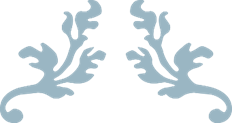
**FINAL YEAR REPORT**



Final year Project



**COMP1682 FINAL REPORT**

**The Creative Collective for Handmade Craft Tutorials Project**

**MAI DINH PHONG**

**Supervisor: PHAM THANH SON**

**Final Year Report**

**COMP1682 Final Year Project**

**Program Title: BSc Hons Computing**

**Abstract**

**Acknowledgements**

Table of Contents

**No table of contents entries found.**

1. INTRODUCTION

**1.1 Background**

**1.2 Aim**

2. OBJECTIVES

**a) To Create a Comprehensive Community Website:**   
The main goal is to create, develop, and launch an approachable website for craft aficionados. This website will act as a central hub where users can interact with other crafters, share their own products, and access a variety of craft project guides.  
  
**b) To Provide Top-Notch Craft Project Guides:**  
Another goal is to compile and offer a wide range of craft project tutorials, each with comprehensive step-by-step directions and illustrations. To accommodate users with a range of interests and ability levels, these lessons will cover a variety of crafting techniques, styles, and difficulty levels.

**c) To Create a Craft Supply Marketplace:**  
The project's goal is to develop a marketplace on the internet where people who make crafts may purchase and sell the materials and resources they need to make crafts. This marketplace will help small companies, provide consumers access to high-quality materials, and give creative a platform to sell their creations.  
  
**d) To Integrate Third-Party Services for Enhanced Functionality:**  
To improve functionality and user experience, the website will use third-party services like PayPal, Google, Azure, and safe payment processors. Among other things, these services will provide search capabilities, analytics, cloud storage, authentication, and safe payment processing.

3. APPROACH

**3.1 Justification of suitability of a methodology of a Framework followed.**

In this section, we will explain the chosen software development life cycle (SDLC) model for the project and justify its applicability, considering that it is being created independently.

**Introduce Chosen Framework**  
For this project, I've chosen the Agile Scrum technique as my preferred SDLC model. Agile Scrum is an iterative and incremental software development methodology that emphasizes flexibility, cooperation, and continual improvement. Despite being commonly associated with team-based projects, Agile Scrum may also be efficiently used to individual projects, bringing advantages like flexibility and adaptability.

**Reason 1: Flexibility and adaptability.**   
One of the key reasons for using Agile Scrum is its inherent flexibility and adaptability, which are critical for unique projects whose requirements and priorities vary over time. As the primary developer of this community website, I expect the need to respond fast to input, change priorities, and iterate on features in response to user demands and market trends. Agile Scrum's iterative methodology enables gradual development and frequent review of project goals, ensuring that the website remains in line with changing needs and objectives.

**Reason 2: Iterative Development with Continuous Feedback**   
Another important consideration in selecting Agile Scrum is the emphasis on iterative development and continual feedback. Creating a community website entail meeting the requirements and interests of users, which may change over time. By dividing down the project into digestible iterations or sprints, I can release features progressively while gathering user input early and often. This iterative feedback loop allows me to make more informed judgments, properly prioritize features, and guarantee that the website satisfies user expectations throughout the development process.

**Reason 3: Collaboration and Empowerment (Adapted to Individual Work)**   
While Agile Scrum is generally linked with team cooperation, the ideals of collaboration and empowerment may also be applied to individual projects. As the project's single developer, I will use Agile Scrum methods such as self-organization, regular reflection, and continuous improvement to guide the development process. By adopting these ideas, I hope to instill a sense of ownership, accountability, and drive, all of which are required to sustain momentum and achieve project goals on their own.

**Conclusion**   
To summarize, the Agile Scrum technique provides various benefits for individual projects, including flexibility, adaptability, iterative development, and continual feedback. I plan to create a high-quality community website that serves the demands of crafters while delivering value progressively using Agile Scrum principles and practices. Despite being a solitary undertaking, Agile Scrum offers a systematic framework for efficiently managing the development process and attaining project success.

4. LITERATURE REVIEW

**Overview**

Figure 1 - Overview.   
Introducing an Overview of System Technology.   
  
The community website for craft enthusiasts is supported by several essential technology components that work together to enhance both its usability and operation. The use of cutting-edge web development tools to guarantee scalability, responsiveness, and interaction is at the heart of this system. The website's front end is developed with React JS, a popular JavaScript toolkit recognized for its speed and flexibility in developing dynamic user interfaces. Meanwhile, the backend is driven by.NET Core, a strong platform for developing scalable and dependable server-side logic. SQL Server and MongoDB enhance data management by providing relational and NoSQL database solutions to meet a wide range of data storage requirements. These technologies constitute the core of the community website, allowing for smooth integration, fast data processing, and maximum performance.

**Technology for front-end: (React.JS)**

In this context, Technology A refers to React JS, a prominent JavaScript package used to create user interfaces. React JS was chosen because of its efficiency, reusability, and component-based design, which enables the development of dynamic and interactive user interfaces. Developers may use React JS to rapidly manage complicated UI components, handle data changes, and improve the overall performance of their websites.

**Technology for back-end: (ASP .NET Core)**

The community website's backend operations are powered by.NET Core, a flexible and cross-platform framework for developing scalable and powerful online apps. .NET Core includes a complete set of tools and frameworks for building server-side logic, processing data, and managing system resources effectively. With its performance, security, and scalability features,.NET Core acts as the backbone of the website's backend architecture, providing dependable and high-performance operation.

**Technology for Database: (SQL Server)**

SQL Server is used as a relational database management system to store and manage structured data on the community website. SQL Server's comprehensive transaction management, data integrity, and scalability characteristics make it a dependable and efficient choice for handling relational data requirements. SQL Server guarantees data integrity, dependability, and accessibility, so efficiently meeting the website's data storage and retrieval requirements.

**Technology Database: (MongoDB)**

Technology D represents MongoDB, a NoSQL database management system renowned for its adaptability, scalability, and performance. MongoDB was chosen because of its document-oriented data model, which enables data storage and retrieval in JSON-like documents. This schema-less method allows developers to respond fast to changing data requirements and expand horizontally to handle increasing data quantities. MongoDB enables developers to effectively store and handle data, hence enabling the community website's dynamic and developing nature.

**Market analysis**

**Survey Introduction / Research Activity / Research Method**

To gain insights into the market landscape and understand the preferences and behaviors of crafting enthusiasts, a comprehensive survey and research activities were conducted. The survey aimed to gather data on various aspects, including user demographics, crafting interests, preferred online platforms, and purchasing behaviors.

In addition to the survey, qualitative research approaches such as interviews and focus group discussions were used to acquire detailed information on user preferences and pain areas. These actions contributed to the identification of developing trends, user motivations, and opportunities for improvement in existing crafting platforms.

**Result Analysis**

Demographics: Many study respondents were women aged 16 to 45, indicating that this is the predominant demographic for crafters. However, there was a considerable representation of male members, demonstrating the inclusive character of the creative community.

Crafting hobbies: The poll found that respondents had a wide range of crafting hobbies, including sewing, knitting, crocheting, carpentry, and jewelry making. This diversity reflects the wide attraction of crafting as a creative outlet, emphasizing the necessity for a community platform that caters to a variety of crafting interests.

Preferred Online sites: While respondents were pleased with current crafting sites such as Etsy and Pinterest, several pointed out drawbacks such as a lack of detailed lessons, restricted community participation capabilities, and difficulty acquiring high-quality crafting supplies.

Purchasing Behaviors: According to the poll, crafting aficionados frequently confront difficulties in obtaining high-quality crafting equipment and materials, with many expressing displeasures at restricted availability and high prices. Furthermore, respondents expressed a readiness to support independent producers and small companies through their purchase habits.

**Conclusion**

In conclusion, the market analysis gave useful information about the requirements, interests, and behaviors of craft aficionados. Understanding the target audience's demographics, crafting hobbies, preferred online platforms, and purchasing patterns allows us to create a community website that efficiently meets their needs while also providing a beneficial platform for sharing, learning, and interacting with other crafters.

**User Persona**

FIGURE 2: USER PERSONA   
Persona Explanation:   
  
Based on the market study findings, a user persona was developed to reflect the average crafts hobbyist. The user persona captures crucial demographic information, including interests, pain points, and motivations, to provide a comprehensive knowledge of the target audience's requirements and preferences. This persona is a vital resource throughout the development process, driving choices about product priority, user experience design, and content creation.

5. LEGAL, SOCIAL ETHICAL AND PROFESSIONAL ISSUES AND CONSIDERATIONS

In the establishment and administration of the community website for crafting enthusiasts, it is critical to address many legal, social, ethical, and professional aspects to assure compliance, encourage user trust, and preserve ethical standards.

**5.1 Legal Points to Remember**   
Obligations to Intellectual Property:   
**Copyrights:** Ensuring that all information published on the site, such as guides, pictures, and user-generated material, abides by copyright regulations and protects the producers' intellectual property rights.   
**Trademark:** Preventing unlawful use of third-party trademarks or logos and taking precautions to safeguard the website's identity and branding.   
Privacy and Data Protection:   
GDPR Adherence: Respecting the rights and privacy of users by following the General Data Protection Regulation (GDPR) requirements when gathering, storing, and processing user data.   
Putting in place a transparent and thorough privacy policy that describes how user data is gathered, utilized, and safeguarded on the platform.

**User Agreement and Terms of Service:**   
Creating terms of service agreements that specify user rights, obligations, and appropriate conduct and regulate how the website is used.   
User Contract: To reduce legal risks and conflicts, users must accept the terms of service before they may access and use the site.

**5.2 Social Aspects Community Standards:**   
  
defining explicit community rules of behavior and principles to promote a welcoming and constructive online community.   
putting in place rules and procedures for moderation to deal with offensive or dangerous behavior and material in the community.

Diversity & Inclusion: accepting people with a range of identities, ethnicities, and histories to foster diversity and inclusion within the crafting community.   
establishing channels for underrepresented groups to use the platform to express their viewpoints, experiences, and creative abilities.

**5.3 Ethics-Related Matters**

**Integrity and Transparency:**   
being open and honest with users about the platform's workings, including its income streams, business plan, and any sponsored material or adverts.   
Maintaining honesty and integrity in all dealings with partners, stakeholders, and users to establish credibility and confidence.

**User Safety and Empowerment:**giving consumers the ability to manage their privacy settings and personal data, as well as the choice to refuse data collection or sharing.   
putting an emphasis on user safety and wellbeing by putting policies in place to combat cyberbullying, harassment, and other types of online abuse.

**5.4 Professional Points to Remember**

Professionalism and Accountability: Exhibiting professionalism in the development of content, in user interactions, and in the resolution of complaints or disputes, among other facets of website administration.   
keeping oneself responsible for moral principles and industry best practices to preserve the platform's reputation and make constant improvements.   
Constant Learning and Development: Making a commitment to lifelong learning and development to be up to date on the legal, social, and ethical concerns that are changing in the online world.   
Participating in conferences, industry forums, and professional networks to network with experts and peers to share information, ideas, and best practices.

**5.5 Conclusion:**

The significance of considering legal, social, ethical, and professional factors in the creation and management of the community website is emphasized in this section. Through proactive resolution of these concerns, the platform may cultivate a good community culture, provide a secure and welcoming environment for aficionados of crafts.

6. REQUIREMENTS

**Analysis of requirements**

The requirements study for the community website for crafting lovers includes a detailed evaluation of user needs and expectations, as well as the functionality and features required to effectively satisfy those goals. The key requirements include:

**Comprehensive lessons:** Users expect to have access to a wide choice of lessons covering everything from basic to sophisticated crafting skills. These lessons should include clear step-by-step instructions, as well as photos or videos to help students understand.

**Interactive Community tools:** The website should have tools like forums, groups, and comments to encourage user engagement and cooperation. This stimulates information exchange, feedback, and networking among the crafting community.

**Integrated Marketplace:** A marketplace component is necessary for users to purchase and sell crafting supplies, materials, and finished goods. This marketplace should have a simple interface, safe payment methods, and support for independent producers and small creators.

**User Profiles and Preferences:** Users should be able to build individualized profiles, adjust their preferences, and follow their progress across the platform. This increases user engagement and fosters a feeling of belonging and identity within the community.

**Existing Solutions**

**Conclusion**

BUSINESS REQUIREMENTS

**Overall Picture**

**Functional Requirements with MoSCoW prioritization**

**7.3 Non-functional Requirements**

ANALYSIS AND DESIGN

**Context Diagram**

**Container Diagram**

**Assumptions**

**3rd party services**

**Programming language**

**Front-end technology**

**8.7 Back-end technology**

**Database**

**Mobile app (optional)**

**8.10 Operation system**

**Hosting (AWS, GCloud, Heroku, Azure …)**

**Component Diagram**

**8.3.1 API Endpoints**

**Code Diagram**

**Use case diagram**

**Entity relationship diagrams**

**Class diagram**

**Activity diagram**

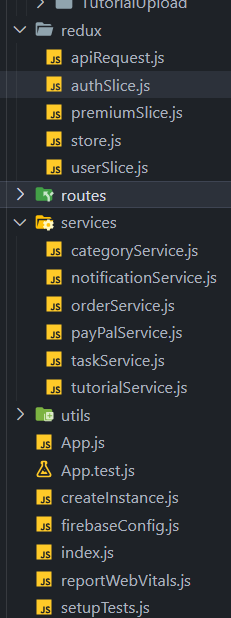
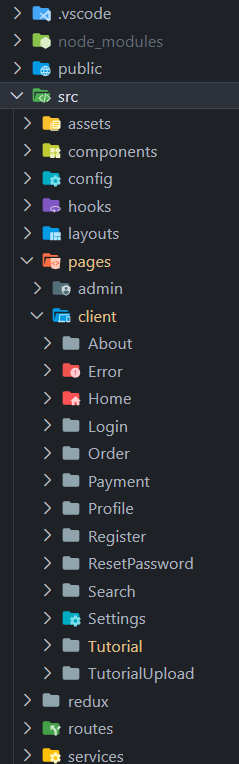
IMPLEMENTATION

11.1. Database

**11.2. Project Overview**

**11.3. Front-end**

**a) Project folder structure**



**Source code samples.**

**Category Api services**

import \* as httpRequest from '~/utils/httpRequest';

export const getAllCategories = async () => {

    try {

        const response = await httpRequest.get('categories');

        return response;

    } catch (error) {

        throw new Error('Error fetching categories'); // Throw an error if the request fails

    }

};

**Order call api service:**// orderServices.js

import \* as httpRequest from '~/utils/httpRequest';

export const createOrder = async (orderRequest) => {

    try {

        const res = await httpRequest.post(`/orders/${orderRequest.userId}`, orderRequest);

        return res;

    } catch (error) {

        console.log(error);

        throw error;

    }

};

export const getOrders = async (userId) => {

    try {

        const res = await httpRequest.get(`/orders/all/${userId}`);

        return res;

    } catch (error) {

        console.error('Error fetching orders:', error);

        throw error;

    }

};

export const getOrderById = async (orderId) => {

    try {

        const res = await httpRequest.get(`/orders/${orderId}`);

        console.log(res);

        return res;

    } catch (error) {

        console.error('Error fetching order:', error);

        throw error;

    }

};

**Paypal service:**//Paypal service

import axios from 'axios';

const apiClient = axios.create({

    baseURL: 'https://localhost:5001/api/', // Update the base URL to your backend API

});

export const upgradeToPremium = async (userId) => {

    try {

        const response = await axios.apiClient(`/users/${userId}/upgrade-to-premium`);

        return response.data;

    } catch (error) {

        throw new Error('Failed to upgrade user to premium: ' + error.message);

    }

};

export const createOrder = async (amount) => {

    try {

        // Convert the amount to a number

        const numericAmount = parseFloat(amount);

        // Check if the conversion is successful

        if (isNaN(numericAmount)) {

            throw new Error('Invalid amount');

        }

        const response = await axios.apiClient('/paypal/create-order', { amount: numericAmount });

        console.log(response.data);

        return response.data.orderId;

    } catch (error) {

        throw new Error('Failed to create order.');

    }

};

export const capturePayment = async (orderId) => {

    try {

        await axios.post('/paypal/capture-payment', { orderId });

        return true;

    } catch (error) {

        throw new Error('Failed to capture payment.');

    }

};

export const connectPaypal = async (userId, paypalEmail, paypalFirstName, paypalLastName) => {

    try {

        await apiClient.post('/PayPal/connect-paypal', {

            userId,

            paypalEmail,

            paypalFirstName,

            paypalLastName,

        });

    } catch (error) {

        throw new Error('Failed to connect PayPal.');

    }

};

export const sendPayment = async (paypalEmail, amount) => {

    try {

        // Make a request to your backend to send payment to the provided PayPal email with the specified amount

        const res = await apiClient.post(`/PayPal/send-payment?recipientEmail=${paypalEmail}&amount=${amount}`);

        return res;

    } catch (error) {

        console.log(error);

        throw error; // Re-throw the error to handle it elsewhere

    }

};

**Task service:**import \* as httpRequest from '~/utils/httpRequest';

export const getAllTasks = async () => {

    try {

        const res = await httpRequest.get('task');

        return res;

    } catch (error) {

        console.log(error);

    }

};

export const createTask = async (formData) => {

    try {

        const res = await httpRequest.post('task', formData);

        return res;

    } catch (error) {

        console.log(error);

    }

};

export const getTaskById = async (id) => {

    try {

        const res = await httpRequest.getById('task', {

            params: {

                id,

            },

        });

        return res;

    } catch (error) {

        console.log(error);

    }

};

export const updateTask = async (formData, id) => {

    try {

        const res = await httpRequest.update('task', formData, {

            params: {

                id,

            },

        });

        return res;

    } catch (error) {

        console.log(error);

    }

};

export const deleteTask = async (id) => {

    try {

        const res = await httpRequest.deleteById('task', {

            params: {

                id,

            },

        });

        return res;

    } catch (error) {

        console.log(error);

    }

};

**Tutorial service:**// tutorialService.js

import \* as httpRequest from '~/utils/httpRequest';

export const getAllTutorials = async () => {

    try {

        const res = await httpRequest.get('tutorials');

        const tutorialsWithDetails = res.map(async (tutorial) => {

            const details = await getTutorialDetails(tutorial.id);

            return { ...tutorial, ...details };

        });

        return await Promise.all(tutorialsWithDetails);

    } catch (error) {

        console.log(error);

    }

};

export const getTutorialDetails = async (id) => {

    try {

        const res = await httpRequest.getById(`tutorials/${id}`);

        return { likes: res.likes, comments: res.comments };

    } catch (error) {

        console.log(error);

        throw error; // Re-throw the error to handle it elsewhere

    }

};

export const getTutorialById = async (id) => {

    try {

        const res = await httpRequest.getById(`tutorials/${id}`);

        return res;

    } catch (error) {

        console.log(error);

        throw error; // Re-throw the error to handle it elsewhere

    }

};

export const createTutorial = async (tutorialData) => {

    try {

        const res = await httpRequest.post('tutorials', tutorialData);

        return res;

    } catch (error) {

        console.log(error);

    }

};

export const updateTutorial = async (id, tutorialData) => {

    try {

        const res = await httpRequest.update('tutorials', tutorialData, {

            params: {

                id,

            },

        });

        return res;

    } catch (error) {

        console.log(error);

    }

};

export const deleteTutorial = async (id) => {

    try {

        const res = await httpRequest.deleteById('tutorials', {

            params: {

                id,

            },

        });

        return res;

    } catch (error) {

        console.log(error);

    }

};

export const addLikeToTutorial = async (tutorialId, userId) => {

    try {

        const response = await fetch(`/api/tutorials/${tutorialId}/likes`, {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json',

            },

            body: JSON.stringify({ userId }),

        });

        if (!response.ok) {

            throw new Error(`Failed to add like: ${response.statusText}`);

        }

        // Check if the response body is not empty before parsing as JSON

        const responseData = await response.text();

        return responseData ? JSON.parse(responseData) : null;

    } catch (error) {

        console.error('Error adding like:', error);

        throw error;

    }

};

export const removeLikeFromTutorial = async (tutorialId, likeId) => {

    try {

        const response = await fetch(`/api/tutorials/${tutorialId}/likes/${likeId}`, {

            method: 'DELETE',

            headers: {

                'Content-Type': 'application/json',

            },

        });

        if (!response.ok) {

            const errorData = await response.json().catch(() => ({}));

            throw new Error(`Failed to remove like: ${errorData.message || 'Unknown error'}`);

        }

        // Check if the response body is not empty before parsing as JSON

        const responseData = await response.text();

        return responseData ? JSON.parse(responseData) : null;

    } catch (error) {

        console.error('Error removing like:', error);

        throw error;

    }

};

export const addCommentToTutorial = async (tutorialId, commentContent, currentUserID) => {

    try {

        // Simulate an API call (replace this with your actual API endpoint)

        const response = await fetch(`/api/tutorials/${tutorialId}/comments`, {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json',

            },

            body: JSON.stringify({ content: commentContent, userId: currentUserID }),

        });

        if (!response.ok) {

            throw new Error('Failed to add comment');

        }

        // In a real scenario, you might return the added comment from the API response

        const addedComment = {

            id: new Date().getTime().toString(),

            content: commentContent,

            timestamp: new Date().toLocaleString(),

        };

        return addedComment;

    } catch (error) {

        console.error('Error adding comment:', error);

        throw error;

    }

};

export const deleteCommentFromTutorial = async (tutorialId, commentId) => {

    try {

        const response = await fetch(`/api/tutorials/${tutorialId}/comments/${commentId}`, {

            method: 'DELETE',

            headers: {

                'Content-Type': 'application/json',

            },

        });

        if (!response.ok) {

            const errorData = await response.json().catch(() => ({}));

            throw new Error(`Failed to delete comment: ${errorData.message || 'Unknown error'}`);

        }

        // Check if the response body is empty

        const responseData = await response.text();

        return responseData ? JSON.parse(responseData) : null;

    } catch (error) {

        console.error('Error deleting comment:', error);

        throw error;

    }

};

export const searchTutorials = async (searchTerm) => {

    try {

        const res = await httpRequest.get(`tutorials/search?searchTerm=${searchTerm}`);

        return res;

    } catch (error) {

        console.log(error);

        throw error; // Re-throw the error to handle it elsewhere

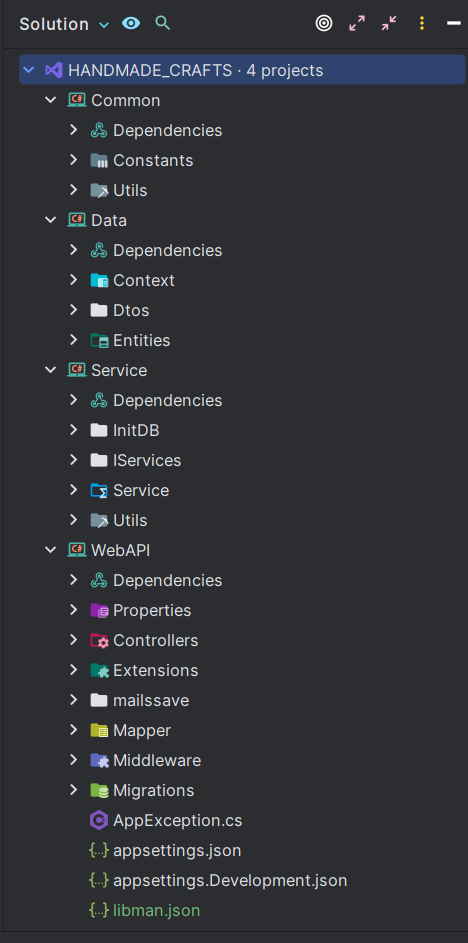
    }

};

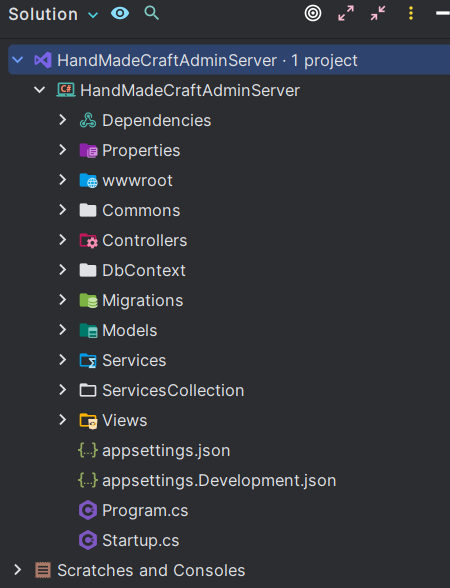
**11.4. Back-end**

**a) Project folder structure**

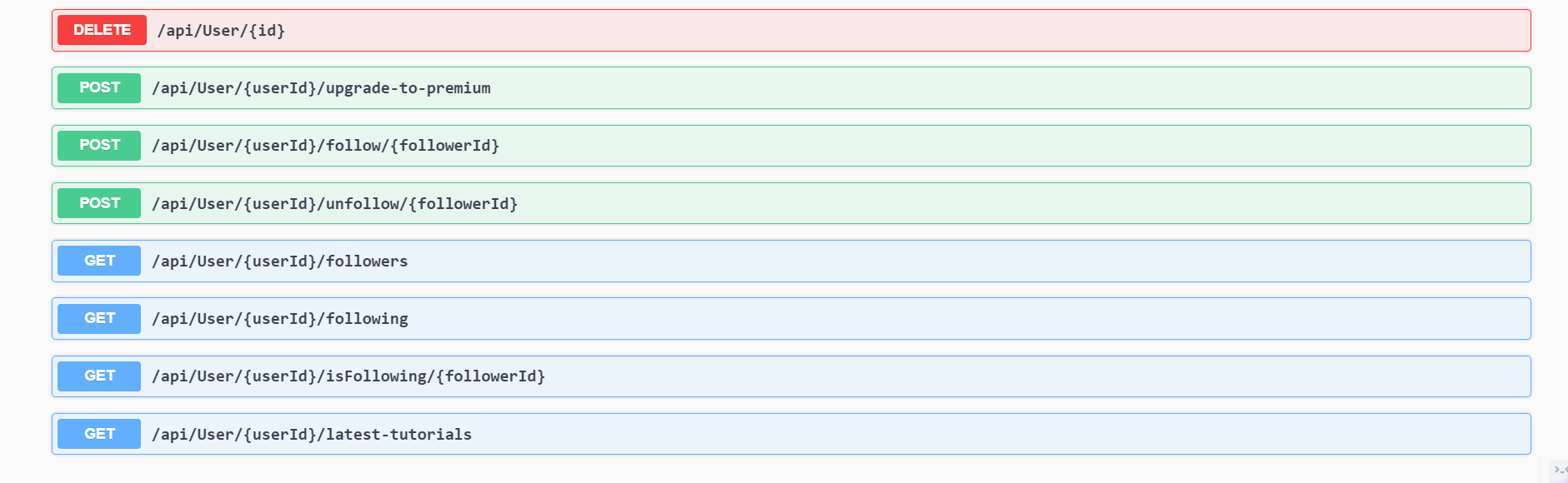
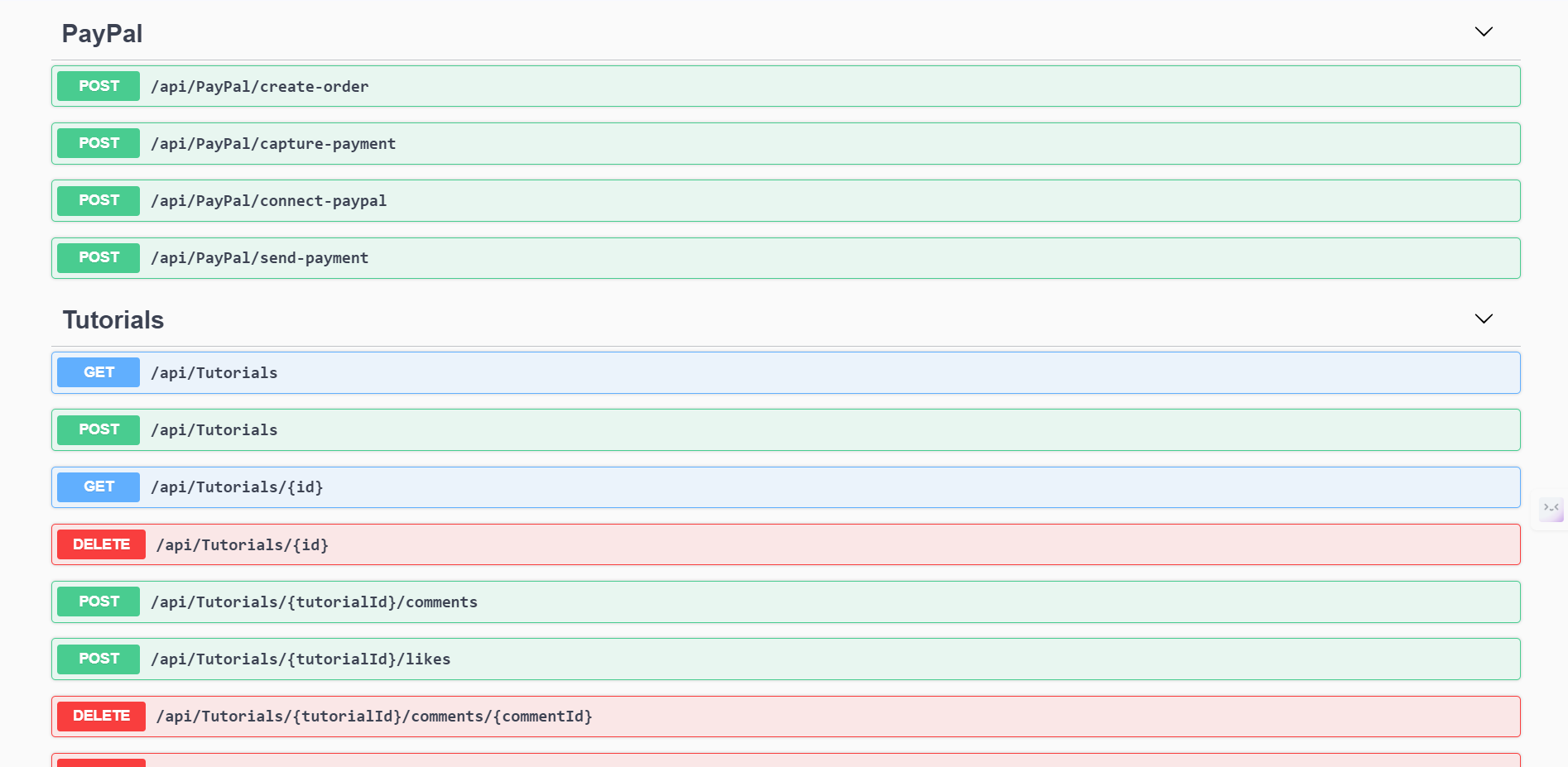
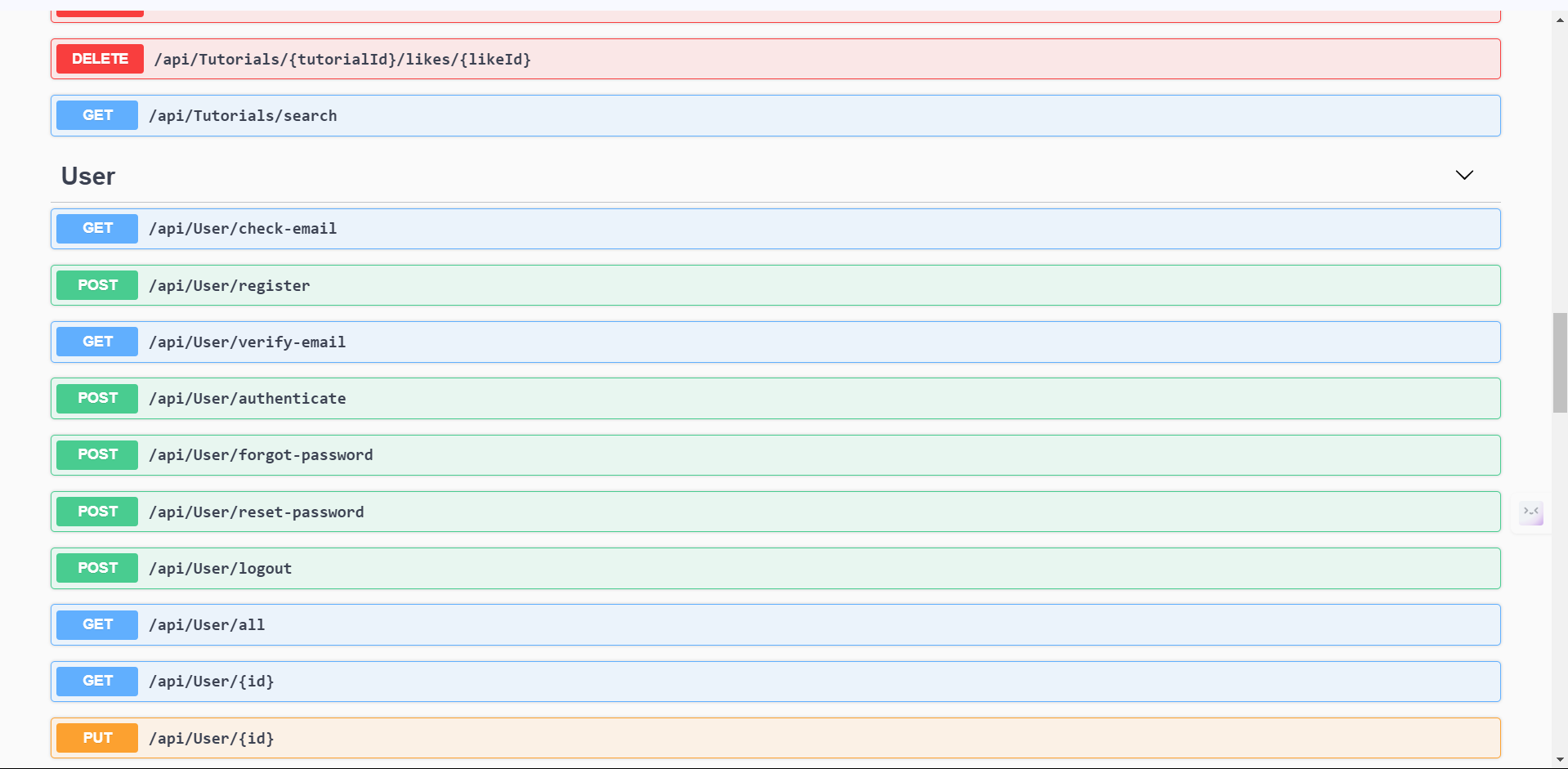
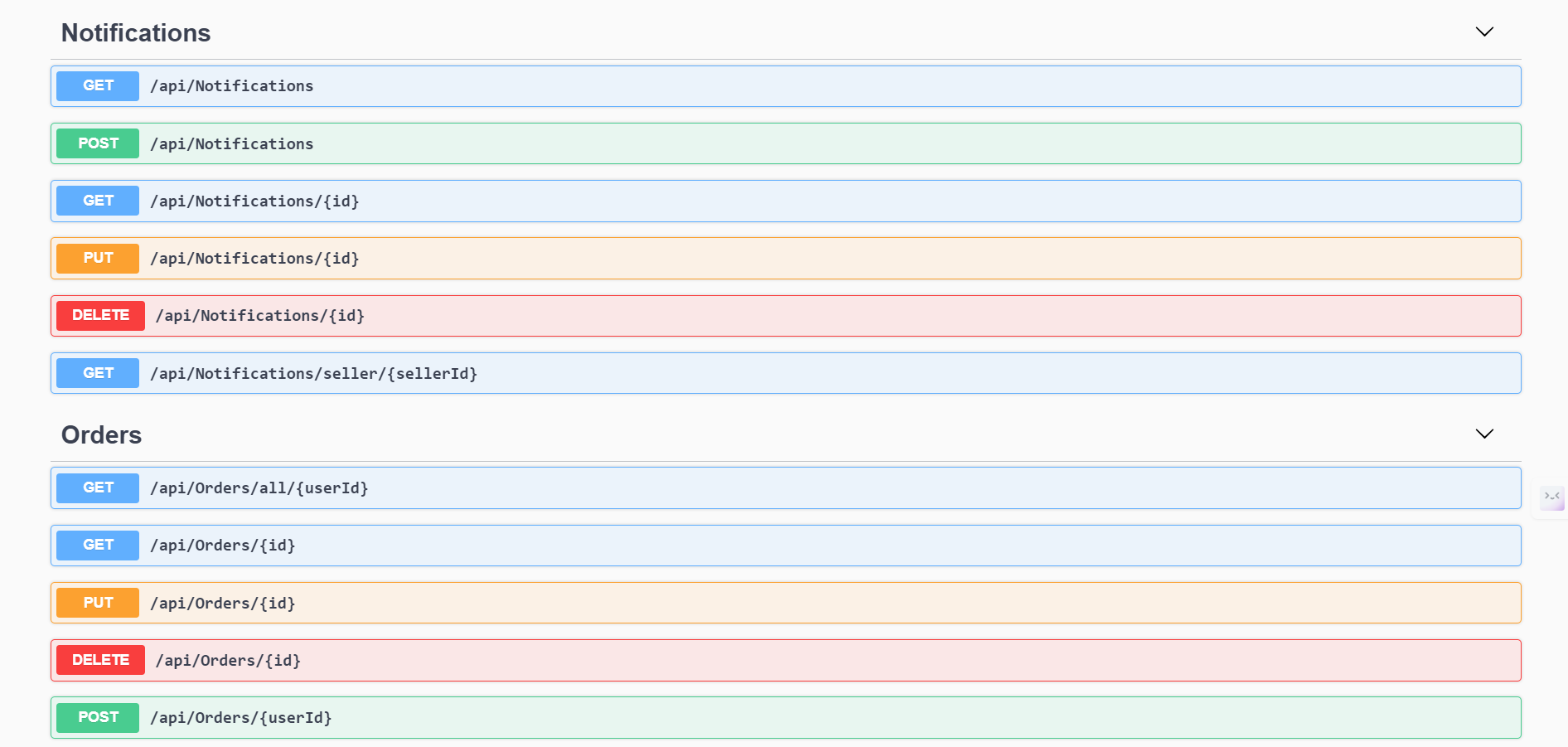
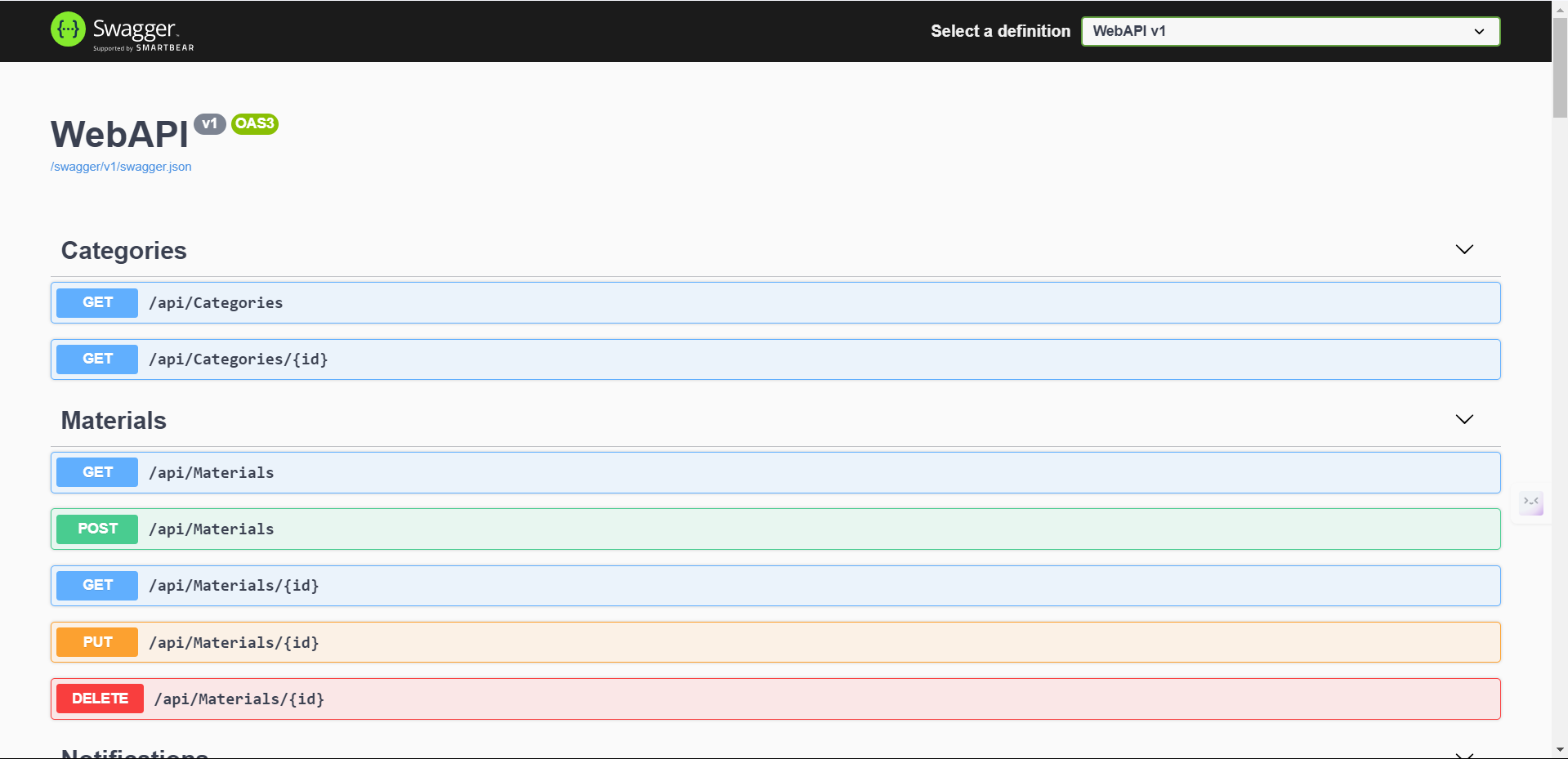
**a1) Client Server folder structure**



**a2) Admin server folder structure**



**b) Swagger documentation.**



**source code samples.**

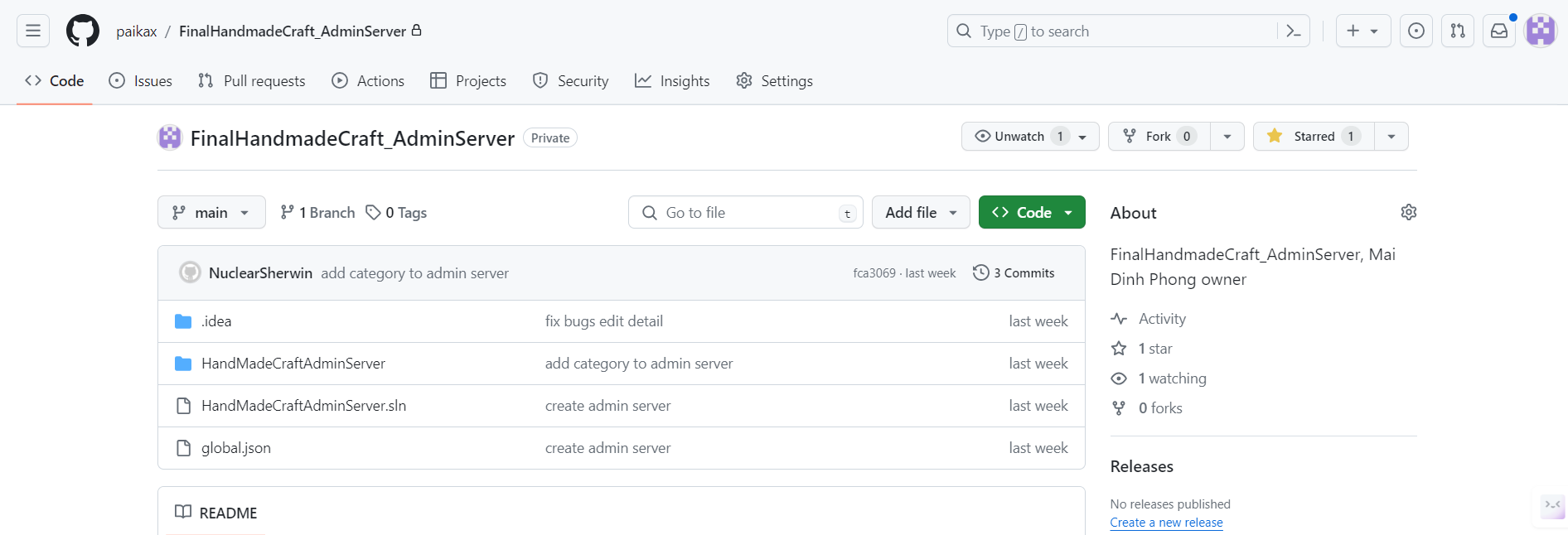
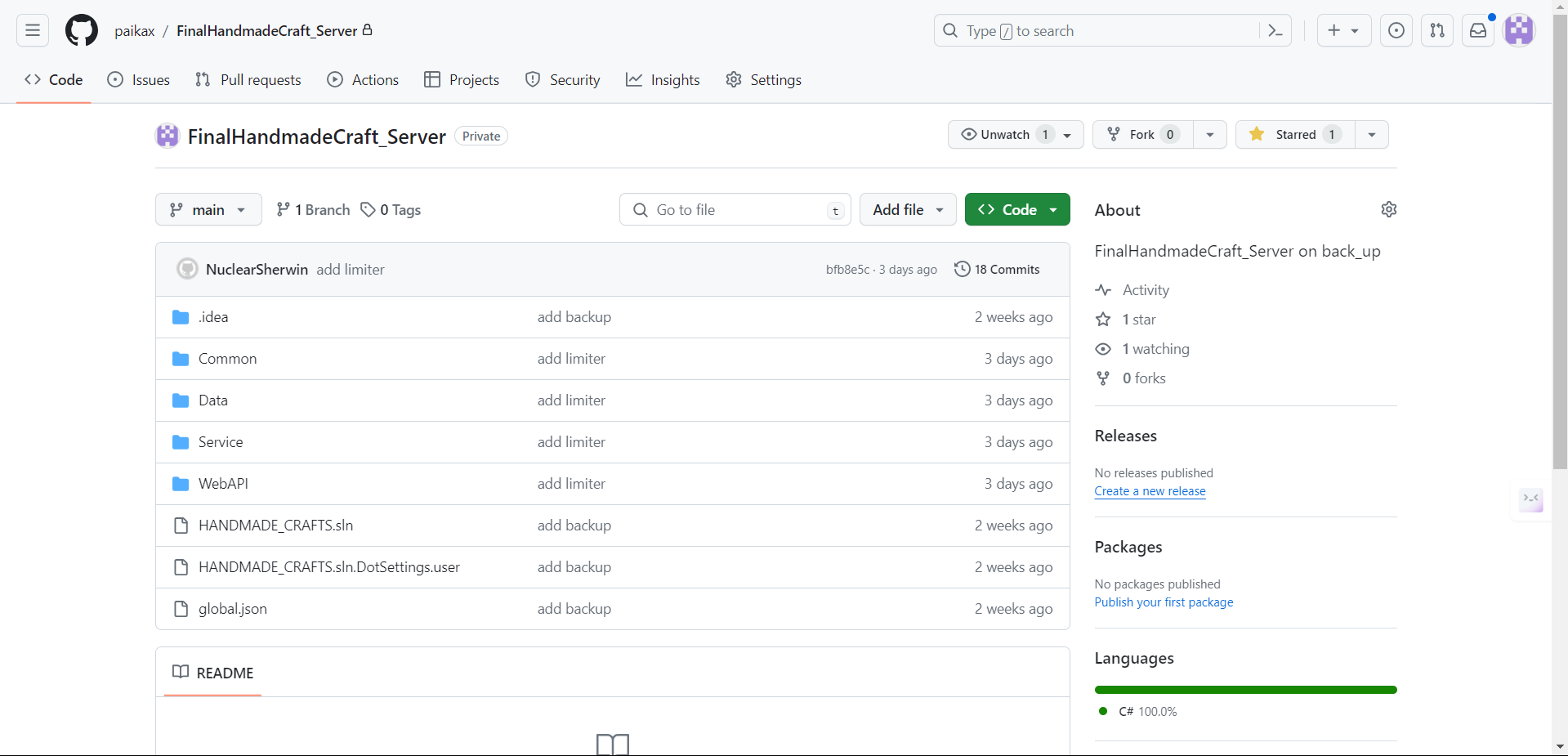
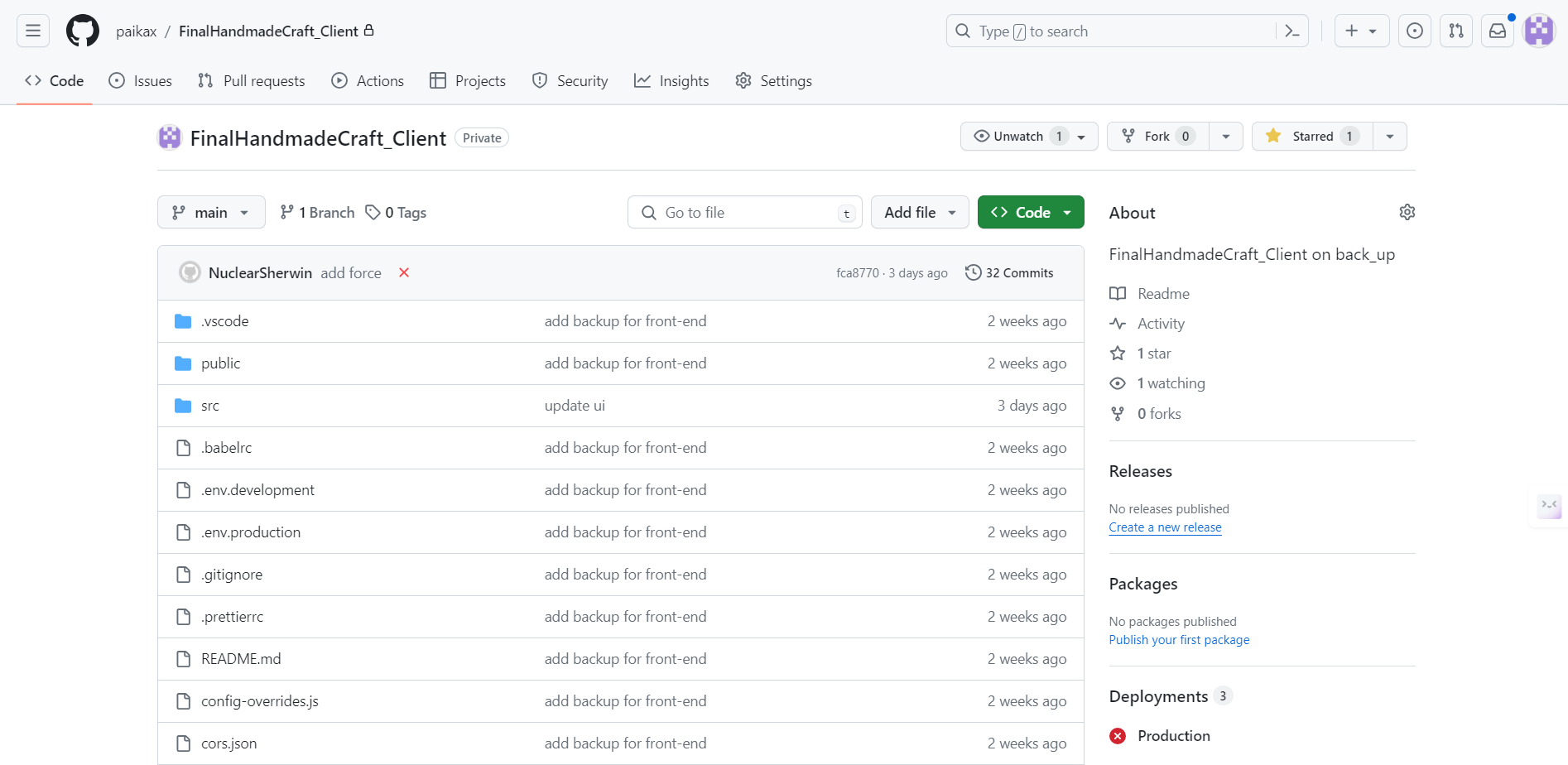
**UsersController code:**using System;  
using System.Threading.Tasks;  
using AutoMapper;  
using Data.Entities.User;  
using Data.ViewModels.User;  
using Microsoft.AspNetCore.Mvc;  
using Service.IServices;  
using Microsoft.AspNetCore.Http;  
  
namespace WebAPI.Controllers  
{  
 [ApiController]  
 [Route("api/**[controller]**")]  
 public class UserController : ControllerBase  
 {  
 private readonly IUserService \_userService;  
 private readonly IMapper \_mapper;  
  
 public UserController(IUserService userService, IMapper mapper)  
 {  
 \_mapper = mapper;  
 \_userService = userService;  
 }  
   
 [HttpGet("check-email")]  
 public async Task<IActionResult> CheckEmail(string email)  
 {  
 var emailExists = await \_userService.EmailExists(email);  
 return Ok(new { EmailExists = emailExists });  
 }  
  
  
 [HttpPost("register")]  
 public async Task<IActionResult> Register([FromBody] RegisterRequest model)  
 {  
 try  
 {  
 var user = \_mapper.Map<User>(model);  
 await \_userService.Register(user, model.Password, Request.Headers["origin"]);  
 return Ok(new { Message = "User registered successfully. Verification email has been sent." });  
 }  
 catch (Exception ex)  
 {  
 return BadRequest(new { Message = ex.Message });  
 }  
 }  
   
 [HttpGet("verify-email")]  
 public async Task<IActionResult> VerifyEmail(string token)  
 {  
 try  
 {  
 var user = await \_userService.VerifyEmail(token);  
 return Ok(new { Message = "User email verified successfully." });  
 }  
 catch (Exception ex)  
 {  
 return BadRequest(new { Message = ex.Message });  
 }  
 }  
   
   
   
 [HttpPost("authenticate")]  
 public async Task<IActionResult> Authenticate([FromBody] AuthenticationRequest model)  
 {  
 try  
 {  
 var response = await \_userService.Authenticate(model, IpAddress());  
 return Ok(response);  
 }  
 catch (Exception ex)  
 {  
 return BadRequest(new { Message = ex.Message });  
 }  
 }  
   
 [HttpPost("forgot-password")]  
 public async Task<IActionResult> ForgotPassword(ForgotPasswordRequest model)  
 {  
 await \_userService.ForgotPassword(model, Request.Headers["origin"]);  
 return Ok(new { message = "Please check your email for password reset instructions" });  
 }  
   
 [HttpPost("reset-password")]  
 public async Task<IActionResult> ResetPassword(ResetPasswordRequest model)  
 {  
 await \_userService.ResetPassword(model);  
 return Ok(new { message = "Password reset successful, now you can login" });  
   
 }  
   
 [HttpPost("logout")]  
 public async Task<IActionResult> Logout(string refreshToken)   
 {  
 await \_userService.Logout(refreshToken);  
 return Ok(new { message = "Logged out successfully" });  
 }  
   
   
  
 [HttpGet("all")]  
 public async Task<IActionResult> GetAllUsers()  
 {  
 try  
 {  
 var users = await \_userService.GetAll();  
 return Ok(users);  
 }  
 catch (Exception ex)  
 {  
 return BadRequest(new { Message = ex.Message });  
 }  
 }  
  
 [HttpGet("{id}")]  
 public async Task<IActionResult> GetUserById(string id)  
 {  
 try  
 {  
 var user = await \_userService.GetById(id);  
 return Ok(user);  
 }  
 catch (Exception ex)  
 {  
 return BadRequest(new { Message = ex.Message });  
 }  
 }  
  
 [HttpPut("{id}")]  
 public async Task<IActionResult> UpdateUser(string id, [FromBody] UpdateRequest updateRequest)  
 {  
 try  
 {  
 await \_userService.Update(id, updateRequest);  
 return Ok(new { Message = "User updated successfully" });  
 }  
 catch (Exception ex)  
 {  
 return BadRequest(new { Message = ex.Message });  
 }  
 }  
  
 [HttpDelete("{id}")]  
 public async Task<IActionResult> DeleteUser(string id)  
 {  
 try  
 {  
 await \_userService.Delete(id);  
 return Ok(new { Message = "User deleted successfully" });  
 }  
 catch (Exception ex)  
 {  
 return BadRequest(new { Message = ex.Message });  
 }  
 }  
 // helper method  
 private void SetTokenCookie(string token)  
 {  
 var cookieOptions = new CookieOptions()  
 {  
 HttpOnly = true,  
 Expires = DateTime.UtcNow.AddDays(7)  
 };  
   
 Response.Cookies.Append("refreshToken", token, cookieOptions);  
 }  
   
 private string IpAddress()  
 {  
 if (Request.Headers.ContainsKey("X-Forwarded-for"))  
 return Request.Headers["X-Forwarded-For"];  
 else  
 {  
 return HttpContext.Connection.RemoteIpAddress.MapToIPv4().ToString();  
 }  
 }  
   
 [HttpPost("{userId}/upgrade-to-premium")]  
 public async Task<IActionResult> UpgradeToPremium(string userId)  
 {  
 try  
 {  
 // Pass userId to the service method for upgrading to premium  
 await \_userService.UpgradeToPremium(userId);  
 return Ok(new { Message = "User upgraded to premium successfully." });  
 }  
 catch (Exception ex)  
 {  
 return BadRequest(new { Message = ex.Message });  
 }  
 }  
  
 [HttpPost("{userId}/follow/{followerId}")]  
 public async Task<IActionResult> Follow(string userId, string followerId)  
 {  
 if (userId == followerId)  
 {  
 return BadRequest("Users cannot follow themselves.");  
 }  
  
 try  
 {  
 await \_userService.FollowUser(followerId, userId);  
 return Ok(new { Message = "Followed successfully" });  
 }  
 catch (Exception ex)  
 {  
 return BadRequest(new { Message = ex.Message });  
 }  
 }  
  
  
 [HttpPost("{userId}/unfollow/{followerId}")]  
 public async Task<IActionResult> Unfollow(string userId, string followerId)  
 {  
 await \_userService.UnfollowUser(followerId, userId);  
 return Ok();  
 }  
  
 [HttpGet("{userId}/followers")]  
 public async Task<IActionResult> GetFollowers(string userId)  
 {  
 var followers = await \_userService.GetFollowers(userId);  
 return Ok(followers);  
 }  
  
 [HttpGet("{userId}/following")]  
 public async Task<IActionResult> GetFollowing(string userId)  
 {  
 var following = await \_userService.GetFollowing(userId);  
 return Ok(following);  
 }  
   
 [HttpGet("{userId}/isFollowing/{followerId}")]  
 public async Task<IActionResult> IsFollowing(string userId, string followerId)  
 {  
 var isFollowing = await \_userService.IsFollowing(userId, followerId);  
 return Ok(isFollowing);  
 }  
   
 [HttpGet("{userId}/latest-tutorials")]  
 public async Task<IActionResult> GetLatestUserTutorials(string userId, int count = 4)  
 {  
 try  
 {  
 var tutorials = await \_userService.GetLatestTutorialsByUser(userId, count);  
 return Ok(tutorials);  
 }  
 catch (Exception ex)  
 {  
 return BadRequest(new { Message = ex.Message });  
 }  
 }  
  
   
 }  
}

**TutorialsController:**using System;  
using System.Collections.Generic;  
using System.Threading.Tasks;  
using Data.Dtos.Tutorial;  
using Data.Entities.Comment;  
using Data.Entities.Tutorial;  
using Microsoft.AspNetCore.Mvc;  
using Microsoft.Extensions.Caching.Memory;  
using Service.IServices;  
  
namespace WebAPI.Controllers  
{  
 [Route("api/**[controller]**")]  
 [ApiController]  
 public class TutorialsController : ControllerBase  
 {  
 private readonly ITutorialService \_tutorialService;  
  
 public TutorialsController(ITutorialService tutorialService)  
 {  
 \_tutorialService = tutorialService;  
 }  
  
 [HttpGet]  
 public async Task<ActionResult<List<TutorialDTO>>> GetTutorials()  
 {  
 var tutorials = await \_tutorialService.GetAllTutorials();  
 return Ok(tutorials);  
 }  
  
 [HttpGet("{id}")]  
 public async Task<ActionResult<TutorialDTO>> GetTutorial(string id)  
 {  
 var tutorial = await \_tutorialService.GetTutorialById(id);  
  
 if (tutorial == null)  
 {  
 return NotFound();  
 }  
  
 return Ok(tutorial);  
 }  
  
 [HttpPost]  
 public async Task<ActionResult<TutorialDTO>> PostTutorial([FromBody] TutorialCreateRequest model)  
 {  
 // Perform the tutorial creation  
 var createdTutorial = await \_tutorialService.CreateTutorial(model);  
  
 // Invalidate the cache for the list of tutorials  
 // var cacheKey = "/api/tutorials";  
 // \_cache.Remove(cacheKey);  
  
 return Ok(createdTutorial);  
 }  
 // [HttpPut("{id}")]  
 // public async Task<IActionResult> PutTutorial(string id, [FromBody] TutorialUpdateRequest model)  
 // {  
 // await \_tutorialService.UpdateTutorial(id, model);  
 //  
 // return NoContent();  
 // }  
  
 [HttpDelete("{id}")]  
 public async Task<IActionResult> DeleteTutorial(string id)  
 {  
 await \_tutorialService.DeleteTutorial(id);  
  
 return NoContent();  
 }  
  
 [HttpPost("{tutorialId}/comments")]  
 public async Task<IActionResult> AddCommentToTutorial(string tutorialId, [FromBody] CommentCreateRequest comment)  
 {  
 await \_tutorialService.AddCommentToTutorial(tutorialId, comment);  
 return NoContent();  
 }  
  
 [HttpPost("{tutorialId}/likes")]  
 public async Task<IActionResult> AddLikeToTutorial(string tutorialId, [FromBody] LikeDTO like)  
 {  
 try  
 {  
 await \_tutorialService.AddLikeToTutorial(tutorialId, like);  
 // Return a JSON response with a success message  
 return Ok(new { message = "Like added successfully" });  
 }  
 catch (Exception ex)  
 {  
 Console.Error.WriteLine($"Error adding like: {ex.Message}");  
 // Return a JSON response with an error message  
 return StatusCode(500, new { error = "Internal Server Error" });  
 }  
 }  
  
  
 [HttpDelete("{tutorialId}/comments/{commentId}")]  
 public async Task<IActionResult> RemoveCommentFromTutorial(string tutorialId, string commentId)  
 {  
 await \_tutorialService.RemoveCommentFromTutorial(tutorialId, commentId);  
 return NoContent();  
 }  
  
 [HttpDelete("{tutorialId}/likes/{likeId}")]  
 public async Task<IActionResult> RemoveLikeFromTutorial(string tutorialId, string likeId, string userId)  
 {  
 await \_tutorialService.RemoveLikeFromTutorial(tutorialId, likeId, userId);  
 return NoContent();  
 }  
   
 [HttpGet("search")]  
 public async Task<ActionResult<List<TutorialDTO>>> SearchTutorials(string searchTerm)  
 {  
 var tutorials = await \_tutorialService.SearchTutorials(searchTerm);  
 return Ok(tutorials);  
 }  
  
 }  
}

**OrdersController code:**using System;  
using Microsoft.AspNetCore.Mvc;  
using System.Collections.Generic;  
using System.Threading.Tasks;  
using Data.Entities.Order;  
using Service.IServices;  
  
namespace WebAPI.Controllers  
{  
 [Route("api/**[controller]**")]  
 [ApiController]  
 public class OrdersController : ControllerBase  
 {  
 private readonly IOrderService \_orderService;  
  
 public OrdersController(IOrderService orderService)  
 {  
 \_orderService = orderService;  
 }  
  
 [HttpGet("all/{userId}")]  
 public async Task<ActionResult<List<OrderDto>>> GetOrders(string userId)  
 {  
 var orders = await \_orderService.GetOrders(userId);  
 return Ok(orders);  
 }  
   
   
 [HttpGet("{id}")]  
 public async Task<ActionResult<OrderDto>> GetOrder(string id)  
 {  
 var order = await \_orderService.GetOrderById(id);  
 if (order == null)  
 {  
 return NotFound();  
 }  
 return Ok(order);  
 }  
  
  
 [HttpPost("{userId}")]  
 public async Task<ActionResult<OrderDto>> PostOrder(OrderRequest orderRequest, string userId)  
 {  
 try  
 {  
 if (!ModelState.IsValid)  
 {  
 return BadRequest(ModelState);  
 }  
  
 var createdOrder = await \_orderService.CreateOrder(orderRequest, userId);  
 return CreatedAtAction(nameof(GetOrder), new { id = createdOrder.Id }, createdOrder);  
 }  
 catch (Exception ex)  
 {  
 return StatusCode(500, $"Internal server error: {ex.Message}");  
 }  
 }  
  
 [HttpPut("{id}")]  
 public async Task<IActionResult> PutOrder(string id, OrderDto orderDto)  
 {  
 if (id != orderDto.Id)  
 {  
 return BadRequest();  
 }  
  
 await \_orderService.UpdateOrder(id, orderDto);  
  
 return NoContent();  
 }  
  
 [HttpDelete("{id}")]  
 public async Task<IActionResult> DeleteOrder(string id)  
 {  
 await \_orderService.DeleteOrder(id);  
  
 return NoContent();  
 }  
   
 }  
}

**PaypalsController code:**// PayPalController.cs  
  
using System;  
using System.Threading.Tasks;  
using Data.Context;  
using Data.Entities.User;  
using Microsoft.AspNetCore.Mvc;  
using Microsoft.EntityFrameworkCore;  
using Service.IServices;  
using Service.Service;  
  
namespace WebAPI.Controllers  
{  
 [ApiController]  
 [Route("api/**[controller]**")]  
 public class PayPalController : ControllerBase  
 {  
 private readonly IPayPalService \_payPalService;  
 private readonly AppDbContext \_db;  
 private readonly UserService \_userService;  
  
 public PayPalController(IPayPalService payPalService, AppDbContext db)  
 {  
 \_payPalService = payPalService;  
 \_db = db;  
 }  
  
 [HttpPost("create-order")]  
 public async Task<IActionResult> CreateOrder(decimal amount)  
 {  
 try  
 {  
 var orderId = await \_payPalService.CreateOrder(amount);  
 return Ok(new { OrderId = orderId });  
 }  
 catch (Exception ex)  
 {  
 return BadRequest(new { Message = ex.Message });  
 }  
 }  
  
 [HttpPost("capture-payment")]  
 public async Task<IActionResult> CapturePayment(string orderId)  
 {  
 try  
 {  
 var success = await \_payPalService.CaptureOrder(orderId);  
 if (success)  
 {  
 return Ok(new { Message = "Payment captured successfully." });  
 }  
 else  
 {  
 return BadRequest(new { Message = "Failed to capture payment." });  
 }  
 }  
 catch (Exception ex)  
 {  
 return BadRequest(new { Message = ex.Message });  
 }  
 }  
   
 [HttpPost("connect-paypal")]  
 public async Task<IActionResult> ConnectPayPal([FromBody] PayPalConnectRequest request)  
 {  
 // Validate request  
 if (!ModelState.IsValid)  
 {  
 return BadRequest(ModelState);  
 }  
  
 // Save PayPal information to the database  
 var user = await \_db.Users.FindAsync(request.UserId);  
 if (user == null)  
 {  
 return NotFound();  
 }  
  
 user.PayPalEmail = request.PayPalEmail;  
 user.PayPalFirstName = request.PayPalFirstName;  
 user.PayPalLastName = request.PayPalLastName;  
 user.IsPayPalLinked = true;  
  
 await \_db.SaveChangesAsync();  
  
 return Ok(new { user.PayPalEmail, user.PayPalFirstName, user.PayPalLastName });  
 }  
   
 [HttpPost("send-payment")]  
 public async Task<IActionResult> SendPayment(string recipientEmail, decimal amount)  
 {  
 try  
 {  
 var success = await \_payPalService.SendPayment(recipientEmail, amount);  
 if (success)  
 {  
 return Ok(new { Message = "Payment sent successfully." });  
 }  
 else  
 {  
 return BadRequest(new { Message = "Failed to send payment." });  
 }  
 }  
 catch (Exception ex)  
 {  
 return BadRequest(new { Message = ex.Message });  
 }  
 }  
 }  
}

**11.8. GitHub**



**11.9. Project Management**

**11.10. Deployment**

**11.11. Images**

10. TESTING

11. EVALUATION

**Summarized key findings from the project**

**Recommendations for the future development**

**Project evaluation**

**Personal evaluation**

**Conclusion**

12. REFERENCES

13. APPENDIX A – PROJECT PROPOSAL

14. APPENDIX B – PLANNING

15. APPENDIX C