FEI Art Gallery web Documentation

Version: 1.0

Prepared by: Marián Figula, Tomáš Jenčík, Radoslava Kridlová, Pavol Polednák, Ema Ševčíková

Date: December 4, 2024

Contents

1 Introduction

1.1 Purpose

The purpose of this application is to provide an online platform for artists and art enthusiasts to connect, share, and engage with artwork. The platform allows users to:

- Upload their artworks to showcase their creativity to a wider audience.
- Browse and view artworks submitted by others to appreciate diverse styles and techniques.
- Review artworks by leaving comments or ratings.
- Buy or sell artworks, enabling artists to monetize their talent and art lovers to acquire unique pieces for personal or commercial purposes.

1.2 Scope

This web application is designed to serve the needs of both individual artists and art collectors by providing the following functionalities:

- User Registration and Authentication
- Artwork Management
- Art Browsing and Discovery
- Social Interaction
- E-Commerce Integration
- Moderation and Security

1.3 Technologies Used

This section provides an overview of the technologies utilized in development of the web application:

Frontend

- **React.js**
- **CSS**

Backend

- **PHP**
- **REST API**
- **MySQL**
- **phpMyAdmin**
- **JWT (JSON Web Tokens)**

Development Tools and Platforms

- **Docker**.
- **GitHub**

Testing

- **Postman**

Design

- **Figma**

Security

- **bcrypt**

2 Installation Guide

This section provides instructions for setting up and running the application.

2.1 Requirements

To set up and run the application, you will need the following:

- **Docker**: Ensure that Docker is installed on your system. Docker is required to create a containerized environment for the application.
- **Docker Compose**: Docker Compose is required to manage multi-container Docker applications.

2.2 Setup Instructions

Follow these steps to build and configure the application:

1. Clone the repository containing the application code.
2. Open a terminal and navigate to the project directory.
3. Create a configuration file for the application by adding the following line to the '.env' file in the project directory:
   - `REACT_APP_SERVER_URL=http://localhost:8000/` for general setups.
   - `REACT_APP_SERVER_URL=http://localhost:9000/` when running on Linux.
4. Run the following command to build the Docker containers:

```
docker-compose build
```

5. Once the build is complete, run the following command to start the containers in detached mode:

```
docker-compose up -d
```

2.3 Running the Application

Once the application is up and running, access it through your web browser:

- Open your browser and navigate to `localhost:3000`.
- The application should be live and accessible from this address.

3 Features and Functionality

3.1 Core Features

- **User Registration and Authentication**: Users can create an account with email and password. After registration, users can log in to their account to access their dashboard and perform other actions.
- **Artwork Upload**: Registered users can upload their artwork to the platform. Each artwork can include details such as the title, description, image file, and price.
- **Artwork Display and Gallery**: Users can browse and view artworks uploaded by other users. Artworks are displayed in a visually appealing gallery format, with options to filter by category, artist, or price.
- **Artwork Review System**: Users can leave reviews and rate artworks, providing feedback to the artists and helping other users make informed decisions.
- **Artwork Purchase**: Users can purchase artworks directly through the platform. The app integrates with payment systems to process transactions securely.
- **Search and Filter**: Users can search for artworks by keywords, categories, price range, or artist. Filters enable users to quickly find artworks that match their preferences.
- **Responsive Design**: The application is fully responsive, ensuring that users can access the platform on desktop, tablet, and mobile devices without any issues.
- **Admin Dashboard**: Admins have access to a special dashboard where they can manage users, monitor uploaded artworks and handle content moderation.

3.2 User Roles and Permissions

- **Admin**:
  - Full access to all user data.
  - Can modify user accounts.

  - Can manage all artworks, including the ability to delete, or modify uploads.
  - Can view all user reviews and ratings while also has the ability to modify them.

- **Registered User**:
  - Can upload, edit, and delete their own artworks.
  - Can browse and view all publicly available artworks.
  - Can leave reviews and ratings on artworks.
  - Can purchase artworks listed for sale.
  - Can access their own profile and manage own uploads and reviews.

- **Guest User**:
  - Can browse and view publicly available artworks but cannot leave reviews, make purchases, or upload content.
  - Can search for artworks using filters.

4 Architecture

4.1 System Architecture Diagram

This subsection includes a high-level diagram of the overall system architecture, illustrating the interaction between the frontend, backend, and database.

The diagram 1 depicts key components such as:

- The user interface.
- Backend API services.
- The database and data flow between components.
- Additional services such as **Docker** (for containerization) and **phpMyAdmin** (for database management).

Arrows can represent data exchange, such as user requests, API responses, and database queries.

4.2 Acivity diagram

This activity diagram 2 illustrates the user flow for registration, login, and subsequent actions on the platform.

- **Registration and Login**:
  - Users can either register through the `RegisterSite` or log in via the `LoginSite`.
  - Validation checks ensure the registration or login credentials are correct. If invalid, the user is prompted to re-enter the details.

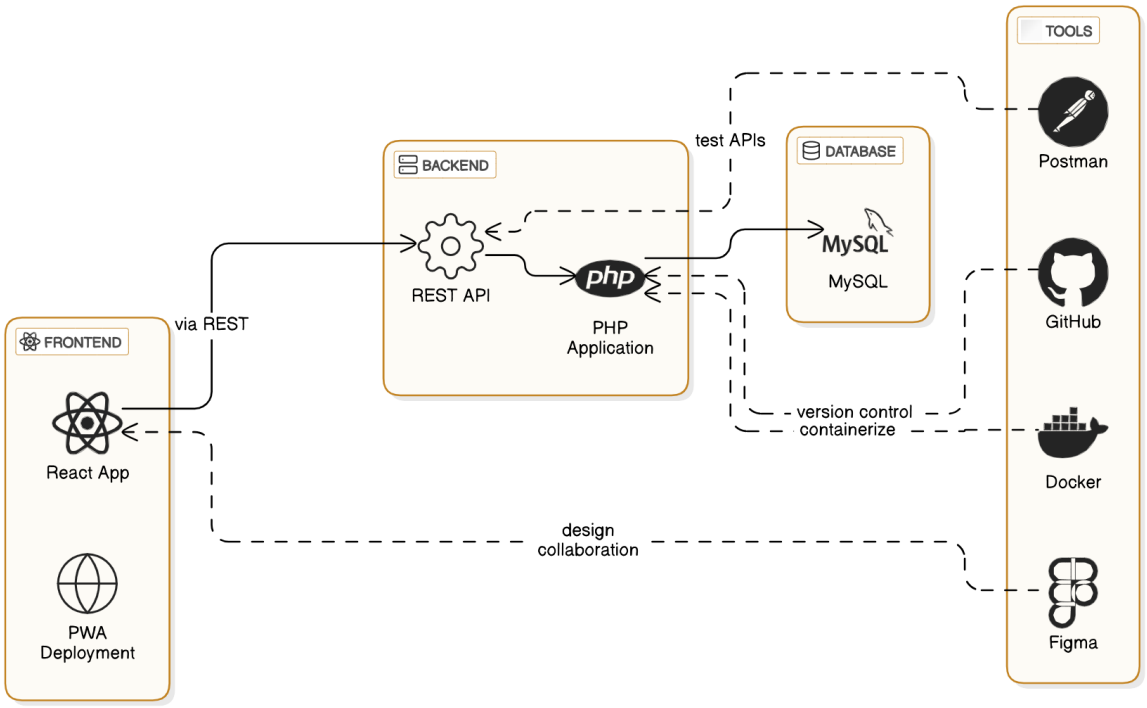

Figure 1: Architecture diagram of the app

- **Homepage Access**:
  - Upon successful login or registration, users are redirected to the `Homepage`.
- **User Actions**:
  - On the homepage, users decide whether to make changes or proceed with specific actions:
    - `Add Review`
    - `Edit Account Information`
    - `Upload Images`
- **Authorization Checks**:
  - Before completing any action, the system checks if the user is authenticated. If not, the user is redirected accordingly.
- **Redirection**:
  - Once authenticated, users are redirected to the appropriate pages based on their selected actions.

4.3 Database Schema

Figure 3 presents the database schema used for the platform. It comprises four key entities:

- **User Table**:
  - Stores user information such as `email`, `username`, `password`, and security questions.
  - The primary key is `user_id`.
- **Art Table**:
  - Contains details of uploaded artworks, including `title`, `description`, `image_URL`, `price`, and `upload_date`.
  - The primary key is `art_id`, and it references the `user_id` as a foreign key to associate artworks with their creators.
- **Review Table**:
  - Records user reviews for artworks, including `review_text`, `rating`, and `review_creation_da`
  - The primary key is `review_id`, and it references both `art_id` and `user_id` as foreign keys.
- **Credit Card Table**:
  - Stores payment information, such as `card_number`, `expiration_date`, and `cvc`.
  - The primary key is `credit_card_id`, and it references the `user_id` as a foreign key to associate payment details with users.

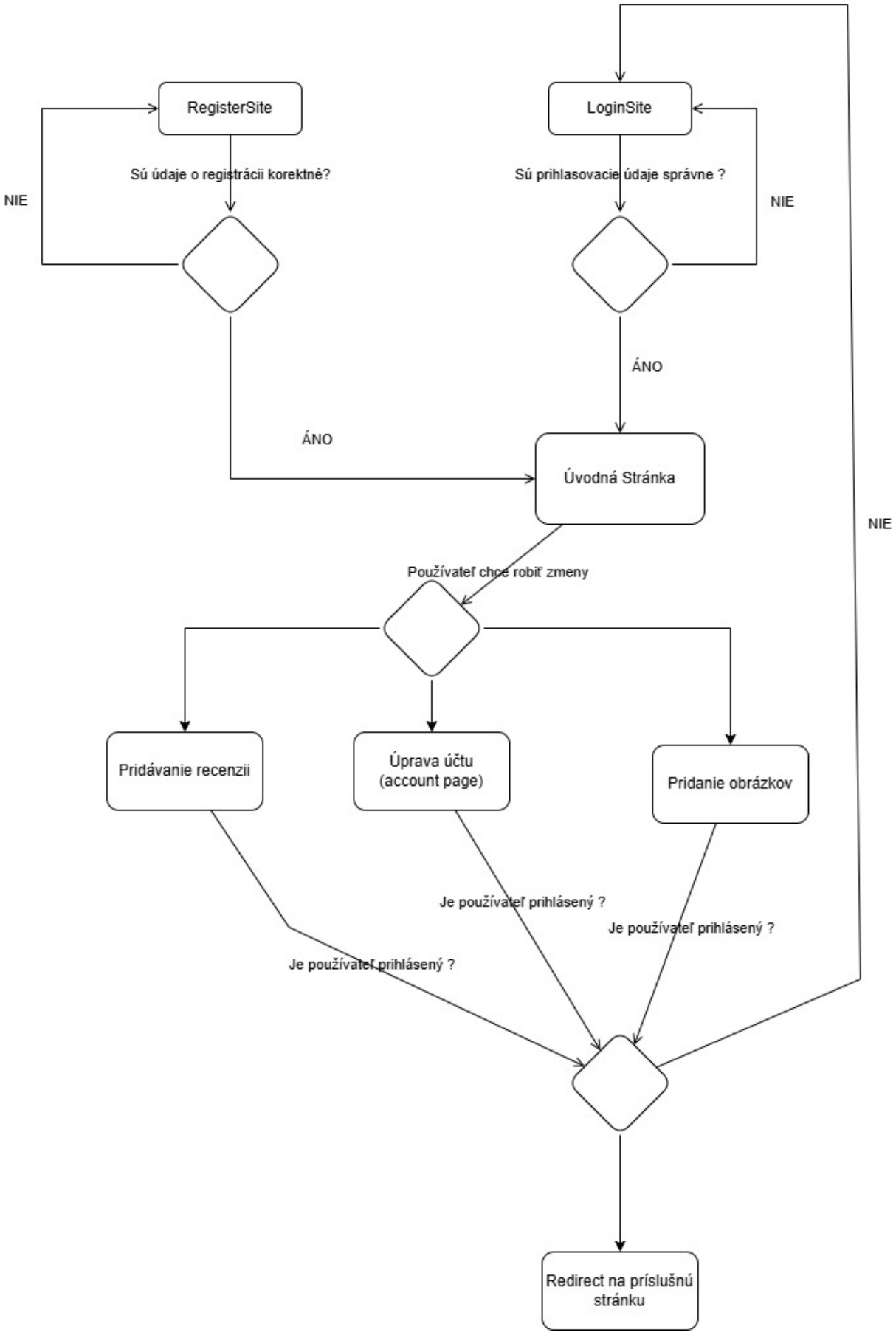

Figure 2: Activity diagram

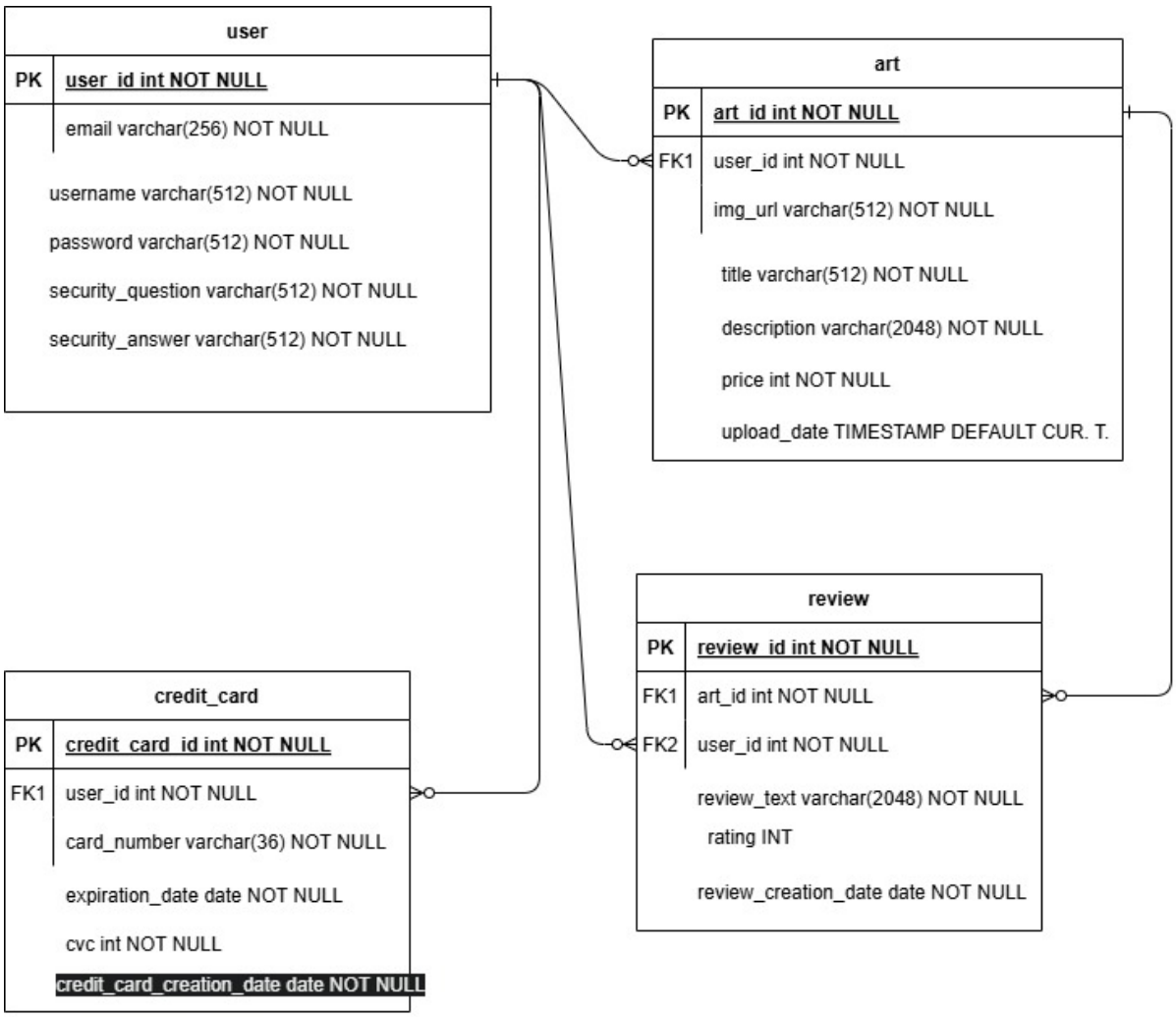

Figure 3: Diagram of the database

5 API Documentation

5.1 Art endpoints

5.1.1 Create

This endpoint allows creating a new art entry for a user with with details like title, description, price, and image. The art data must be provided via a POST request, and an image file must be uploaded as well.

- **Method:** POST
- **URL:** /api/art/create.php
- **Request Body:**

```
{
    "email": "user@example.com"          (string, required)
    "title": "Artwork Title"             (string, required)
    "description": "Artwork description" (string, required)
    "price": 100                         (integer, optional)
    "file":                              (file, required)
}
```

- **Response Codes:**
  - 201 Created: *Artwas successfully created.*
  - 400 Bad Request: *Invalid input provided or missing required fields.*
  - 404 Not Found: *User not found.*
  - 500 Internal Server Error: *Failed to create art due to server error.*

5.1.2 Delete

This endpoint allows deleting multiple artworks by their IDs. The IDs must be provided in an array within the request body, and the request method must be DELETE.

- **Method:** DELETE
- **URL:** /api/art/delete.php
- **Request Body:**

```
{
    "ids": [1, 2, 3]    (array of valid positive integers)
}
```

- **Response Codes:**
  - 200 OK: *Artworks were successfully deleted.*
  - 400 Bad Request: *No valid IDs provided.*
  - 500 Internal Server Error: *Failed to delete artworks.*
  - 405 Method Not Allowed: *Invalid request method (only DELETE is allowed).*

5.1.3 Read

This endpoint allows retrieving art information by specifying the art ID or user ID as query parameters. If neither is provided, all art records are returned.

- **Method:** GET
- **URL:** /api/art/read.php
- **Query Parameters:**

```
- id (int, optional)
- user_id (int, optional)
```

- **Response Codes:**
  - 200 OK: *Art(s) successfully retrieved.*
  - 404 Not Found: *Art with the specified ID or user does not exist.*
  - 405 Method Not Allowed: *Invalid request method used (only GET is allowed).*

5.1.4 Update

This endpoint allows partial updates to an artwork's information. The art ID is required to identify the artwork, and at least one of the following parameters must be provided to update: title, description, or price.

- **Method:** PUT
- **URL:** /api/art/update.php
- **Request Body:**

```
{
    "id": 1,                        (int, required)
    "title": "New Title",           (string)
    "description": "Description",   (string)
    "price": 100                    (int or null)
}
```

- **Response Codes:**
  - 200 OK: *Artworksuccessfully updated.*
  - 400 Bad Request: *Missing required parameters (id) or invalid data.*
  - 404 Not Found: *Artworkd with the specified ID does not exist.*
  - 405 Method Not Allowed: *Invalid request method used (only PUT is allowed).*
  - 500 Internal Server Error: *Failed to update the credit card due to server error.*

5.2 Card endpoints

5.2.1 Create

This endpoint allows creating a new credit card entry for a user with required details like user_id, card_number, expiration_date, and CVC. The credit card data must be provided in JSON format within the request body, and the request method must be POST.

- **Method:** POST
- **URL:** /api/credit_card/create.php
- **Request Body:**

```
{
    "user_id": 1,                       (int, required)
    "card_number": "4946511278435961", (string, required)
    "expiration_date": "2027-12-31",   (string in YYYY-MM-DD format, required)
    "cvc": "123"                        (string, required)
}
```

- **Response Codes:**
  - 200 OK: *Credit card was successfully created.*
  - 400 Bad Request: *Invalid input provided or missing required fields.*
  - 500 Internal Server Error: *Failed to create credit card due to server error.*

5.2.2 Delete

This endpoint allows deleting multiple credit cards by their IDs. The IDs must be provided in an array within the request body, and the request method must be DELETE.

- **Method:** DELETE
- **URL:** /api/credit_card/delete.php
- **Request Body:**

```
{
    "ids": [1, 2, 3]    (array of valid positive integers)
}
```

- **Response Codes:**
  - 200 OK: *Credit cards were successfully deleted.*
  - 400 Bad Request: *No valid IDs provided.*
  - 500 Internal Server Error: *Failed to delete credit cards.*
  - 405 Method Not Allowed: *Invalid request method (only DELETE is allowed).*

5.2.3 Read

This endpoint allows retrieving credit card information by specifying the card ID or user ID as query parameters. If neither is provided, all cards are returned for the specified user (admin privileges are required to view all cards).

- **Method:** GET
- **URL:** /api/credit_card/read.php
- **Query Parameters:**

```
- id (int, optional)
- user_id (int, optional)
```

- **Response Codes:**
  - 200 OK: *Credit card(s) successfully retrieved.*
  - 403 Forbidden: *Admin privileges are required to view all credit cards.*
  - 404 Not Found: *Credit card with the specified ID does not exist.*
  - 405 Method Not Allowed: *Invalid request method used (only GET is allowed).*

5.2.4 Update

This endpoint allows partial updates to a credit card's information. The credit card ID is required to identify the card, and the expiration date can be updated.

- **Method:** PUT
- **URL:** /api/credit_card/update.php
- **Request Body:**

```
{
    "id": 1,                            (int, required)
    "expiration_date": "2026-05-31"     (string in YYYY-MM-DD format)
}
```

- **Response Codes:**
  - 200 OK: *Credit card successfully updated.*
  - 400 Bad Request: *Missing required parameters (id) or invalid data.*
  - 404 Not Found: *Credit card with the specified ID does not exist.*
  - 405 Method Not Allowed: *Invalid request method used (only PUT is allowed).*
  - 500 Internal Server Error: *Failed to update the credit card due to server error.*

5.3 Cart endpoints

5.3.1 Create

This endpoint is designed to handle the creation of a new cart entry for a user. It accepts a JSON payload in the request body and processes the data only if the HTTP method is POST.The ‘user_id‘ is a mandatory field, and failure to provide it results in a 400 Bad Request error.

- **Method:** POST
- **URL:** /api/cart/create.php
- **Request Body:**

```
{
    "user_id": 1,           (int, required)
}
```

- **Response Codes:**
  - 201 Created: *Cart and associated art were successfully created.*
  - 400 Bad Request: *Invalid input provided or missing required fields.*
  - 405 Method Not Allowed: *The endpoint only supports the POST method..*
  - 500 Internal Server Error: *An error occurred during the creation process.*

5.3.2 Delete

This endpoint allows deleting a cart entry based on the provided ’cart_id’. The ’cart_id’ must be included in the request body as a JSON payload, and the request method must be DELETE.

- **Method:** DELETE
- **URL:** /api/cart/delete.php
- **Request Body:**

```
{
     "cart_id": 123 (int, required)
}
```

- **Response Codes:**
  - 200 OK: *Cart successfully deleted.*
  - 400 Bad Request: *Missing or invalid ‘cart_id‘ in the request.*
  - 405 Method Not Allowed: *The endpoint only supports the DELETE method.*
  - 500 Internal Server Error: *Failed to delete the cart due to a server error.*

5.3.3 Read

This endpoint allows retrieving cart details based on either a specific cart ID or a user ID. It supports the GET method and requires query parameters to filter the cart data.

- **Method:** GET
- **URL:** /api/cart/read.php
- **Query Parameters:**

```
- id (int, optional)
- user_id (int, optional)
```

- **Response Codes:**
  - 200 OK: *Cart details successfully retrieved.*
  - 400 Bad Request: *No valid query parameters provided or invalid input.*
  - 404 Not Found: *No cart found matching the provided criteria.*
  - 405 Method Not Allowed: *The endpoint only supports the GET method.*
  - 500 Internal Server Error: *Failed to retrieve cart details due to a server error.*

5.4 Cart art endpoints

5.4.1 Create

This endpoint allows adding a specific art item to a user's cart. It requires the POST method and a JSON request body containing the 'user_id' and 'art_id'.

- **Method:** POST
- **URL:** /api/cart_art/create.php
- **Request Body:**

```
{
    "user_id": 1,   (int, required)
    "art_id": 123   (int, required)
}
```

- **Response Codes:**
  - 201 Created: *Art was successfully added to the cart..*
  - 400 Bad Request: *Missing or invalid 'user_id' or 'art_id' in the request body.*
  - 404 Not Found: *Cart not found for the specified user.*
  - 405 Method Not Allowed: *The endpoint only supports the POST method.*
  - 500 Internal Server Error: *Failed to add the art due to a server-side error.*

5.4.2 Delete

This endpoint allows removing a specific art item from a user's cart. It requires the DELETE method and a JSON request body containing the 'user_id' and 'art_id'.

- **Method:** DELETE
- **URL:** /api/cart_art/delete.php
- **Request Body:**

```
{
    "user_id": 1,   (int, required)
    "art_id": 123   (int, required)
}
```

- **Response Codes:**
  - 200 OK: *Art successfully removed from the cart.*
  - 400 Bad Request: *Missing or invalid 'user_id' or 'art_id' in the request body.*
  - 404 Not Found: *Cart not found for the specified user.*
  - 405 Method Not Allowed: *The endpoint only supports the DELETE method.*
  - 500 Internal Server Error: *Failed to remove the art due to a server-side error.*

5.4.3 Read

This endpoint allows retrieving a list of art IDs associated with a user's cart. It supports the GET method and requires a 'user_id' to fetch the cart and its associated art data.

- **Method:** GET
- **URL:** /api/cart_art/read.php
- **Query Parameters:**

```
- user_id (int, required)
```

- **Response Codes:**
  - 200 OK: *Successfully retrieved the list of art IDs associated with the cart.*
  - 400 Bad Request: *Missing or invalid 'user_id' query parameter.*
  - 404 Not Found: *No cart found for the provided user ID, or no associated art found.*
  - 405 Method Not Allowed: *The endpoint only supports the GET method.*
  - 500 Internal Server Error: *Failed to retrieve cart or art data due to a server error.*

5.4.4 Buy

This endpoint allows the user to buy and remove specific art items from their cart. It requires the POST method and a JSON request body containing the ‘user_id‘ and an array of ‘art_ids‘ to be purchased.

- **Method:** POST
- **URL:** /api/cart_art/buy.php
- **Request Body:**

```
{
    "user_id": 1,           (int, required)
    "art_ids": [1, 2, 3]    (int array, required)
}
```

- **Response Codes:**
  - 200 OK: *Arts successfully bought and removed from the cart.*
  - 400 Bad Request: *Missing or invalid ‘user_id‘ or ‘art_ids‘ in the request body.*
  - 404 Not Found: *Cart not found for the specified user or art items not found.*
  - 405 Method Not Allowed: *The endpoint only supports the POST method.*
  - 500 Internal Server Error: *Failed to remove arts from the cart or delete them due to a server-side error.*

5.4.5 Art details

This endpoint allows retrieving multiple art details by their IDs.

- **Method:** POST
- **URL:** /api/art/artDetails.php
- **Request Body:**

```
{
    "art_ids": [1, 2, 3]    (int array, required)
}
```

- **Response Codes:**
  - 200 OK: *Art(s) successfully retrieved.*
  - 400 Bad Request: *Missing or invalid art IDs.*
  - 404 Not Found: *One or more artworks not found.*

5.5 Review endpoints

5.5.1 Create

This endpoint allows creating a new review by specifying user email, art ID, review text, and rating. The user email and art ID are required to identify the user and artwork, and review text and rating are required for the review itself.

- **Method:** POST
- **URL:** /api/review/create.php
- **Request Body:**

```
{
        "email": "user@example.com", (string, required)
        "art_id": 1,                 (integer, required)
        "review_text": "Great art!", (string, required)
        "rating": 5                  (int between 1 and 5, required)
}
```

- **Response Codes:**
  - 201 Created: *Review successfully created.*
  - 400 Bad Request: *Missing required parameters or invalid data.*
  - 404 Not Found: *User or artwork with the specified identifiers does not exist.*
  - 405 Method Not Allowed: *Invalid request method used (only POST is allowed).*
  - 500 Internal Server Error: *Failed to create the review due to server error.*

5.5.2 Delete

This endpoint allows deleting multiple reviews by their IDs. The IDs must be provided in an array within the request body, and the request method must be DELETE.

- **Method:** DELETE
- **URL:** /api/review/delete.php
- **Request Body:**

```
{
    "ids": [1, 2, 3]        (valid positive int array)
}
```

- **Response Codes:**
  - 200 OK: *Reviews were successfully deleted.*
  - 400 Bad Request: *No valid IDs provided.*
  - 405 Method Not Allowed: *Invalid request method (only DELETE is allowed).*
  - 500 Internal Server Error: *Failed to delete reviews.*

5.5.3 Read

This endpoint allows retrieving a list of art IDs associated with a user's cart. It supports the GET method and requires a 'user_id' to fetch the cart and its associated art data.

- **Method:** GET
- **URL:** /api/cart_art/read.php
- **Query Parameters:**

```
- user_id (int, required)
```

- **Response Codes:**
  - 200 OK: *Successfully retrieved the list of art IDs associated with the cart.*
  - 400 Bad Request: *Missing or invalid 'user_id' query parameter.*
  - 404 Not Found: *No cart found for the provided user ID, or no associated art found.*
  - 405 Method Not Allowed: *The endpoint only supports the GET method.*
  - 500 Internal Server Error: *Failed to retrieve cart or art data due to a server error.*

5.5.4 Update

This endpoint allows updating review information by specifying the review ID and providing new values for review text or rating. The review ID is required to identify the review, and at least one of the following parameters must be provided to update either the review text or the rating.

- **Method:** PUT
- **URL:** /api/review/update.php
- **Request Body:**

```
{
    "id": 1,                    (int, required)
    "review_text": "new text",  (string, optional)
    "rating": 5                 (int between 1 and 5, otional)
}
```

- **Response Codes:**
  - 200 OK: *Review successfully updated.*
  - 400 Bad Request: *Missing required parameters (id) or invalid data.*
  - 404 Not Found: *Review with the specified ID does not exist.*
  - 405 Method Not Allowed: *Invalid request method used (only PUT is allowed).*
  - 500 Internal Server Error: *Failed to update the review due to server error.*

5.6 User endpoints

5.6.1 Admin read

This endpoint retrieves all users from the database. The endpoint can only be accessed by users with an admin role (‘S‘). The request method must be ‘GET‘.

- **Method:** GET
- **URL:** /api/user/adminRead.php
- **Query Parameters:**

```
none
```

- **Response Codes:**
  - 200 OK: *Users data successfully retrieved.*
  - 400 Bad Request: *Invalid input provided.*
  - 405 Method Not Allowed: *The endpoint only supports the GET method.*
  - 500 Internal Server Error: *Failed to retrieve users due to server error.*

5.6.2 Create

This endpoint allows creating a new user with required details like username, email, password, and optional security questions. The user data must be provided in JSON format within the request body, and the request method must be POST.

- **Method:** POST
- **URL:** /api/user/create.php
- **Request Body:**

```
{
    "username": "exampleUser",          (string)
    "email": "user@example.com",        (string)
    "password": "password123",          (string)
    "security_question": "Your pet's name?", (string)
    "security_answer": "Fluffy",        (string).
    "role": "U"                         'U' (User) or 'A' (Admin).
                                        Defaults to 'U'.
}
```

- **Response Codes:**
  - 201 Created: *User successfully created.*
  - 400 Bad Request: *Invalid input provided or missing required fields.*
  - 500 Internal Server Error: *Failed to create user due to server error.*

5.6.3 Delete

This endpoint allows deleting multiple users by their IDs. The IDs must be provided in an array within the request body, and the request method must be DELETE.

- **Method:** DELETE
- **URL:** /api/user/delete.php
- **Request Body:**

```
{
    "ids": [1, 2, 3]        (valid positive int array)
}
```

- **Response Codes:**
  - 200 OK: *Users were successfully deleted.*
  - 400 Bad Request: *No valid IDs provided.*
  - 405 Method Not Allowed: *Invalid request method (only DELETE is allowed).*
  - 500 Internal Server Error: *Failed to delete users.*

5.6.4 Forgot Password

This endpoint allows a user to reset their password by providing their email, new password, repeated password, and security answer.

- **Method:** POST
- **URL:** /api/user/forgotPassword.php
- **Request Body:**

```
{
    "email": "user@example.com",            (string, required)
    "password": "newPassword123",           (string, required)
    "repeated_password": "newPassword123",  (string, required)
    "securityAnswer": "Fluffy"              (string, required)
}
```

- **Response Codes:**
  - 201 Created: *Password reset successfully.*
  - 400 Bad Request: *Missing or invalid input data.*
  - 401 Unauthorized: *Invalid security answer.*
  - 404 Not Found: *User with the specified email does not exist.*
  - 405 Method Not Allowed: *Invalid request method used (only POST is allowed).*
  - 500 Internal Server Error: *Failed to reset the password due to server error.*

5.6.5 Login

This endpoint allows a user to log in by providing their email and password. If the credentials are correct, a session is started.

- **Method:** POST
- **URL:** /api/user/login.php
- **Request Body:**

```
{
    "email": "user@example.com",            (string, required)
    "password": "newPassword123",           (string, required)
}
```

- **Response Codes:**
  - 201 Created: *Login successful.*
  - 400 Bad Request: *Missing required fields (email or password).*
  - 401 Unauthorized: *Invalid email or password.*
  - 404 Not Found: *User with the specified email does not exist.*
  - 405 Method Not Allowed: *Invalid request method used (only POST is allowed).*

Upon successful login, session variables are set to maintain the user's session.

5.6.6 Read

This endpoint allows retrieving user information by specifying the user ID or email as query parameters. If neither is provided, all users are returned (admin only).

- **Method:** GET
- **URL:** /api/user/read.php
- **Query Parameters:**

```
- id        (int, optional)
- email     (string, optional)
```

- **Response Codes:**
  - 200 OK: *User(s) successfully retrieved.*
  - 403 Forbidden: *Admin privileges are required to view all users.*
  - 404 Not Found: *User with the specified ID or email does not exist.*
  - 405 Method Not Allowed: *The endpoint only supports the GET method.*

If neither 'id' nor 'email' is provided, all users are returned, which requires admin privileges.

5.6.7 Register

This endpoint allows a new user to register by providing necessary details such as username, email, password, and security information. If the registration is successful, a session is started.

- **Method:** POST
- **URL:** /api/user/register.php
- **Request Body:**

```
{
    "username": "newUser",                  (string, required)
    "email": "user@example.com",            (string, required)
    "password": "newPassword123",           (string, required)
    "repeatedPassword": "password123",      (string, required)
    "securityQuestion": "Your pet's name?", (string, required)
    "securityAnswer": "Fluffy"              (string, required)
}
```

- **Response Codes:**
  - 201 Created: *User successfully created..*
  - 400 Bad Request: *Missing or invalid input data.*
  - 405 Method Not Allowed: *Invalid request method used (only POST is allowed).*
  - 500 Internal Server Error: *Failed to create the user due to server error.*

5.6.8 Update

This endpoint allows partial updates to a user's information. The user ID is required to identify the user, and at least one of the following parameters must be provided to update either the username or the email.w ID is required to identify the review, and at least one of the following parameters must be provided to update either the review text or the rating.

- **Method:** PUT
- **URL:** /api/user/update.php
- **Request Body:**

```
{
    "id": 1,                        (int, required)
    "username": "newUsername",      (string, optional)
    "email": "newemail@example.com" (string, otional)
}
```

- **Response Codes:**
  - 200 OK: *User successfully updated.*

- 400 Bad Request: *Missing required parameters (id) or invalid data.*
- 404 Not Found: *User with the specified ID does not exist.*
- 405 Method Not Allowed: *Invalid request method used (only PUT is allowed).*
- 500 Internal Server Error: *Failed to update the user due to server error.*

Upon successful registration, session variables are set to maintain the user's session.

6 Security

The implementation includes security measures and user role management. Passwords are securely hashed to protect user data and ensure compliance with security best practices. The system supports two distinct roles: user and admin, each with specific permissions and access levels. Authentication and authorization are managed using JWT (JSON Web Tokens), enabling secure and efficient verification of user identities and access rights. This ensures that only authorized users can perform actions appropriate to their role.

7 User Guide

7.1 Homepage

The homepage allows users to browse through all the available artworks. User can view details about each artwork, including its name, description, author, price, and user reviews. User has the option to sort the artworks by price or rating. A review for any artwork can be easily added. User can also upload an artwork of his own or add an artwork to the shopping cart. User can also navigate to the shopping cart or user profile.

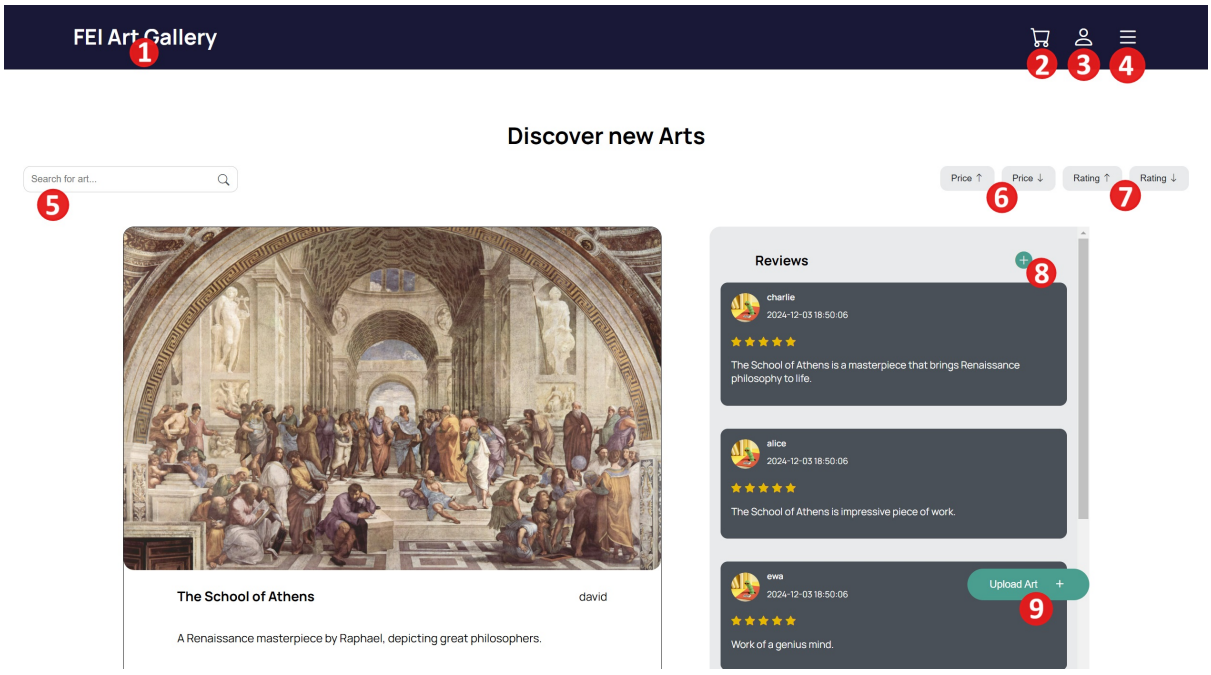

Figure 4: Homepage

1. Navigate to the Homepage - Return to the main page anytime by clicking the homepage button.
2. Navigate to Your Cart - View the items in your shopping cart.
3. Navigate to Your Profile - Access your account information.
4. Display the Web Menu - Open the menu to other options.

5. Search for Artwork - Use the search bar to find specific artworks by their name.
6. Sort Artworks by Price - Organize the artworks in ascending or descending order of price.
7. Sort Artworks by Rating - Sort the artworks based on their user ratings.
8. Add a Review - Share your feedback on an artwork by submitting a review.
9. Upload New Artwork - Upload your own artwork to display and sell on the platform.

7.2 Cart

This part of the website lists the selected artworks with their details, including the title, artist, and price. Each artwork has a corresponding delete icon for removal from the cart. The total price of all items is displayed at the bottom of the page, next to the label. Users can either click "Continue" to proceed to the payment or "Continue Shopping" to add more items to their cart.

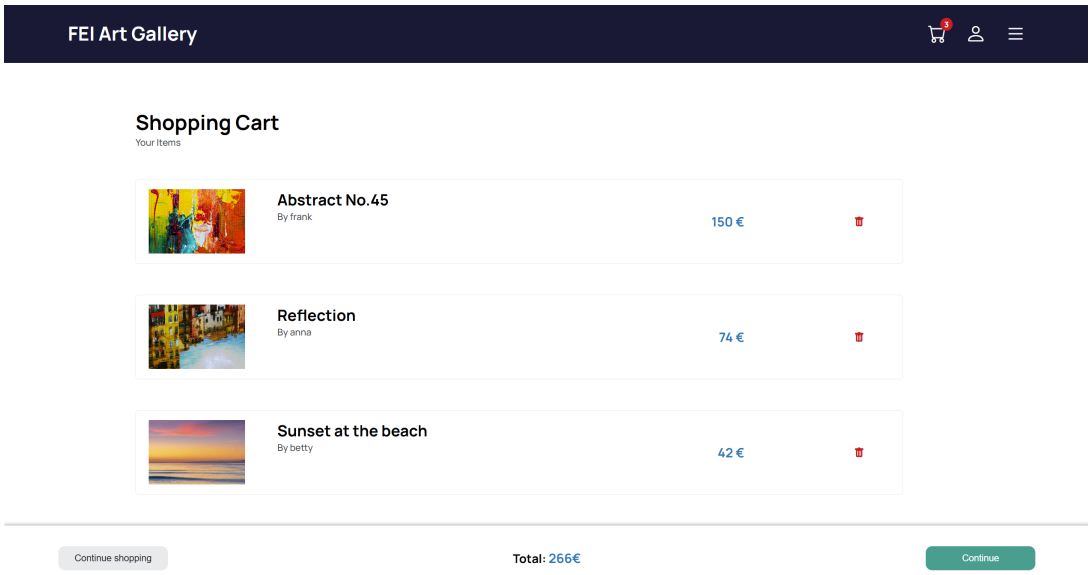

Figure 5: Cart

7.3 Payment

The website features a payment section where users can finalize their purchase. After entering their card details, they can click the "Pay Now" button to complete the transaction and purchase the artworks in their cart. The total price of the selected artworks is prominently displayed in the bottom-right corner of the screen. Additionally, users have the option to return to the Order Summary page for review or modifications.

7.4 Profile

At the top, the username and email address are presented in text fields. A profile picture is displayed to the left. Below the input fields, there are two buttons: "My Posts", which

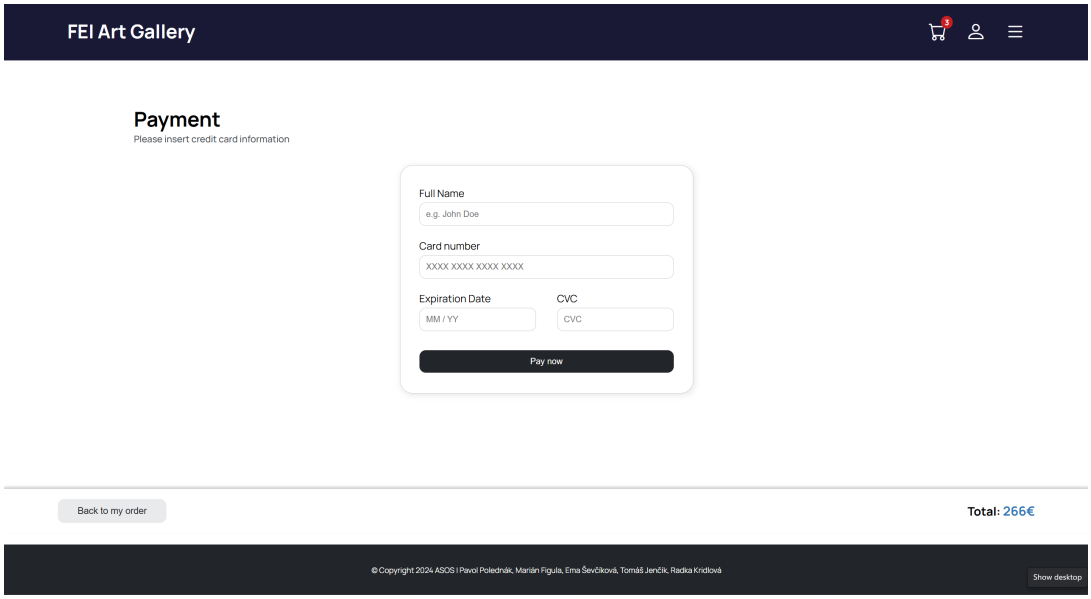

Figure 6: Payment

navigates to the user's artwkors, and "Review History", which navigates to the list of user's reviews.

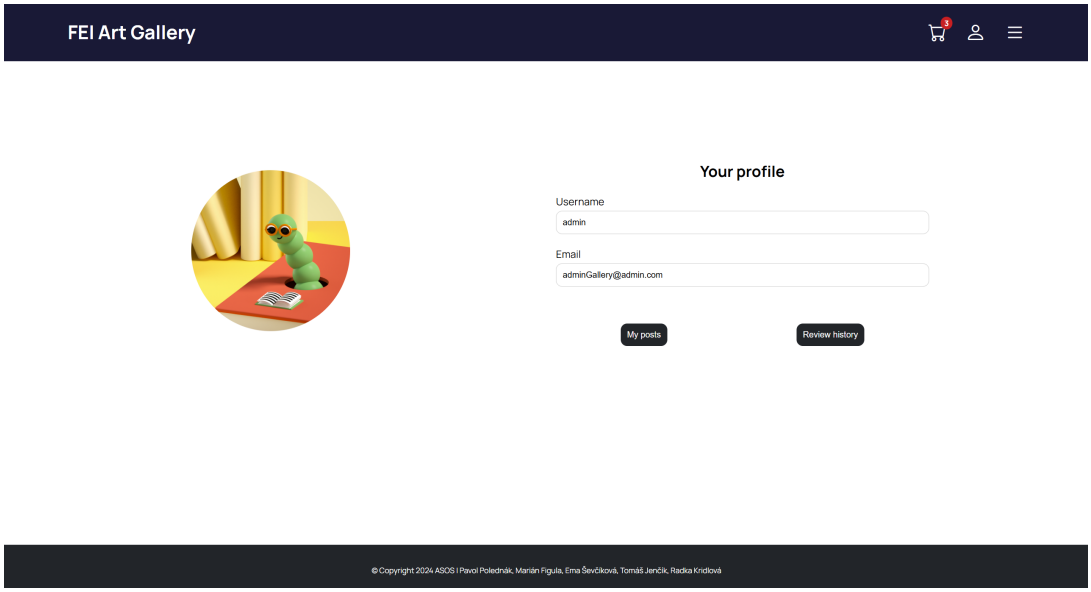

Figure 7: Profile

7.5 My Posts

The My Posts screen allows users to view all the artworks they have uploaded to the platform. Each artwork is displayed with its information. Title, description, and price can be edited by clicking on the edit icon.

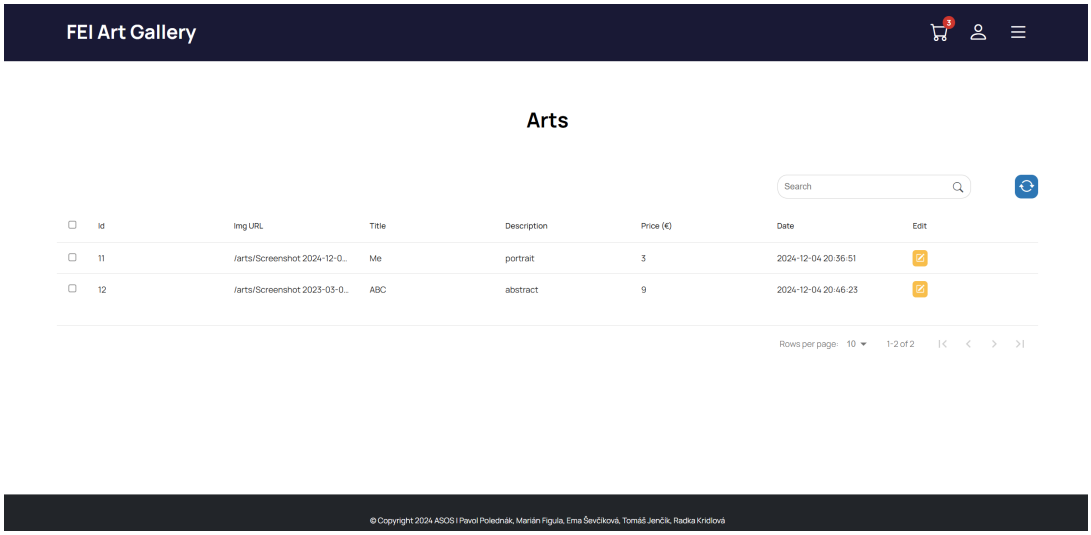

Figure 8: My posts

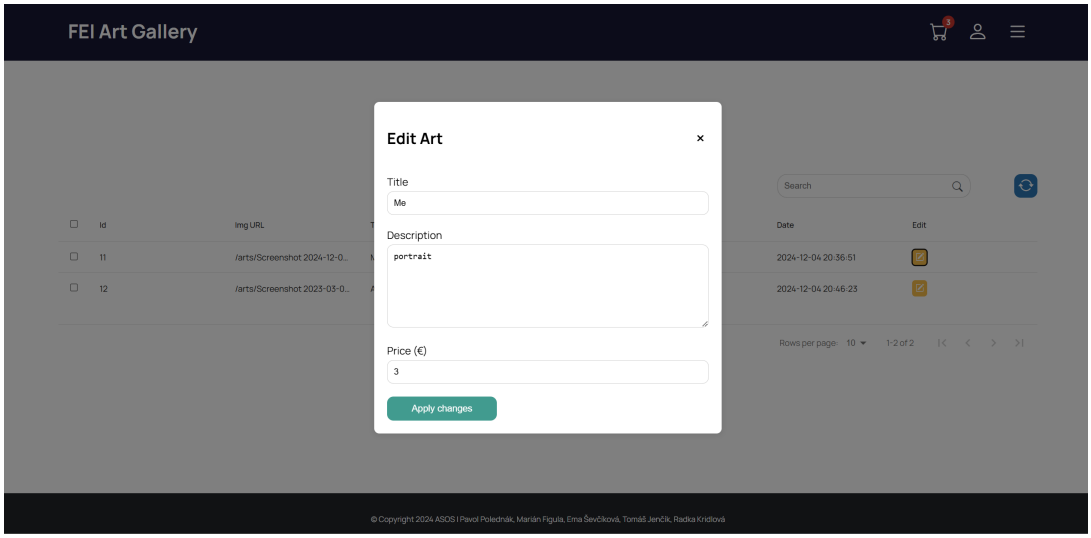

Figure 9: My posts - artwork edit

7.6 Review history

The Review history screen allows users to view all their reviews. Each review is displayed with its information. Review text and rating can be edited by clicking on the edit icon.

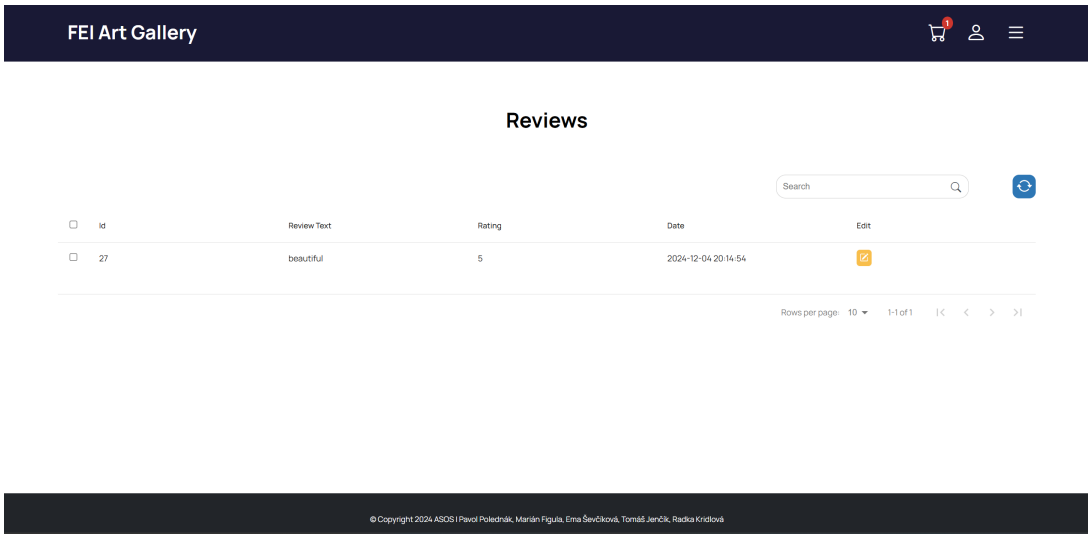

Figure 10: Review history

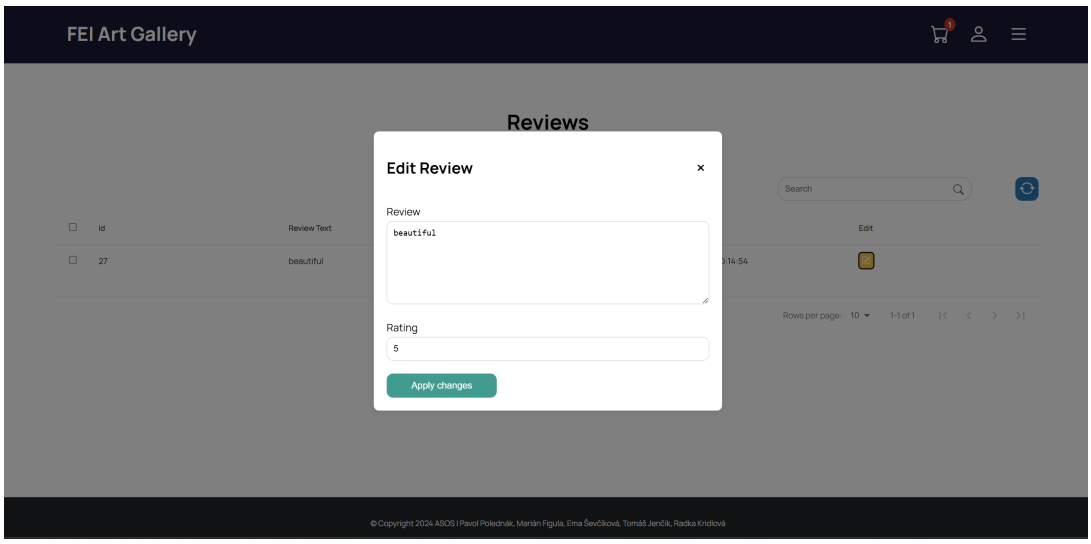

Figure 11: Review history - edit review

8 Testing

Both unit and integration tests were implemented for four different components using the Jest testing framework. Additionally, load tests were implemented to assess the performance and scalability of a web application under varying levels of user load.

8.1 Unit and integration tests

8.1.1 Search bar component

The tests for the Search bar component are implemented in the file located at src/components/searchBar/SearchBar.test.js. The description of the tests and their results are provided below:

- checks if searchBar renders with correct default placeholder *(unit test)*
  - **Result:** Success – The input renders correctly with the default placeholder.
- checks if searchBar renders with custom placeholder *(unit test)*
  - **Result:** Success – The input renders correctly with the custom placeholder.
- calls handleFilter on input change *(unit test)*
  - **Result:** Success – The handleFilter function is triggered correctly on input change.

8.1.2 Header component

The tests for the Header component are implemented in the file located at src/components/header/Header.test.js. The description of the tests and their results are provided below:

- navigates to CartSite when bi bi-cart is clicked *(integration test)*
  - **Result:** Success – Navigation to the CartSite works correctly when the "cart" icon is clicked.
- opens sidebar when click on sidebar icon *(integration test)*
  - **Result:** Success – Clicking the sidebar icon opens the sidebar.
- navigates to SignIn when bi bi-person is clicked *(integration test)*
  - **Result:** Success – Navigation to LoginSite works as expected when the "SignIn" icon is clicked.
- navigates to MainSite when the link is clicked *(integration test)*
  - **Result:** Success – The Header successfully navigates to the MainSite.

8.1.3 Login site

The tests for the Login component are implemented in the file located at src/sites/ loginSite/LoginSite.test.js. The description of the tests and their results are provided below:

- renders the login form with inputs and submit button *(unit test)*
  - **Result:** Success – The form and its elements render as expected.
- allows the user to input email and password *(unit test)*
  - **Result:** Success – The component accepts user input as intended.
- calls handleSubmit when the form is submitted *(unit test)*
  - **Result:** Success – The submit button triggers the handleSubmit function.
- displays an error message when there is an error *(integration test)*
  - **Result:** Success – The error message displays when submission fails.

8.1.4 Main site

The tests for the main site are implemented in the file located at src/sites/ mainSite/MainSite.test.js. The description of the tests and their results are provided below:

- activates the sort by price ascending button and toggles the active class *(integration test)*
  - **Result:** Success – The button toggles its active class as expected.
- button does not have "active" class when not clicked *(unit test)*
  - **Result:** Success – The button is not active by default.

8.2 Load tests

The load tests are implemented in the file load-test.yml. The description of the tests and their results are provided below:

- Steady Load Simulation
  - Simulate 10 users per second for 180 seconds: Users access the homepage (/), wait for 10 seconds, and navigate to the login page (/login).
  - **Result:** Successfully handled steady traffic with no significant degradation in response time or errors observed.
- Spike Load Simulation
  - Simulate a steady load of 10 users per second for 180 seconds, spike to 100 users per second for 30 seconds, and return to 10 users per second for 60 seconds. Users access the homepage (/), navigate to the cart page (/cart), perform a login action, and visit the profile page (/profile).

  - **Result:** Successfully handled the traffic spike. Response times increased briefly during the spike but returned to normal levels after the spike ended. No significant error rates detected.

- Incremental Ramp-Up Load
  - Gradually increase traffic from 10 users per second to 160 users per second over 5 minutes. Users access the homepage (/), navigate to the registration page (/register), and visit the cart page (/cart).
  - **Result:** Successfully managed increasing traffic with consistent response times up to 120 users per second. Minor latency observed at 160 users per second, but no critical failures occurred.

- Minimal Load with Multiple Flows
  - Simulate 10 users per second for 60 seconds, with users accessing the homepage (/), navigating to the login page (/login), and accessing the cart page (/cart).
  - **Result:** Successfully handled minimal concurrent traffic. All endpoints responded within acceptable time limits, and no errors were observed.