**Semester 1 2024**

**Jinsook Web Gallery and Admin Panel**

**Development Project**

student Number: | 27037902

**COMP.7214 Client Server Web Development**

**Rain Ruan**

Contents

[Introduction 2](#_Toc168921207)

[Project Aim Review 2](#_Toc168921208)

[Scope of the project 2](#_Toc168921209)

[Deliverables and Achievements 4](#_Toc168921210)

[Updated Api endpoints 5](#_Toc168921211)

[Dependencies 7](#_Toc168921212)

[Client-side 7](#_Toc168921213)

[Main Dependencies 7](#_Toc168921214)

[Development Dependencies 7](#_Toc168921215)

[Server-side 7](#_Toc168921216)

[Global 8](#_Toc168921217)

[Testing 9](#_Toc168921218)

[Testing plan 9](#_Toc168921219)

[Testing results 9](#_Toc168921220)

[Guide for run the application: 10](#_Toc168921221)

[Login detail for admin panel 10](#_Toc168921222)

[Port 10](#_Toc168921223)

[Start command 10](#_Toc168921224)

# Introduction

This document serves as a comprehensive support guide for the Single Page Application (SPA) project developed for Jinsook Taylor. The purpose of this document is to offer an overview of the development process, the technologies used, and the design patterns implemented. Additionally, it includes instructions for setting up and running the application, information on the project's structure, and details on the implemented features and functionalities, as well as a testing data and results.

## Project Aim Review

The aim of this project is to develop a Single Page Application (SPA) for Jinsook Taylor, a Taupo-based local business focusing on personalized art paintings, illustrations, and handmade goods. The goal is to create an online gallery that showcases Jinsook Taylor's artwork effectively, facilitating better interaction with audiences and potential buyers. This project will use the MERN stack (MongoDB, Express.js, React.js, Node.js) along with other supporting technologies to build a robust, user-friendly, and engaging website.

## Scope of the project

|  |  |
| --- | --- |
| Area | Description |
| Client-Side Development |  |
| Framework | React.js for building dynamic and interactive user interfaces |
| Features | - Online Gallery: Display high-quality images of artwork with detailed information  - Contact Form: Enable visitor inquiries and messages - Search and Filter: Allow visitors to find specific artworks based on criteria such as title, medium, or dimension  - Admin Panel: Secure login for managing the artwork database with CRUD functions |
| Server-Side Development |  |
| Framework | Node.js and Express.js for building RESTful APIs |
| Database | MongoDB database to store artwork details and user information |
| Features | - REST API: CRUD operations for managing artwork data  - Authentication: Secure user authentication using JWT and bcrypt for password hashing |
| Design and Technology Stack |  |
| Design Patterns | - Backend: Model-View-Controller (MVC) architecture  - Frontend: Model-View-Presenter (MVP) architecture |
| Additional Technologies | - JWT for secure user authentication  - bcrypt for password hashing  - Tailwind CSS for styling - Framer Motion for creating animations  - Google Cloud Storage for storing image files |

# Deliverables and Achievements

|  |  |  |
| --- | --- | --- |
| Deliverable | Description | Goal Achieved |
| SPA Website | A working Single Page Application matching the design in Assignment 2 | Fully functional SPA with the chosen framework/library |
| Source Code | Zip file containing all source code | All code files provided and organized |
| Test Data and Results | Evidence of testing, including test data and screenshots | Comprehensive testing data and results supplied |
| MongoDB Database | Working MongoDB with sufficient test data | MongoDB database setup with test data, and instructions to recreate it |
| API Implementation | Working RESTful API with Express routes | Fully functional API with appropriate CRUD operations |
| Authentication | Secure user authentication using JWT and bcrypt | Implemented secure login and password management |
| Admin Panel | Admin interface for managing artwork database | Fully functional admin panel with CRUD operations to database and to application global context. |
| Contact Form | Feature for visitors to send inquiries and messages | Implemented and functional contact form. Additional message system built also. |
| Search and Filter | Feature for searching and filtering artworks | Fully functional search and filter capabilities |
| Responsive designe | Responsive design to various screen sizes | Fully implmented |

# Updated Api endpoints

Based on discussions with the tutor, in addition to the goals defined in the previous design document from Assignment Two, I have also implemented a messaging system. This allows the website admin to receive messages sent from the contact form. To support this functionality, an additional document called 'messages' has been added to the database. Consequently, there is an update to the API endpoints.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Resource | GET | POST | PUT/PATCH | DELETE |
| /api/artworks | Retrieve an array of all artwork objects. | Add a new artwork object | N/A | N/A |
| /api/artworks/{category} | Retrieve an array of artwork objects where the category property value is one of the following:   * paintings * illustrations * childrens-books * handcrafts | N/A | N/A | N/A |
| /api/artworks/{category}/{id} | Retrieve details of a specific artwork object based on its category and ID. | N/A | Update details of a specific artwork object based on its category and ID | Delete the specific artwork object based on its category and ID |
| /api/user/login | N/A | Handle user authentication:  It accepts a username and password, verifies the request body with the ‘users’ document, and returns a JWT token upon successful authentication. | N/A | N/A |
| /api/user/change-password | N/A | N/A | Verify the old password, accept a new password, update the database with the new password, and return a JWT token upon successful password change. | N/A |
| |  | | --- | | **/api/messages** |  |  | | --- | |  |  |  | | --- | |  | | Retrieve an array of all messages. | Add a new message from the contact form | |  | | --- | | N/A | | |  | | --- | | N/A | |
| |  | | --- | | **/api/messages/{id}** | | |  | | --- | | Retrieve details of a specific message based on its ID. | | |  |  |  | | --- | --- | --- | | N/A |  |  | | Update details of a specific message based on its ID (used to mark message is unread or read) | Delete the specific message based on its ID |

# Dependencies

## Client-side

### Main Dependencies

* dateformat
* dotenv
* firebase
* framer-motion
* react
* react-dom
* react-responsive-carousel
* react-router-dom
* validator

### Development Dependencies

* @types/react
* @types/react-dom
* @vitejs/plugin-react
* autoprefixer
* eslint
* eslint-plugin-react
* eslint-plugin-react-hooks
* eslint-plugin-react-refresh
* postcss
* tailwindcss
* vite

## Server-side

* bcrypt
* cors
* dotenv
* express
* jsonwebtoken
* mongoose
* serverless-http
* validator

## Global

* nodemon
* prettier

# Testing

## Testing plan

### Client-side

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Type | Test No | Purpose | Description | Test Data | Expected Results | Actual Result |
| Functionality | 1 | Verify online gallery display | Ensure that all artwork images and details are displayed correctly in the online gallery. | N/A | All artwork images and details should be displayed correctly and match the data in the database. |  |
| 2 | Test contact form submission | Check if the contact form successfully submits messages and stores them in the database. | Name: Test User <br> Email: test@example.com <br> Message: Test message | The contact form should submit successfully, and the message should be stored in the database and visible in the admin panel. |  |
| 3 | Validate search functionality | Ensure the search feature returns correct results based on search criteria (title, medium, dimension). | Search Term: "painting" | The search results should display artworks that match the search term "painting." |  |
| 4 | Test filtering functionality | Verify that the filtering options work correctly and display relevant artworks. | Filter: Medium = "oil" | Only artworks with the medium "oil" should be displayed in the gallery. |  |
| Authentication | 5 | Verify login functionality | Ensure that the admin login works with valid credentials and prevents access with invalid credentials. | Username: admin <br> Password: correctpassword | Admin should be able to log in with valid credentials. Invalid credentials should not allow access and display an appropriate error message. |  |
| 6 | Test change password functionality | Ensure the password change feature works correctly and updates the admin password in the database. | Old Password: correct-password New Password: new-password | The admin should be able to change the password successfully, and the new password should be updated in the database. |  |
| UI/UX | 7 | Validate responsive design | Ensure the website layout is responsive and displays correctly on different screen sizes (desktop, tablet, mobile). | N/A | The website should be responsive, adjusting the layout and design elements correctly on various screen sizes. |  |
| Performance | 8 | Check page load time | Verify that the gallery and other pages load within an acceptable time frame. | N/A | All pages should load within 3 seconds. |  |
| Usability | 9 | Test navigation functionality | Ensure all navigation links and buttons work correctly and lead to the appropriate pages. | N/A | All navigation links and buttons should work correctly and lead to the correct pages. |  |
| Integration | 10 | Verify integration with backend API | Ensure that the client-side correctly interacts with the backend API for all CRUD operations. | API Requests for CRUD operations | All API requests (create, read, update, delete) should function correctly, and the client-side should display the corresponding data appropriately. |  |
| 11 | Test Create operation | Verify that new artwork can be added through the client-side interface. | Title: New Artwork, Category: Painting, Price: 100 | The new artwork should be added to the database and displayed in the gallery. |  |
| 12 | Test Read operation | Verify that artwork details can be retrieved and displayed correctly. | Artwork ID: 12345 | The details of the specified artwork should be retrieved from the database and displayed correctly in the gallery. |  |
| 13 | Test Update operation | Verify that existing artwork details can be updated through the client-side interface. | Artwork ID: 12345, Title: Updated Artwork | The specified artwork details should be updated in the database and reflected correctly in the gallery. |  |
| 14 | Test Delete operation | Verify that artwork can be deleted through the client-side interface. | Artwork ID: 12345 | The specified artwork should be deleted from the database and no longer displayed in the gallery. |  |
| Security | 15 | Test form input validation | Ensure all form inputs are validated to prevent invalid data entry. | Invalid inputs: empty fields, incorrect formats | The form should display appropriate error messages for invalid inputs and prevent submission until the data is valid. |  |

### server-side

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Type | Test No | Purpose | Description | Test Data | Expected Results | Actual Result |
| Functionality | 1 | Test Create Artwork API | Verify that a new artwork can be added through the API. | POST /api/artworks <br> { "title": "New Artwork", "category": "painting", "price": "100", "description": "A beautiful painting" } | The new artwork should be added to the database and the response should return the created artwork object. |  |
| Functionality | 2 | Test Retrieve All Artworks API | Ensure that the API returns an array of all artwork objects. | GET /api/artworks | The response should return an array of all artwork objects stored in the database. |  |
| Functionality | 3 | Test Retrieve Artwork by Category API | Ensure that the API returns artworks based on a specific category. | GET /api/artworks/paintings | The response should return an array of artworks where the category is "paintings". |  |
| Functionality | 4 | Test Retrieve Specific Artwork API | Ensure that the API returns details of a specific artwork based on its category and ID. | GET /api/artworks/paintings/12345 | The response should return the details of the specified artwork. |  |
| Functionality | 5 | Test Update Artwork API | Verify that an existing artwork can be updated through the API. | PUT /api/artworks/paintings/12345 <br> { "title": "Updated Artwork" } | The artwork details should be updated in the database and the response should return the updated artwork object. |  |
| Functionality | 6 | Test Delete Artwork API | Verify that an artwork can be deleted through the API. | DELETE /api/artworks/paintings/12345 | The specified artwork should be deleted from the database and the response should confirm the deletion. |  |
| Authentication | 7 | Test User Login API | Verify that user authentication works correctly. | POST /api/user/login <br> { "username": "admin", "password": "correctpassword" } | The response should return a JWT token upon successful authentication. |  |
| Authentication | 8 | Test Change Password API | Ensure that the password change functionality works correctly. | PUT /api/user/change-password <br> { "oldPassword": "correctpassword", "newPassword": "newpassword" } | The response should confirm the password change and return a new JWT token. |  |
| Functionality | 9 | Test Retrieve All Messages API | Verify that the API returns an array of all messages. | GET /api/messages | The response should return an array of all messages stored in the database. |  |
| Functionality | 10 | Test Create Message API | Verify that a new message can be added through the API. | POST /api/messages <br> { "name": "Test User", "email": "test@example.com", "message": "Test message" } | The new message should be added to the database and the response should return the created message object. |  |
| Functionality | 11 | Test Retrieve Specific Message API | Ensure that the API returns details of a specific message based on its ID. | GET /api/messages/12345 | The response should return the details of the specified message. |  |
| Functionality | 12 | Test Update Message API | Verify that an existing message can be updated through the API. | PUT /api/messages/12345 <br> { "message": "Updated message" } | The message details should be updated in the database and the response should return the updated message object. |  |
| Functionality | 13 | Test Delete Message API | Verify that a message can be deleted through the API. | DELETE /api/messages/12345 | The specified message should be deleted from the database and the response should confirm the deletion. |  |

## Testing screenshots

# Guide for run the application:

## Login detail for admin panel

**URL**: http://localhost:5173/dashboard

**Username**: Doug

**Password**: Doug123!

## Port setting

**Client**: 5173

**Server**:4000

## Start command

**Client**: npm run dev

**Server** npm run start