Final Report | 2023

Multifunctional Task Management System with RESTful API

By

Heba Aljedayeh

Razan Hayajneh

supervised by

Dr.AHMAD ALSALEIM

Table Of Content

List of Figure

Introduction

Welcome to our cutting-edge application project that utilizes Flutter and incorporates various design patterns such as BLoC, MVC, Builder, and Repository. In this project, we aim to develop a robust and scalable mobile application that follows best practices and utilizes these design patterns to enhance code organization, maintainability, and reusability.

Flutter, a powerful cross-platform development framework, forms the foundation of our application. It allows us to write code once and deploy it on both iOS and Android platforms, saving time and effort in the development process.

Now, let's delve into the design patterns we have implemented:

BLoC (Business Logic Component): We have adopted the BLoC pattern to separate business logic from the UI layer. The BLoC pattern helps us manage and maintain the application's state and data flow efficiently. It promotes a reactive programming style, where events trigger state changes, which in turn update the UI. By utilizing streams and sinks, we can handle asynchronous operations and keep the UI in sync with the underlying data.

MVC (Model-View-Controller): Our application follows the MVC pattern to ensure a clear separation of concerns. The model represents the data and business logic, the view handles the UI components and user interactions, and the controller acts as the intermediary, connecting the model and view. This separation allows for easier code maintenance, testing, and extensibility.

Builder Pattern: The Builder pattern is utilized to create complex objects step by step. By breaking down the construction process into multiple steps, the Builder pattern allows us to construct objects with different configurations or parameters, providing flexibility and reducing complexity. This pattern is particularly useful when dealing with complex UI components or constructing objects with multiple dependencies.

Repository Pattern: The Repository pattern provides an abstraction layer between the data sources (such as APIs or databases) and the rest of the application. It centralizes data access and management, allowing us to handle data retrieval, storage, and caching in a consistent manner. By using the Repository pattern, we can easily switch between different data sources or add caching mechanisms without impacting the rest of the application.

By incorporating these design patterns into our application, we aim to achieve several benefits, including:

Improved code organization and maintainability: The design patterns help separate concerns, making the codebase easier to understand, modify, and extend. Each pattern provides a clear structure and promotes code reusability.

Enhanced testability: The design patterns facilitate unit testing as they promote loose coupling and separation of concerns. This allows for easier isolation of components during testing and improves overall test coverage.

Scalability and extensibility: The design patterns ensure that the application is built in a modular and flexible manner. This makes it easier to add new features, integrate with additional APIs, or scale the application as user requirements evolve.

In conclusion, our project encompasses the use of Flutter, along with the adoption of various design patterns such as BLoC, MVC, Builder, and Repository. These patterns enhance the structure, maintainability, and scalability of the application, ultimately providing users with a high-quality and efficient user experience.

Join us as we embark on this exciting journey of building a sophisticated and well-architected application that leverages the power of Flutter and the benefits of these design patterns. Together, we can create an exceptional mobile application that meets the needs and expectations of our users.

.

Chapter 1

Components and Technologies

1.1 Introduction

components and technologies used in our project:

1. Flutter: Flutter is the cross-platform development framework used for building the team member application. It allows us to write code once and deploy it on both iOS and Android platforms, saving development time and effort.
2. GoDaddy (CPanel Hosting): GoDaddy is the hosting provider that we have chosen to host our application. We utilize the CPanel hosting service provided by GoDaddy to manage and deploy our application.
3. Database: MyPHPAdmin: MyPHPAdmin is the tool we use for managing and interacting with our database. It provides a web-based interface that allows us to perform various database operations such as creating tables, executing queries, and managing database settings.
4. Backend: Laravel framework (PHP): We have implemented the backend of our application using the Laravel framework, which is a popular PHP framework known for its simplicity, flexibility, and extensive set of features. Laravel provides a solid foundation for developing robust and scalable web applications. Frontend: For the admin application, we have utilized Node.js as the runtime environment. Node.js allows us to build scalable and high-performance web applications using JavaScript on the server-side. For the team member application, we have chosen Flutter as the frontend framework. Flutter provides a rich set of pre-built UI components, a reactive programming model, and excellent performance, enabling us to create a seamless and intuitive user interface for the team members.

1.2 Conclusion

With this technology stack, we have created a comprehensive task management application. The Flutter-based team member application provides a user-friendly interface for managing tasks, while the admin application, built with Node.js, offers powerful features for managing projects, assigning tasks, and monitoring team progress. The Laravel backend handles data storage, retrieval, and business logic, ensuring a robust and reliable foundation for our application.

To explore our project, you can visit our website at http://taskmanagement.schooldemos.com/. The website provides access to the application, allowing you to experience the functionality and features firsthand.

Please note that the information provided above is based on the given context and technology stack. Adjustments may be necessary depending on the specific requirements and implementation details of your project.

Chapter 2

Demo Application Two side

2.1 introduction

Our demo application is a comprehensive task management system that consists of two sides: an admin application and a team member application. This application is designed to streamline task management processes and facilitate effective collaboration within teams.

On the admin side, we have developed a Node.js-based web application that serves as the central control panel for project management. It provides administrators with a user-friendly interface to create and manage projects, assign tasks to team members, set deadlines, track progress, and generate reports. The admin application utilizes the powerful capabilities of Node.js to handle server-side operations efficiently and deliver a seamless user experience.

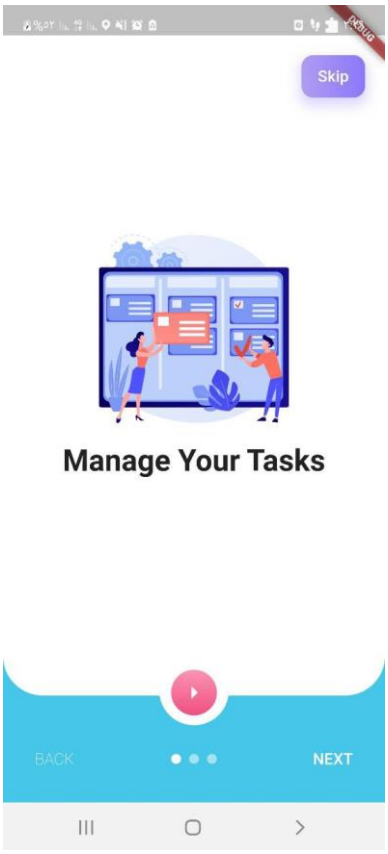


Figure 2.1: splash screen

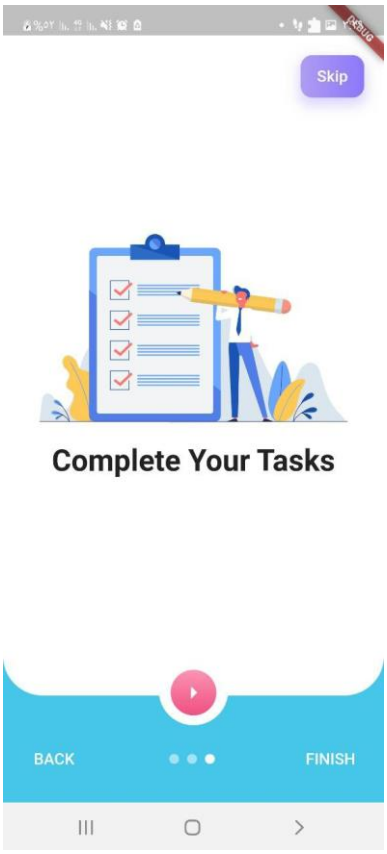


Figure 2.2: splash screen

The Welcome Screen in our Task Management application serves as the initial point of interaction between the user and the application. It plays a crucial role in setting the tone and providing a positive first impression to users.

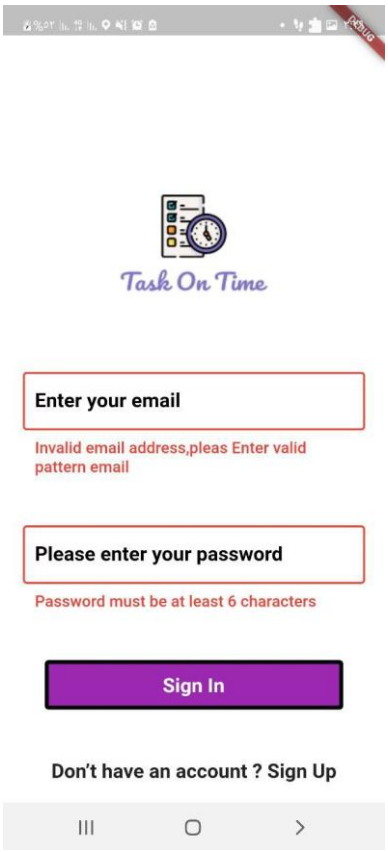


Figure 2.3: sign up screen

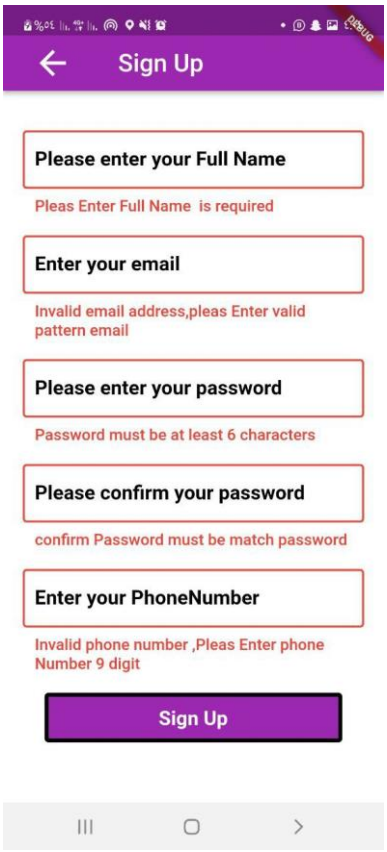


Figure 2.4: sign in screen

The Login and Registration screens in our Task Management application are essential components that enable users to access and manage their accounts. These screens facilitate the interaction between users and the application's backend through RESTful API requests.

When a user wants to create a new account, they can navigate to the Registration screen. This screen allows them to enter their personal information, including full name, phone number, email address, password, and confirmed password. Upon submitting the registration form, a POST request is made to the API endpoint "/api/v1/register" with the user's data in the request body. The API validates the data and creates a new account.

By utilizing the BLoC design pattern, we can separate the business logic from the UI, making the code more manageable and testable. The BLoC classes will handle the API integration and state management, ensuring a smooth and reliable login and registration process for the users of our Task Management application. initial data to the target device before executing computations.

In the Home Page of our Task Management application, we have implemented a feature to fetch and display a list of projects. This is done using the BLoC pattern to handle the API requests and manage the state of the project data.

1. Project BLoC: - We have a dedicated Project BLoC class that handles the business logic

Figure 2.5: Home Screen

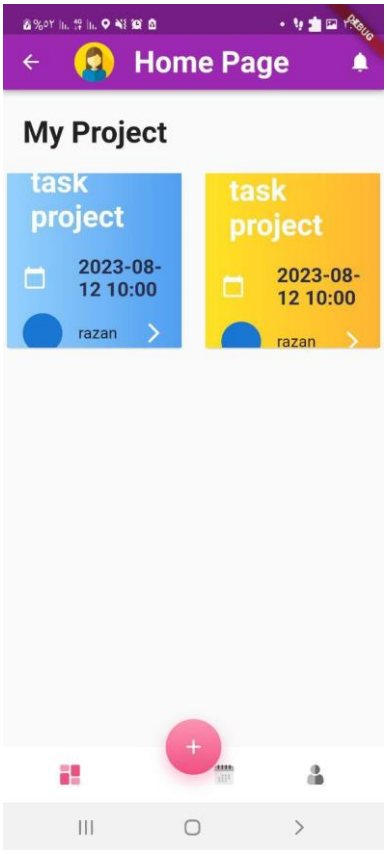


Figure 2.6: splash screen

related to fetching and managing the project data. - The Project BLoC communicates with the API to fetch the project information using the appropriate API endpoint. - It handles the API response, processes the data, and updates the state accordingly.

2. Home Page UI: - In the Home Page UI, we have a section to display the list of projects. - We observe the state changes emitted by the Project BLoC using a StreamBuilder or a similar widget. - When the BLoC emits a loading state, we show a loading indicator to indicate that the project data is being fetched. - Once the data is fetched successfully, we display the project information, including icons or any other relevant details.

3. Requesting Project Data: - When the Home Page is loaded or when a specific event occurs, we trigger the Project BLoC to fetch the project data. - The BLoC makes the API request and handles the response asynchronously. - If the request is successful, the BLoC updates its state with the received project data. - If there is an error during the API request, the BLoC emits an error state, which can be handled in the UI to display an error message.

By using the BLoC pattern for fetching and managing project data, we ensure separation of concerns and maintain a clean and organized code structure. The BLoC takes care of the API integration and state management, allowing the Home Page to focus on displaying the fetched project data in an intuitive and user-friendly manner, with the necessary icons and relevant information.

2.2 Demo Website

The login screen of the Admin website ensures secure access to the system and allows authorized administrators to log in using their credentials. It serves as the entry point for administrators to access and manage the system's data and settings, including the ability to seed initial data into the database.

The Admin website of our Task Management application provides functionality for managing administrators, including adding new admins, viewing all admins in the database, updating admin information, and deleting admins. Here's a description of the features along with the technical aspects related to the API:

Adding a New Admin:

The Admin website allows the logged-in admin to add a new admin to the system. The admin fills in the required information of the new admin, such as full name, email, and password. The Admin website sends a POST request to the API endpoint /api/v1/admins with the new admin's data in the request body. The API processes the request, validates the data, and adds the new admin to the database. Upon successful addition, the API returns a response indicating the success status and any relevant data.

SIGN IN Page:

- The first admin was seeded with Database.
- The Admin can just sign in by website

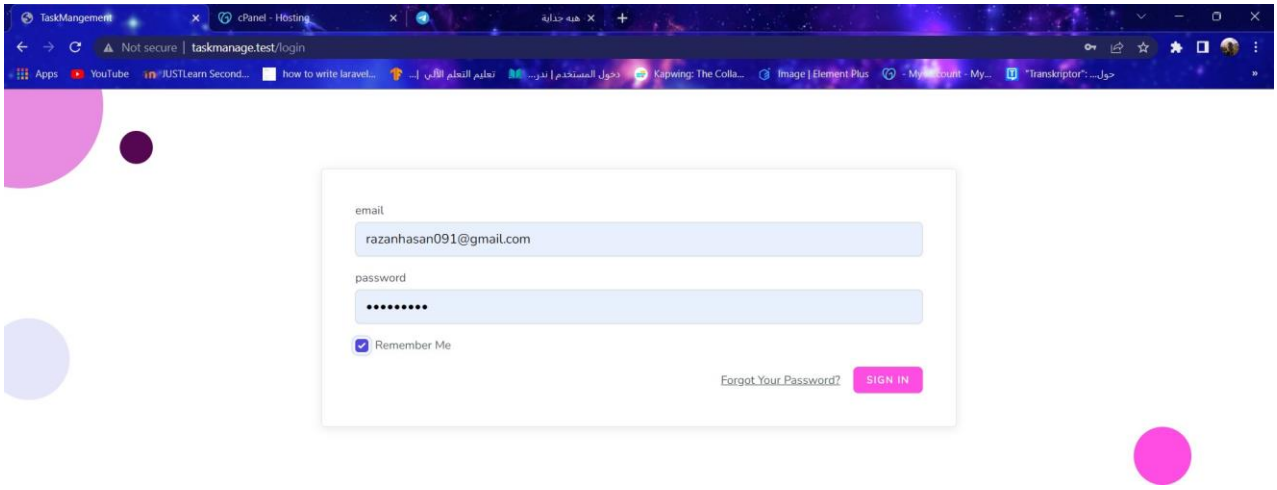


Figure 2.7: Sign in

Admins Page:

- Admin can add new admin, view all admins in database, update on admin information and delete any admin
- That is according their access level

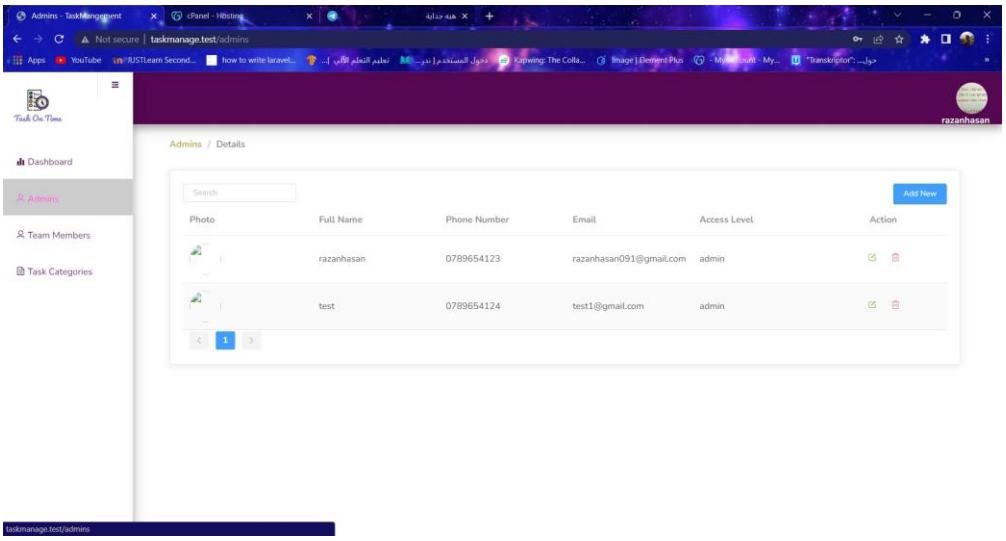


Figure 2.8: Home Page for Admin

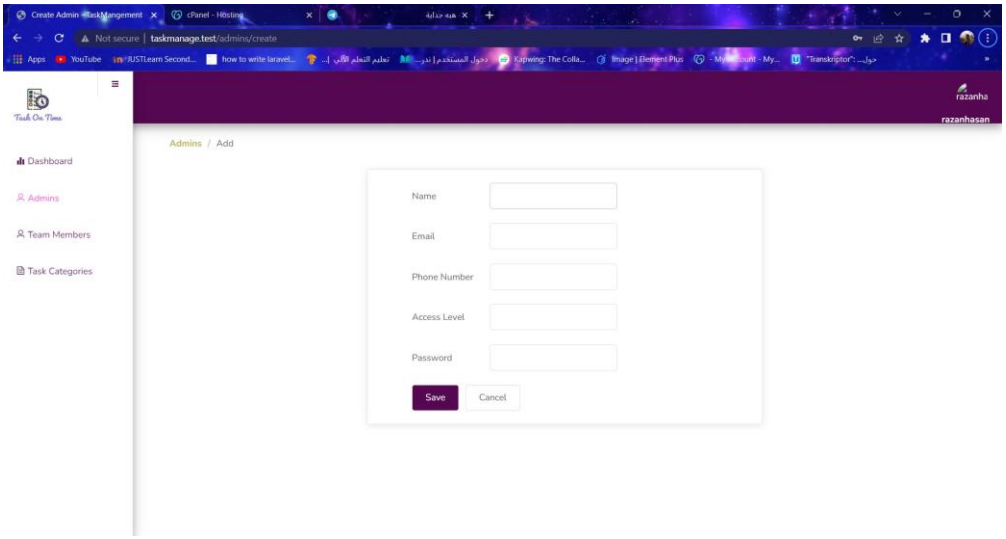


Figure 2.9: Add Page for Admin

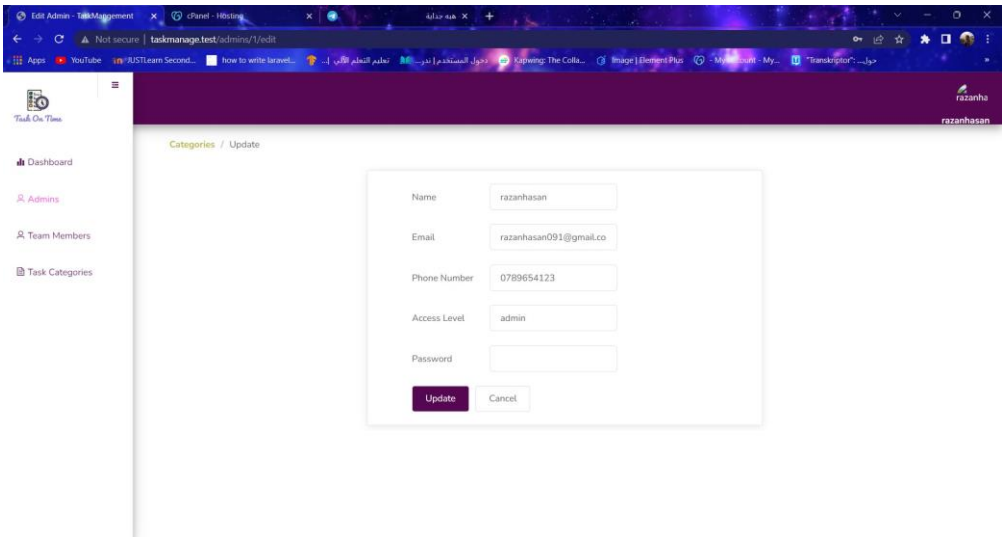


Figure 2.10: Update Page for Admin

Team Members Page:
• Here is the view of all team members that registered by team member application,
• Admin can update or delete any team member

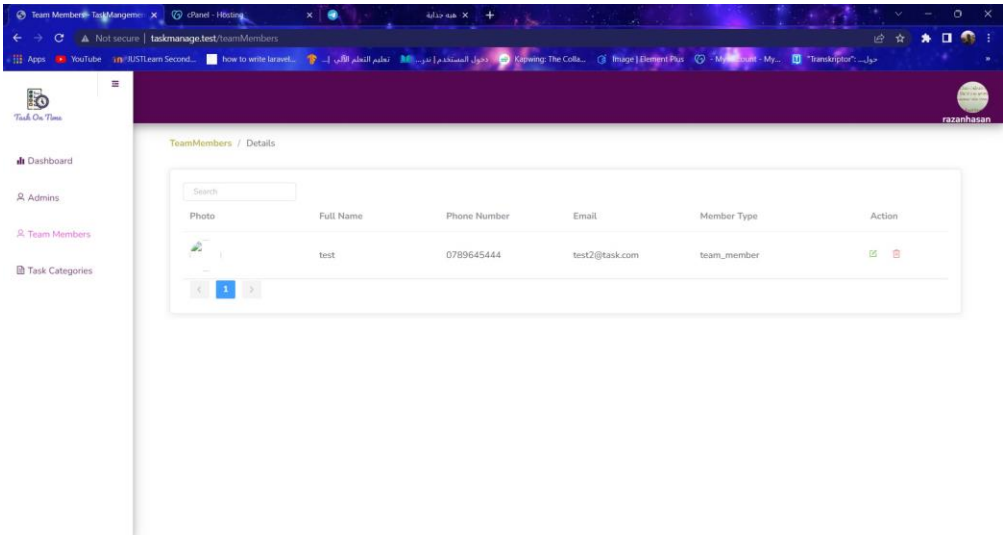


Figure 2.11: view of all team members

Task Categories Page:

- Admin can view all categories that can use to determine category type for task when task creates
- Admin can update and delete specific category

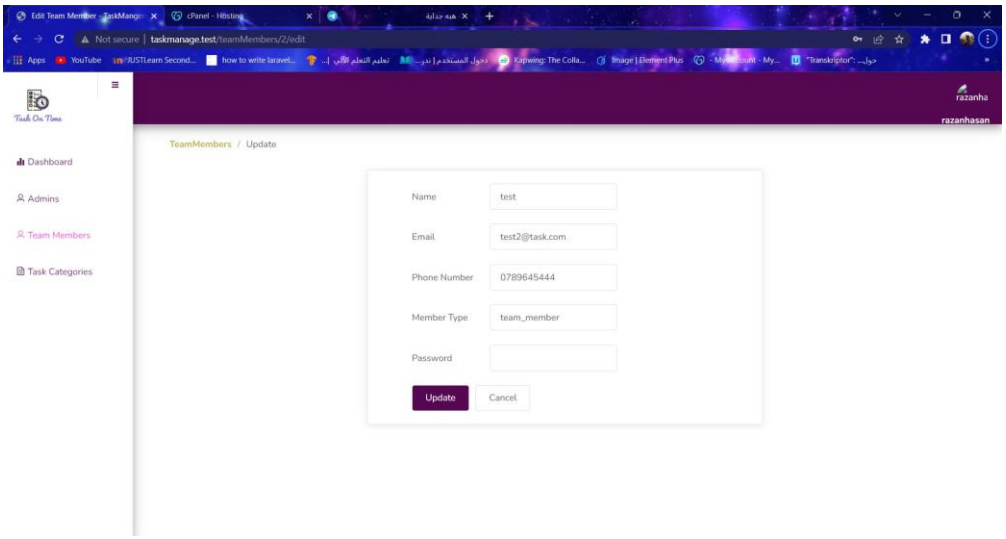


Figure 2.12: all categories

Profile Management Page:

- here can update on own information and reset admin password

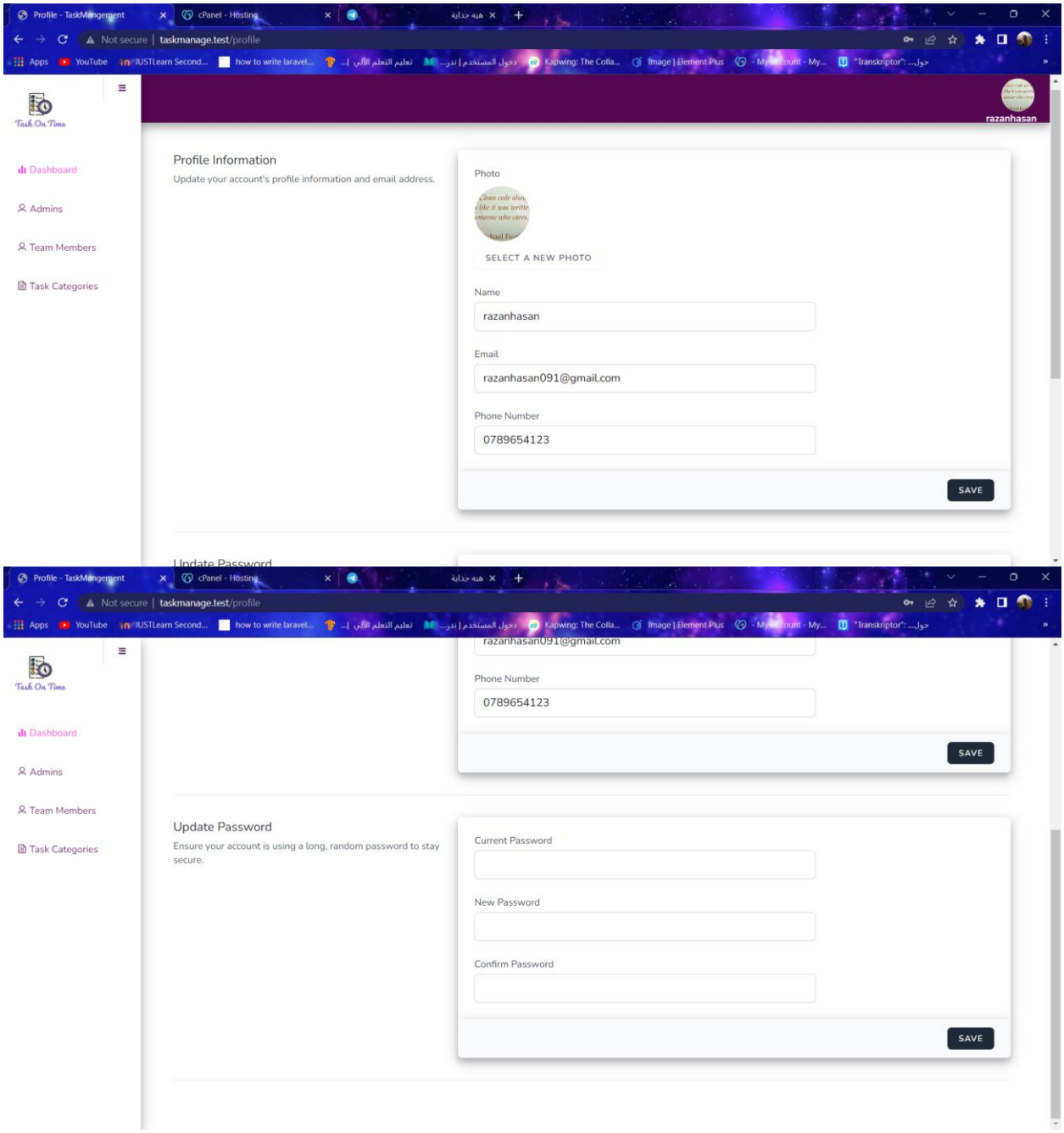
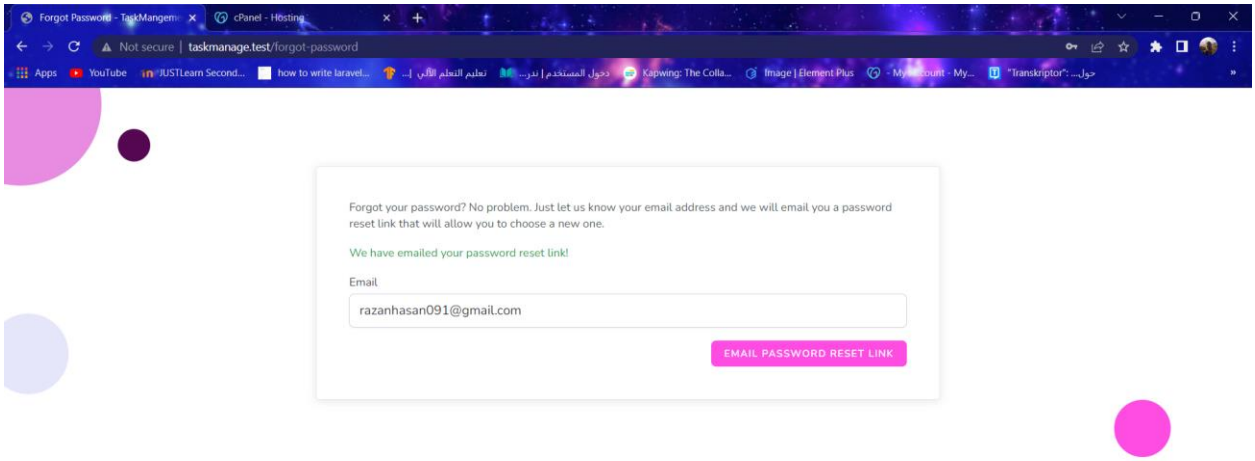


Figure 2.13:Profile

Forget Password:
You can reset your password if you forgot it by send message to your email



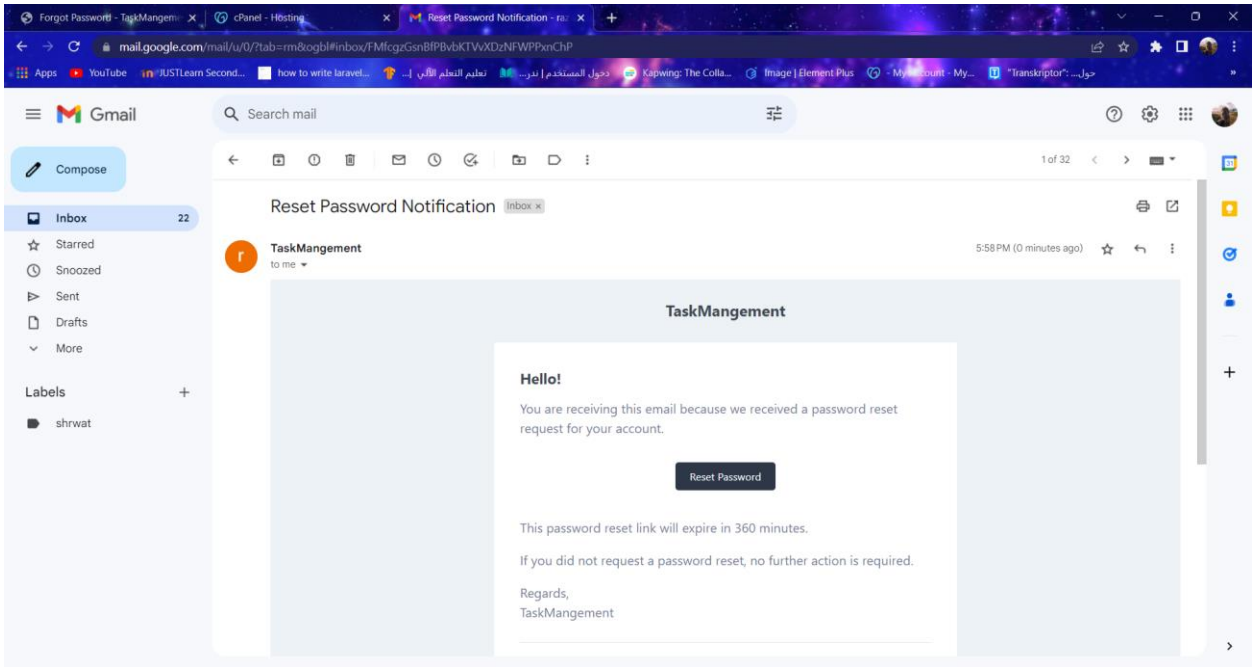


Figure 2.14: email

Reset Password by Email Page:
By clicking on reset password, it will open reset password form as the following picture

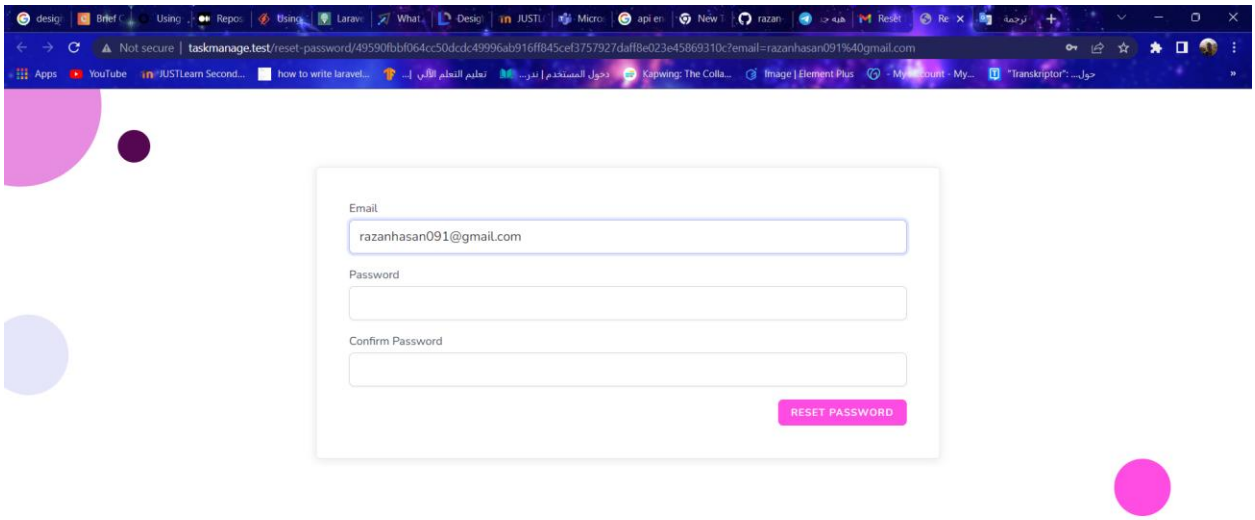


Figure 2.15: Reset password

The Admin website of our Task Management application allows administrators to create new admins and update existing admin information. Here's a description of these features along with the technical aspects related to the API:

Create New Admin:

The Admin website provides a form where administrators can enter the required information for creating a new admin, such as full name, email, and password. Upon submitting the form, the Admin website sends a POST request to the API endpoint /api/v1/admins with the new admin's data in the request body. The API receives the request, validates the data, and creates a new admin in the database with the provided information. The API returns a response indicating the success status of the operation along with any relevant data, such as the newly created admin's ID or a

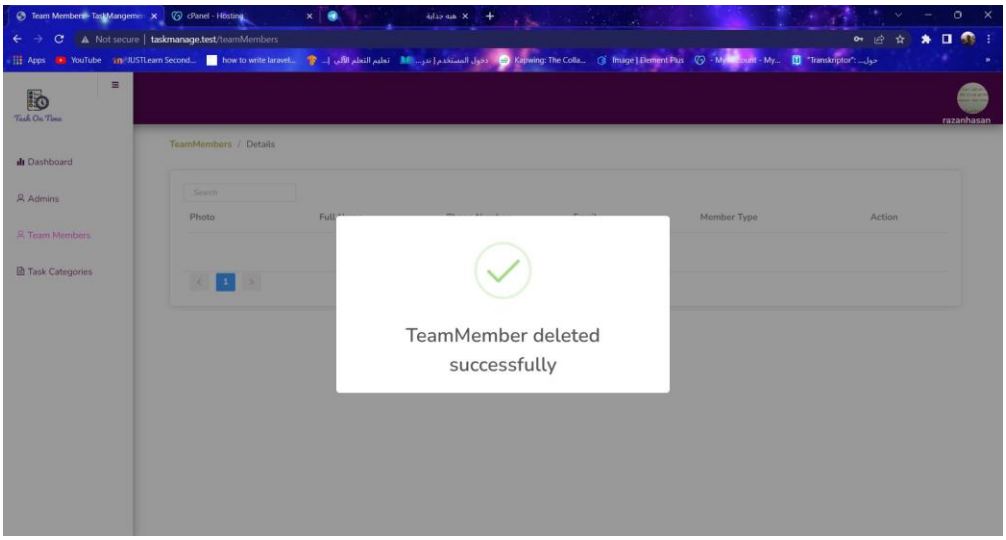


Figure 2.13: update and delete specific category

success message. Update Admin:

The Admin website allows administrators to update the information of an existing admin. Administrators can navigate to the admin details page or a dedicated edit page to modify the admin's information. The Admin website sends a PUT request to the API endpoint /api/v1/admins/adminId with the updated data in the request body. The API receives the request, validates the data, and updates the corresponding admin's information in the database. Upon successful update, the API returns a response indicating the success status, possibly along with the updated admin's details or a success message. Technical Aspects:

The API endpoints /api/v1/admins (POST) and /api/v1/admins/adminId (PUT) are responsible for creating and updating admins, respectively. The API implements appropriate validation rules to ensure the correctness and integrity of the admin data. User authentication and authorization mechanisms are in place to restrict access to these API endpoints to authorized administrators. The API leverages secure communication protocols (e.g., HTTPS) to transmit admin data securely between the Admin website and the API server. Database operations, such as creating a new admin or updating admin information, are performed using suitable database query methods or ORM techniques. Error handling is implemented in the API to handle scenarios where validation fails, database errors occur, or unauthorized access is attempted. Logging and monitoring mechanisms can be implemented in the API to track admin-related actions and capture any errors or abnormal behavior.

Chapter 3

Implementation

3.1 bloc design pattern

3.1.1 introduction

BLoC stands for Business Logic Component. It is a design pattern used in Flutter for managing state and handling data flow in an application. BLoC separates the presentation layer from the business logic layer, making the code more modular and easier to maintain.

In the BLoC pattern, the business logic is encapsulated in a BLoC class, which receives events or actions from the UI and produces states as output. The BLoC class acts as an intermediary between the UI and the data source, such as a database or API. It handles the business logic, performs data transformations, and emits updated states back to the UI.

3.2 why to use

Using the BLoC (Business Logic Component) pattern in our project offers several benefits and advantages:

Separation of Concerns: The BLoC pattern promotes a clear separation between the user interface (UI) and the business logic. This separation allows for better code organization and maintainability.

Reusability: BLoC allows us to encapsulate the business logic into reusable components. This means that the same BLoC can be used across multiple screens or components, reducing code duplication.

Testability: BLoC makes it easier to write unit tests for the business logic independently of the UI. Since the UI and BLoC are decoupled, it becomes simpler to write tests that verify the

correctness of the business logic.

3.3 Restful Api

RESTful API (Representational State Transfer) is an architectural style that defines a set of principles for designing networked applications. It is commonly used in web development to build scalable and efficient web services. RESTful APIs adhere to the following key principles:

3.3.1 what is restful api

RESTful API (Representational State Transfer) is an architectural style that defines a set of principles for designing networked applications. It is commonly used in web development to build scalable and efficient web services. RESTful APIs adhere to the following key principles stateless: Each request from the client must contain all the necessary information for the server to understand and process it. The server does not store any client state between requests.

3.3.2 RESTful API is widely used

RESTful API is widely used and preferred over other approaches for several reasons:

1. Scalability: RESTful APIs are highly scalable as they are stateless and allow for easy horizontal scaling. This means that as the number of clients or requests increases, the system can handle the load by adding more servers.

2. Flexibility: RESTful APIs are flexible and can be used with different client applications, including web, mobile, and desktop applications. They allow for decoupling between the client and server, making it easier to make changes or updates to one without affecting the other.

3. Simplicity: RESTful APIs follow a simple and uniform set of principles, making them easy to understand and use. The use of standard HTTP methods (GET, POST, PUT, DELETE) and status codes adds to their simplicity and ease of implementation.

4. Wide Adoption: RESTful APIs have gained widespread adoption and support in the development community. They are supported by popular frameworks and tools, making it easier for developers to build and consume APIs.

5. Platform Independence: RESTful APIs are platform-independent, which means they can be implemented and consumed by applications built using different programming languages and technologies. This allows for interoperability and integration between diverse systems.

6. Caching: RESTful APIs leverage the HTTP caching mechanisms, allowing clients to cache responses and reduce the load on the server. This improves performance and efficiency.

Overall, RESTful APIs provide a scalable, flexible, and widely supported approach to building web services, making them a popular choice for modern application development.

3.3.3 api for project

API ENDPOINT

Razan Hayajneh

Heba Al-jedayah

Domain for Website

http://taskmanagement.schooldemos.com/

It published by GoDaddy (CPanel Hosting)

WEB APIs

METHOD	URI	NAME	ACTION
GET|HEAD	login	login	Laravel\Fortify\Http\Controllers\AuthenticatedSessionController@create
POST	login	login	Laravel\Fortify\Http\Controllers\AuthenticatedSessionController@store
GET|HEAD	forgot-password	password.request	Laravel\Fortify\Http\Controllers\PasswordResetLinkController@create
POST	forgot-password	password.email	Laravel\Fortify\Http\Controllers\PasswordResetLinkController@store
GET|HEAD	reset-password/{token}	password.reset	Laravel\Fortify\Http\Controllers\NewPasswordController@create
PUT	user/password	user-password.update	Laravel\Fortify\Http\Controllers\PasswordController@update
POST	logout	logout	Laravel\Fortify\Http\Controllers\AuthenticatedSessionController@destroy
GET|HEAD	dashboard	dashboard	App\Http\Controllers\Admin\DashboardController@index
GET|HEAD	profile	profile.show	Closure
GET|HEAD	admins	admins.index	App\Http\Controllers\Admin\AdminController@index
POST	admins	admins.store	App\Http\Controllers\Admin\AdminController@store
GET|HEAD	admins/create	admins.create	App\Http\Controllers\Admin\AdminController@create
GET|HEAD	admins/{ admin }/ edit	admins.edit	App\Http\Controllers\Admin\AdminController@edit
PUT|PATCH	admins/{ admin }	admins.update	App\Http\Controllers\Admin\AdminController@update
DELETE	admins/{ admin }	admins.destroy	App\Http\Controllers\Admin\AdminController@destroy
GET|HEAD	teamMembers	teamMembers.index	App\Http\Controllers\Admin\TeamMemberController@index
GET|HEAD	teamMembers/{teamMember}/ edit	teamMembers.edit	App\Http\Controllers\Admin\TeamMemberController@edit
PUT|PATCH	teamMembers/{ teamMember }	teamMembers.update	App\Http\Controllers\Admin\TeamMemberController@update
DELETE	teamMembers/{ teamMember }	teamMembers.destroy	App\Http\Controllers\Admin\TeamMemberController@destroy
GET|HEAD	categories	categories.index	App\Http\Controllers\Admin\TaskCategoryController@index
POST	categories	categories.store	App\Http\Controllers\Admin\TaskCategoryController@store
GET|HEAD	categories/create	categories.create	App\Http\Controllers\Admin\TaskCategoryController@create
GET|HEAD	categories/{category}/ edit	categories.edit	App\Http\Controllers\Admin\TaskCategoryController@edit
PUT|PATCH	categories/{category}	categories.update	App\Http\Controllers\Admin\TaskCategoryController@update
DELETE	categories/{category}	categories.destroy	App\Http\Controllers\Admin\TaskCategoryController@destroy

APPLICATION APIs

METHOD	URI	ACTION
POST	api/v1/register	App\Http\Controllers\API\AuthAPIController@register
POST	api/v1/login	App\Http\Controllers\API\AuthAPIController@login
POST	api/v1/ logout	App\Http\Controllers\API\AuthAPIController@logout
GET|HEAD	api/v1/get-profile-info	App\Http\Controllers\API\ProfileAPIController@show
POST	api/v1/ update-profile-info	App\Http\Controllers\API\ProfileAPIController@update
POST	api/v1/update-profile-image	App\Http\Controllers\API\ProfileAPIController@updateImage
GET|HEAD	api/v1/projects	App\Http\Controllers\API\ProjectAPIController@index
POST	api/v1/projects	App\Http\Controllers\API\ProjectAPIController@store
GET|HEAD	api/v1/projects/{project}	App\Http\Controllers\API\ProjectAPIController@show
PUT|PATCH	api/v1/projects/{project}	App\Http\Controllers\API\ProjectAPIController@update
DELETE	api/v1/projects	App\Http\Controllers\API\ProjectAPIController@destroy
GET|HEAD	api/v1/project-members	App\Http\Controllers\API\ ProjectMemberAPIController@index
POST	api/v1/project-members	App\Http\Controllers\API\ProjectMemberAPIController@store
DELETE	api/v1/project-members/{project_member}	App\Http\Controllers\API\ProjectMemberAPIController@destroy
GET|HEAD	api/v1/ task-categories	App\Http\Controllers\API\ TaskCategoryAPIController@index
GET|HEAD	api/v1/tasks?project_id={id}	App\Http\Controllers\API\ TaskAPIController@index
POST	api/v1/ tasks?project_id={id}	App\Http\Controllers\API\ TaskAPIController @store
GET|HEAD	api/v1/ tasks/{task}	App\Http\Controllers\API\ TaskAPIController @show
PUT|PATCH	api/v1/ tasks/{ task}	App\Http\Controllers\API\ TaskAPIController @update
DELETE	api/v1/ tasks/{ task}	App\Http\Controllers\API\ TaskAPIController@destroy
GET|HEAD	api/v1/my-tasks	App\Http\Controllers\API\MyTasksAPIController@index
GET|HEAD	api/v1/task-status/{id}	App\Http\Controllers\API\TaskAPIController@updateStatus
GET|HEAD	api/v1/ task-members	App\Http\Controllers\API\ TaskMemberAPIController@index
POST	api/v1/ task-members	App\Http\Controllers\API\ TaskMemberAPIController@store
DELETE	api/v1/ task-members/{ task_member}	App\Http\Controllers\API\ TaskMemberAPIController@destroy
GET|HEAD	api/v1/getCountOfFinishedTaskInProject	App\Http\Controllers\API\TaskCountAPIController@getCountOfFinishedTaskInProject
GET|HEAD	api/v1/getCountOfTaskInProject	App\Http\Controllers\API\TaskCountAPIController@getCountOfTaskInProject
GET|HEAD	api/v1/timelines?task_id={id}	App\Http\Controllers\API\TimelineAPIController@index
GET|HEAD	api/v1/ reports?task_id={id}	App\Http\Controllers\API\ ReportAPIController@index
POST	api/v1/ reports?task_id={id}	App\Http\Controllers\API\ ReportAPIController @store
GET|HEAD	api/v1/ reports/{report}	App\Http\Controllers\API\ ReportAPIController @show
PUT|PATCH	api/v1/ reports/{report}	App\Http\Controllers\API\ ReportAPIController @update
DELETE	api/v1/ reports/{report}	App\Http\Controllers\API\ ReportAPIController@destroy
GET|HEAD	api/v1/notifications	App\Http\Controllers\API\NotificationAPIController@index
POST	api/v1/notifications	App\Http\Controllers\API\NotificationAPIController@store

3.4 Challenge

3.4.1 server side

Find online server support Laravel to upload backend

3.4.2 flutter

1. State Management: Managing the state of the application and ensuring that the UI reflects the latest data can be complex, especially when dealing with asynchronous operations and handling updates from the backend.
2. API Integration: Integrating the Flutter application with the backend API, handling network requests, and parsing the response data require careful implementation and error handling.
3. User Interface Design: Designing a user-friendly and visually appealing user interface that provides a seamless experience for updating and deleting categories can be challenging. It involves creating intuitive UI elements and handling user interactions effectively.
4. Data Validation and Error Handling: Validating user input, handling errors, and providing appropriate error messages to the user when updating or deleting categories require thorough error handling and input validation mechanisms.
5. Testing: Ensuring the reliability and stability of the application by writing comprehensive unit tests for the update and delete functionalities can be time-consuming but is crucial for maintaining a robust codebase.