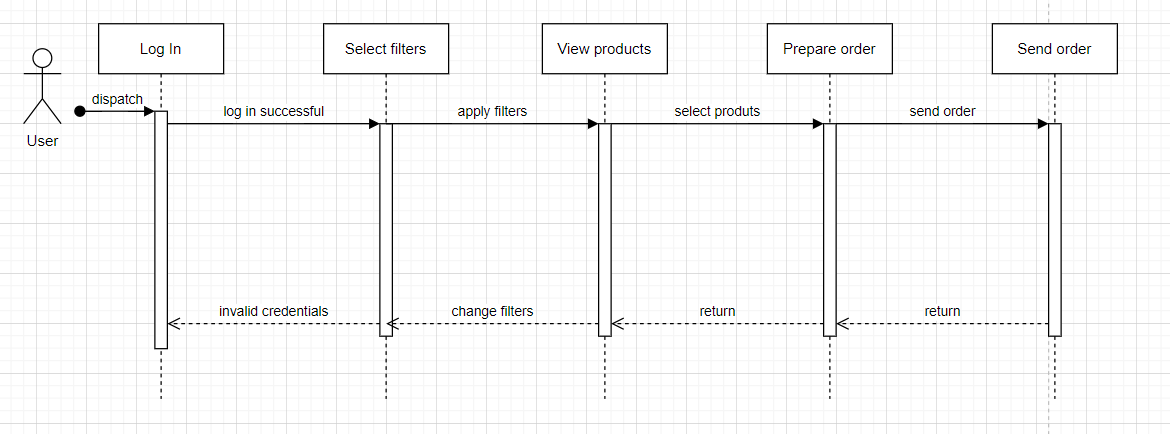
Descriere generată automat

# Elaboration – Iteration 1.2

# Design Model

## Dynamic Behavior

Sequence diagram:



Communication diagram:

O imagine care conține text, de interior

Descriere generată automat

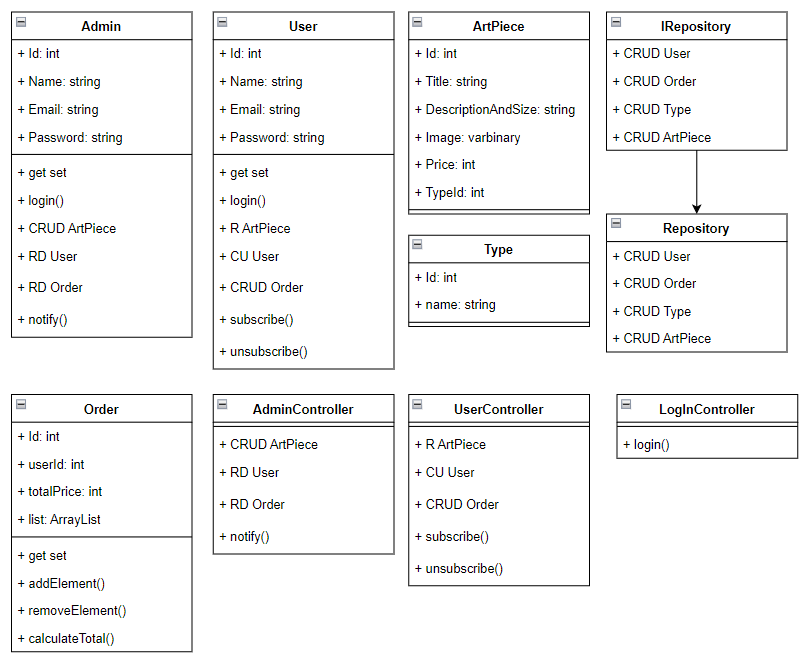
## Class Design

Singletone pattern for database connection

Observer pattern for notification propagation

Admins and users will have separate controllers that handle their operations, as well as separate views and layouts.

Repository implements the IRepository interface which holds all the functionalities related to database access.



# Data Model

O imagine care conține grafic, diagramă

Descriere generată automat

1 order can contain multiple products. Their total price will be calculated. Therefore, there is a one-to-many relation from order to product. One product has only 2 states: is part of an order(just 1) or not.

In table product, orderId is part of a foreign key and it also acts a as flag for revealing if the product is sold or not.

Product and type tables are in a many-to-many relationship, so an additional table was created to break down the dependency into 2 one to many relationships.

Table user has a field: “isAdmin” to distinguish regular users from admins.

# Elaboration – Iteration 2

# Architectural Design Refinement

* Conceptual Architecture:

The conceptual architecture of this application follows a layered architecture pattern, which provides a clear separation of concerns and promotes modular development. The key components of this system include the presentation layer, business logic layer, data access layer, and external dependencies.

The presentation layer is responsible for handling user interactions and displaying information to the users. It includes controllers, views, and other user interface components. The business logic layer contains the core application logic and implements various services and operations related to categories, products, orders, users, and more. It ensures the proper functioning and integrity of the system.

The data access layer interacts with the underlying database and provides access to the persistent storage. It includes repositories and data access objects (DAOs) responsible for performing CRUD operations on entities. External dependencies represent services or APIs utilized by this application for additional functionality or integrations.

The conceptual architecture promotes separation of concerns, making the system more maintainable and scalable. It allows for modular development and easier testing of individual components. The layers communicate with each other through well-defined interfaces, ensuring loose coupling and flexibility in adapting to changes.

* Package Design:

The package design in this application adheres to package design principles, which enhance maintainability, reusability, and modularity. The principles applied include encapsulation, cohesion, loose coupling, and modularity.

Packages in this application are structured in a way that encapsulates related classes and components with a clear separation of responsibilities. Each package represents a logical unit of functionality, such as services, repositories, controllers, or models. This promotes encapsulation, allowing packages to provide a well-defined public interface while hiding implementation details.

The classes within each package exhibit high cohesion, meaning they are closely related and focused on a specific responsibility. This ensures that each package has a clear purpose and that the classes within it work together to achieve a common goal. It enables easier maintenance, as changes made within a package are less likely to have unintended consequences on other parts of the system.

The packages in this application are designed with loose coupling in mind. Dependencies between packages are minimized, and interactions occur through well-defined interfaces. This reduces the impact of changes made to one package on other packages, making the system more resilient to modifications.

Modularity is another key aspect of your package design. Each package represents a self-contained and reusable unit of functionality. They can be easily understood and replaced, facilitating code reuse and promoting modular development practices. This design approach allows for easier maintenance, testing, and scalability of the system.

By following these package design principles, the application's codebase becomes more organized, maintainable, and extensible. It enables effective collaboration among developers and allows for easy integration of new features or changes in the future.

# Design Model Refinement

The refined UML class diagram for the application incorporates various class design principles and GRASP patterns to enhance its structure and maintainability. The diagram has been updated to ensure better encapsulation, cohesion, and responsibility assignment among the classes.

Presentation Layer

O imagine care conține text, captură de ecran, număr, Font

Descriere generată automat

Business Layer

O imagine care conține text, captură de ecran, multimedia, Font

Descriere generată automat

Data Access Layer

O imagine care conține text, captură de ecran, Font, Tipărire

Descriere generată automat

To improve cohesion and adhere to the Single Responsibility Principle (SRP), each class has been assigned a specific responsibility and focuses on a distinct functionality within the application. Methods and attributes that do not directly relate to the class's responsibility have been moved to other classes responsible for handling those functionalities. This refinement enhances the maintainability of the classes and reduces code complexity.

The associations between classes in the diagram have been reviewed to accurately represent the relationships between objects in the system. Cardinalities and multiplicity have been adjusted accordingly to reflect the nature of these associations. Dependencies between classes have been identified, and where possible, dependency injection or interfaces have been utilized to decouple the classes. This decoupling improves flexibility, testability, and extensibility of the system.

# Construction and Transition

# System Testing

*[Describe how you applied integration testing and present the associated test case scenarios.]*

# Future improvements

*[Present future improvements for the system]*

Filters: Filter by sold product, filter by price, filter by date, multiple filters

Orders: Include products in orders

Artist Management: The application should allow artists to create accounts and upload their art pieces to the gallery, along with artist information such as biography, portfolio, and contact information.

Performance Optimization: Analyze the application's performance bottlenecks and optimize critical sections of the code or database queries to enhance overall responsiveness. Techniques such as query optimization, data indexing, and asynchronous processing can be employed to improve performance.

Automated Testing: Expand the test coverage by implementing automated unit tests, integration tests, and end-to-end tests. This ensures the stability and reliability of the application, reduces the occurrence of bugs, and facilitates easier maintenance and refactoring.

Notifications: Add multiple notifications, also on price change and product sales

# Bibliography

Diagrams:

<https://app.diagrams.net/>

ASP.NET:

<https://www.youtube.com/watch?v=j2AYkOSzTUw>

<https://www.youtube.com/watch?v=xxSDmYumUwU>

<https://learn.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-6.0?view=aspnetcore-7.0>

Send Emails through .net:

<https://www.youtube.com/watch?v=lk5dhDzfzsU>

<https://www.youtube.com/watch?v=wa6BQ6Dmgac&t=319s>

Design patters:

<https://www.youtube.com/watch?v=ywJUmfAglPE>

<https://refactoring.guru/design-patterns/command>