BMIT3173 Integrative Programming

ASSIGNMENT 202509

Student Name	:	Chia Ming Yi
Student ID	:	24WMR09040
Programme	:	RSD3
Tutorial Group	:	G5

System Title	:	TARUMT Event Management System
Modules	:	User & Membership Modules

Declaration

- I confirm that I have read and complied with all the terms and conditions of Tunku Abdul Rahman University of Management and technology's plagiarism policy.
- I declare that this assignment is free from all forms of plagiarism and for all intents and purposes is my own properly derived work.

Plagiarism Statement Form

I, CHIA MING YI Student ID 24WMR09040 Programme RSD3 Tutorial Group 5 confirm that the submitted work are all my own work and is in my own words.

I CHIA MING YI acknowledge the use of AI generative technology.

Signature: _____

Date: 19/12/2025

Table of Contents

1. Introduction to the System

The **TARUMT Event Management System** is a centralized web-based platform developed to streamline and manage event organization and participation within the TARUMT community. The system is composed of **six core modules**, namely **Users, Membership, Events, Helpers, Payments, and Feedback**.

The **Users module** manages user authentication, role assignment, and access control for administrators, organizers, and members. The **Membership module** supports organizer membership applications, approval workflows, and credit score tracking. The **Events module** allows organizers to create, manage, and monitor events throughout their lifecycle. The **Helpers module** facilitates the creation, application, and assignment management of helper roles for events. The **Payments module** handles all event-related financial transactions, including deposits, penalties, refunds, and receipt generation. Lastly, the **Feedback module** enables participants to submit ratings and comments, supporting event quality assurance and continuous system improvement.

2. Module Description

2.1 Users Module

The Users Module is responsible for user authentication, authorization, profile management, role-based access control, and user administration. The module supports multiple user roles, namely Guest, Member, Organizer, Admin, and Super Admin, each with different access privileges.

2.1.1 Guest

Guests are users who have not logged into the system. They are only allowed to access authentication-related features.

Functions handled:

- Register a new user account:

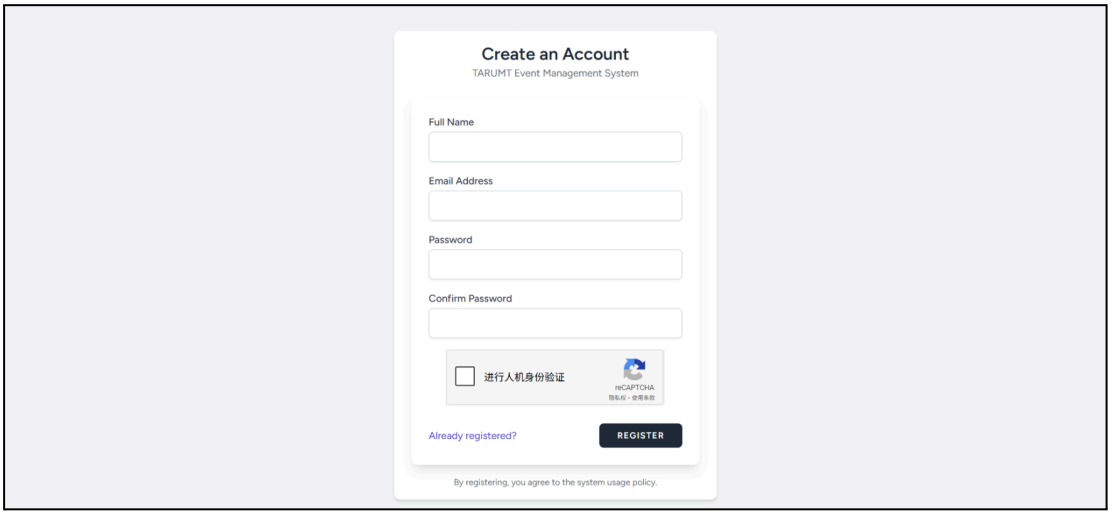


Figure 2.1.1: User Registration Form (Class Path: resources/views/auth/register.blade.php)

This diagram shows the User Registration page for the system.
It allows a guest user to create a new account by entering their full name, email address, password, and password confirmation.

A reCAPTCHA verification is included to prevent automated or malicious registrations.
After completing the form and verification, the user can click the Register button to submit the registration.
Users who already have an account can navigate to the login page using the provided link.

- Email verification after register

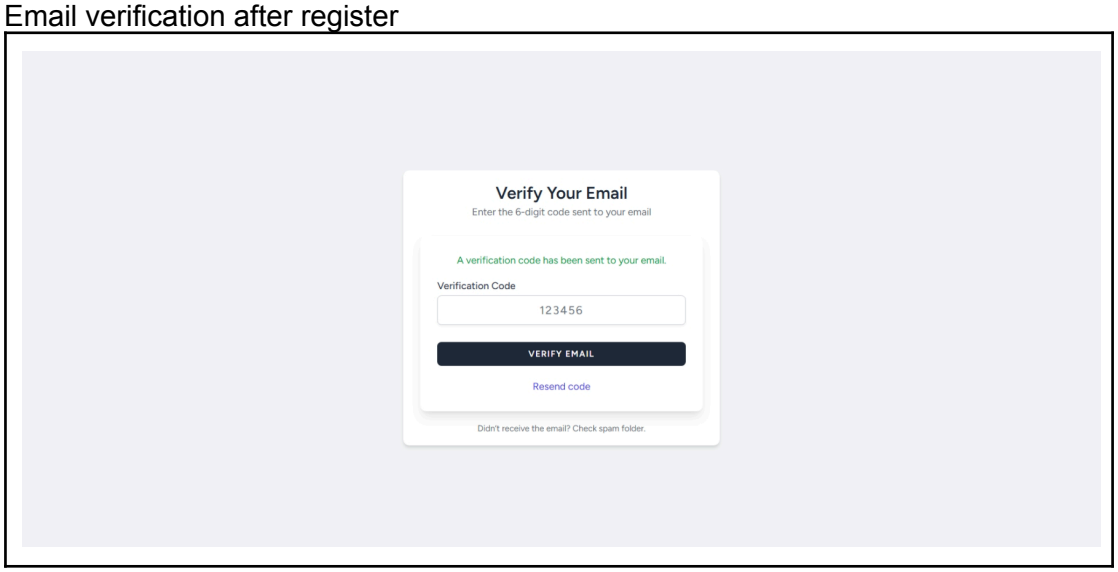


Figure 2.1.2: Email Verification Form After User Registration (Class Path: resources/views/auth/register.blade.php)

This diagram sequence shows the email verification process after user registration.
After registering, the system sends a 6-digit verification code to the user's email address.
The user enters the received code on the Email Verification page to confirm ownership of the email.
Once the verification is successful, the system displays a confirmation message and redirects the user to the Login page.
The user can then log in to the system using their registered email and password.

- Forgot Password: Reset Password via reset link in email

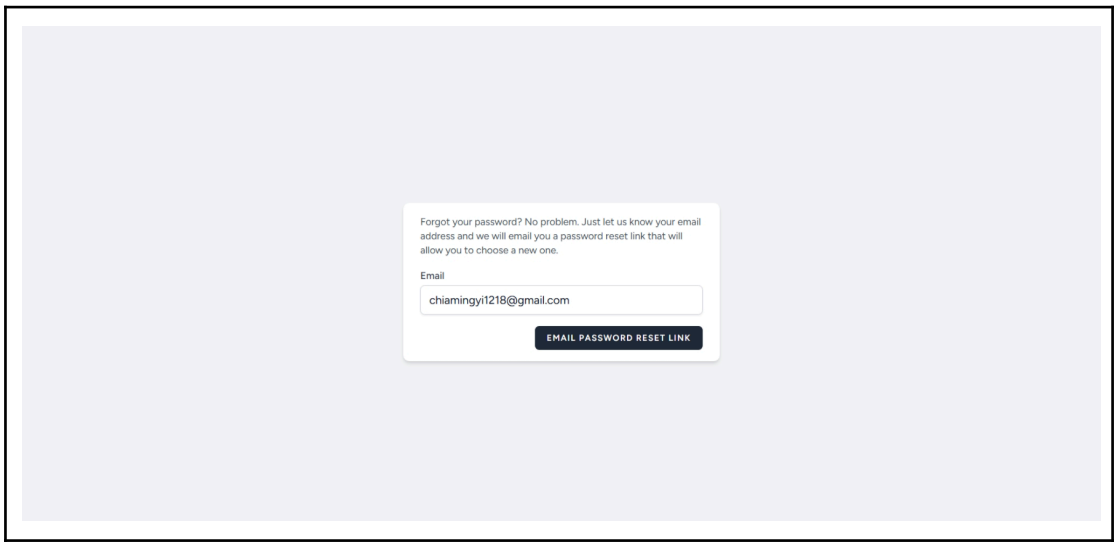


Figure 2.1.3: Send Reset Password Link(Class Path:

resources/views/auth/forgot-password.blade.php)

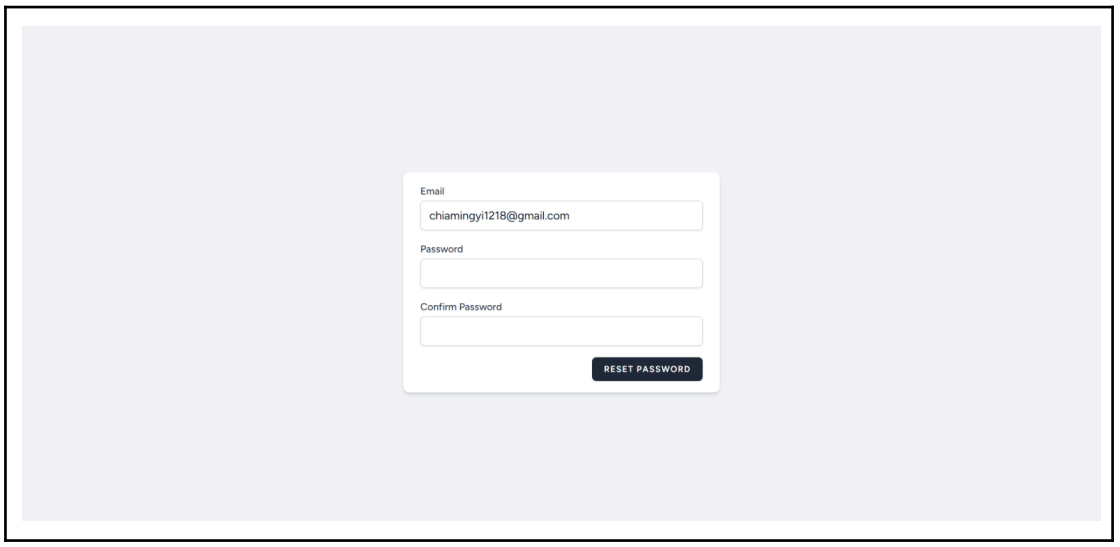


Figure 2.1.4: Reset Password Form (Class Path: resources/views/auth/forgot-password.blade.php)

When a user forgets their password, they enter their registered email address on the Forgot Password page.
The system then sends a password reset link to the user's email address.
By clicking the link in the email, the user is redirected to the Reset Password page, where they can enter and confirm a new password.
Once the password is successfully reset, the user can log in again using the new credentials.

- Login:

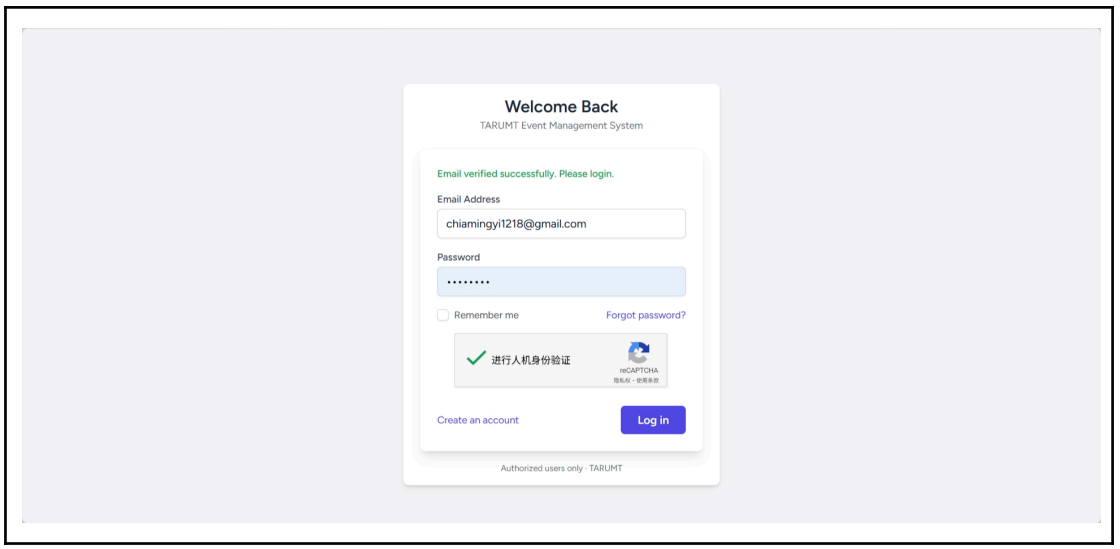


Figure 2.1.5: Login Form (Class Path: resources/views/auth/login.blade.php)

This screen shows the User Registration page for the system.
It allows a guest user to create a new account by entering their full name, email address, password, and password confirmation.
A reCAPTCHA verification is included to prevent automated or malicious registrations.

After completing the form and verification, the user can click the Register button to submit the registration.
Users who already have an account can navigate to the login page using the provided link.

2.1.2 Member

Members are authenticated users who can access their dashboard, manage their profile, and apply to become an organizer.

Functions handled:

- Member Profile Management: edit profile, update password, delete account

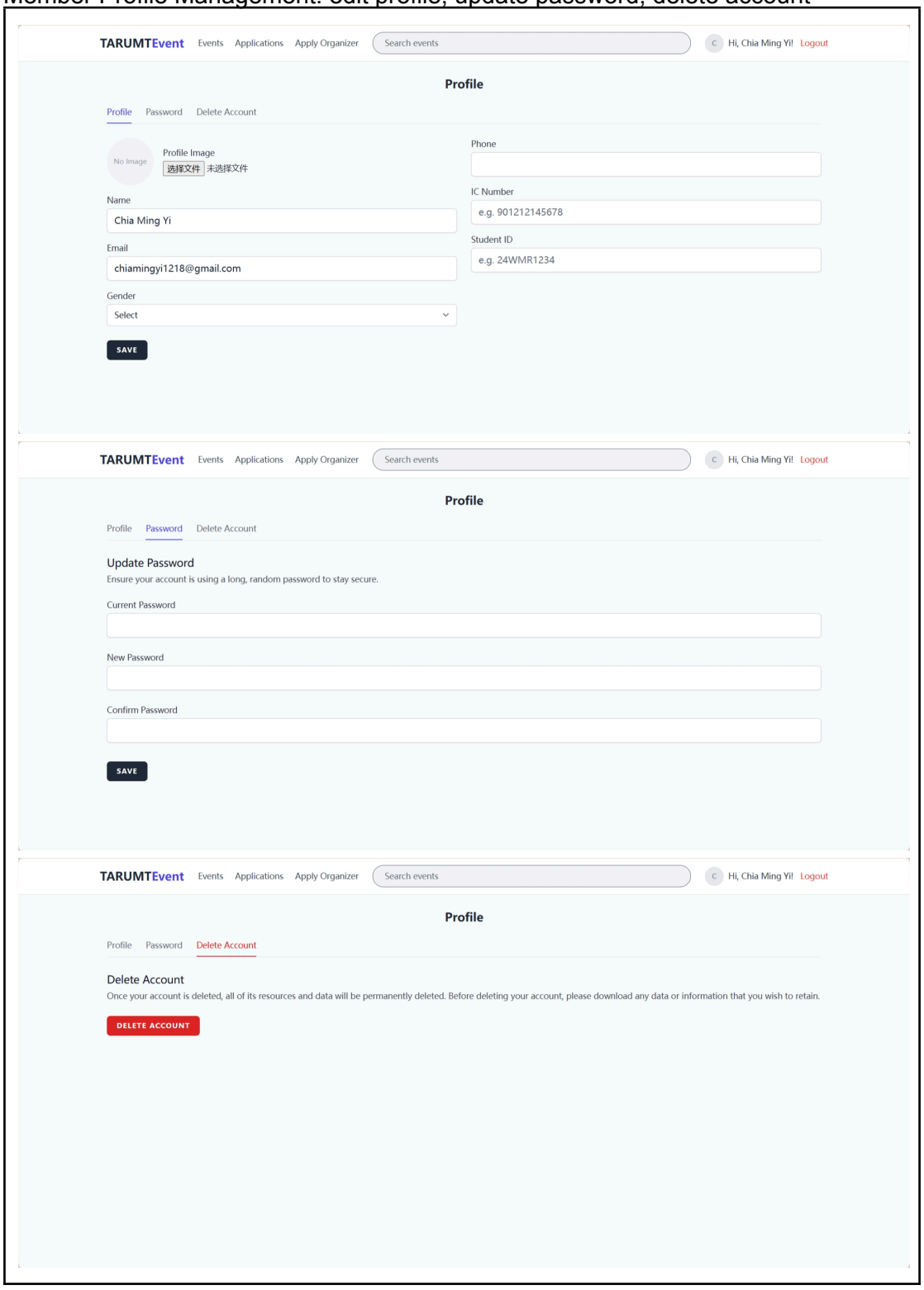
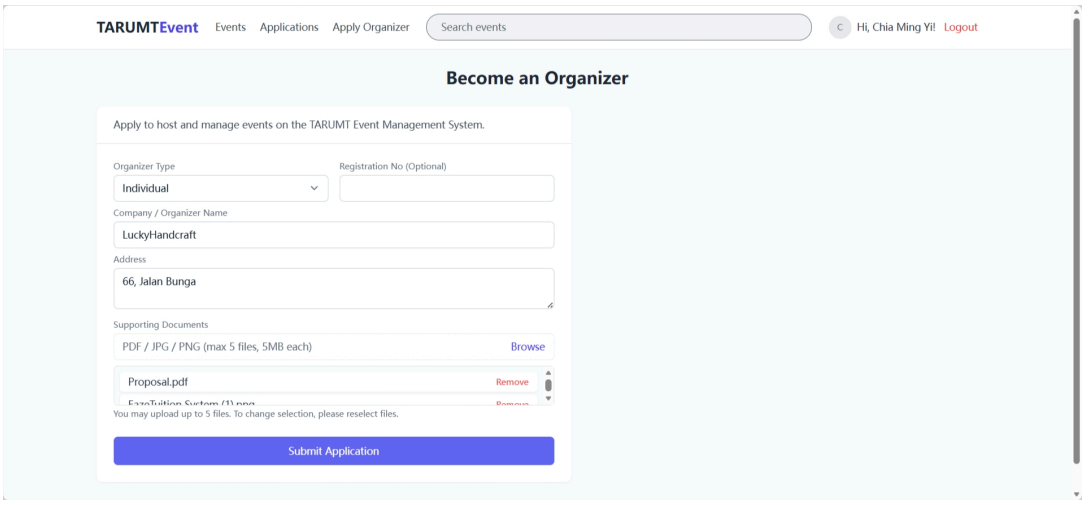
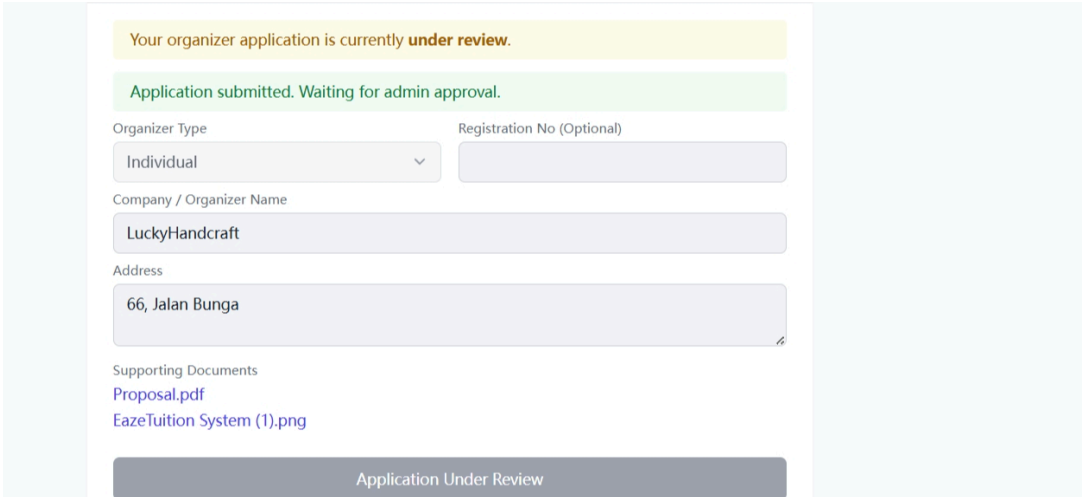


Figure 2.1.6: Member Profile Management (Class Path: resources/views/profile/edit.blade.php)

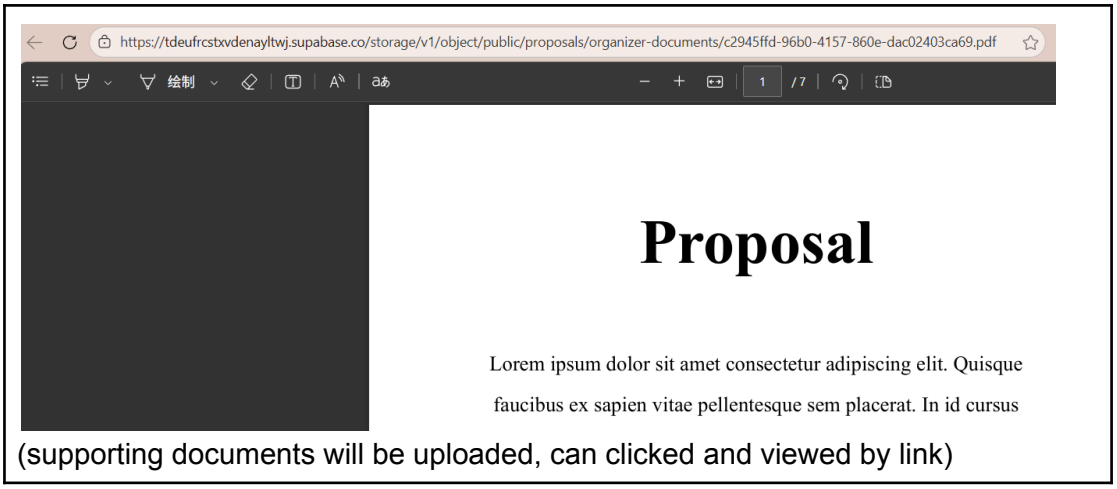
Members can edit their personal profile information, such as name, email, phone number, student ID, gender, and profile image. They can also update their account password by entering the current password and a new password. In addition, members are allowed to delete their account permanently(required password to confirm), which removes all associated data from the system after confirmation.

- Application to become Event Organizer (Need Admin Approval)





(Application submit successfully (Admin later can approve or reject the application))



(supporting documents will be uploaded, can clicked and viewed by link)

Figure 2.1.7: Application to Become Organizer Form (Class Path: resources/views/member/apply-organizer.blade.php)

This feature allows a member to apply to become an organizer in the system.
The member fills in organizer details such as organizer type, company or organizer name, address, and uploads supporting documents (e.g. proposal files).
After submitting the application, the system displays the application status as “Under Review”, indicating that it is pending admin approval.
Uploaded supporting documents can be viewed directly by the admin for verification purposes.

- Resubmit Application to become Event Organizer (If previous application is rejected)

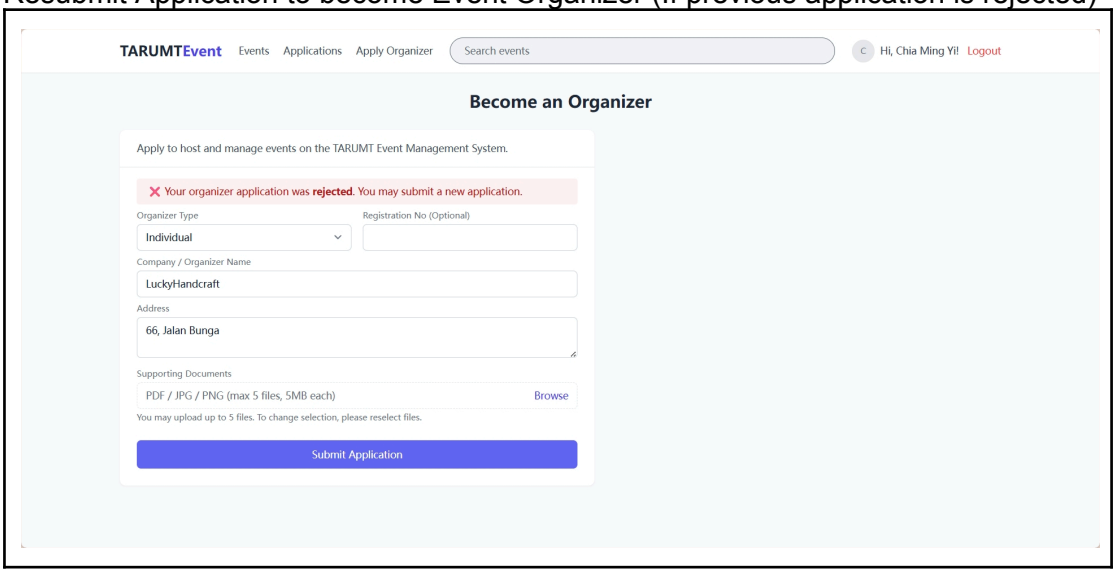


Figure 2.1.8: Resubmit Application to Become Organizer Form (Class Path: resources/views/member/apply-organizer.blade.php)

When a member's organizer application is rejected by the admin, the system displays a rejection message and allows the member to edit the application details and upload new supporting documents.
The member can then resubmit the application for another review by the admin.

2.1.3 Organizer

Organizers are users who have been approved to manage organizational information and memberships.

Functions handled:

- Login as Organizer (after Application is approved by admin)

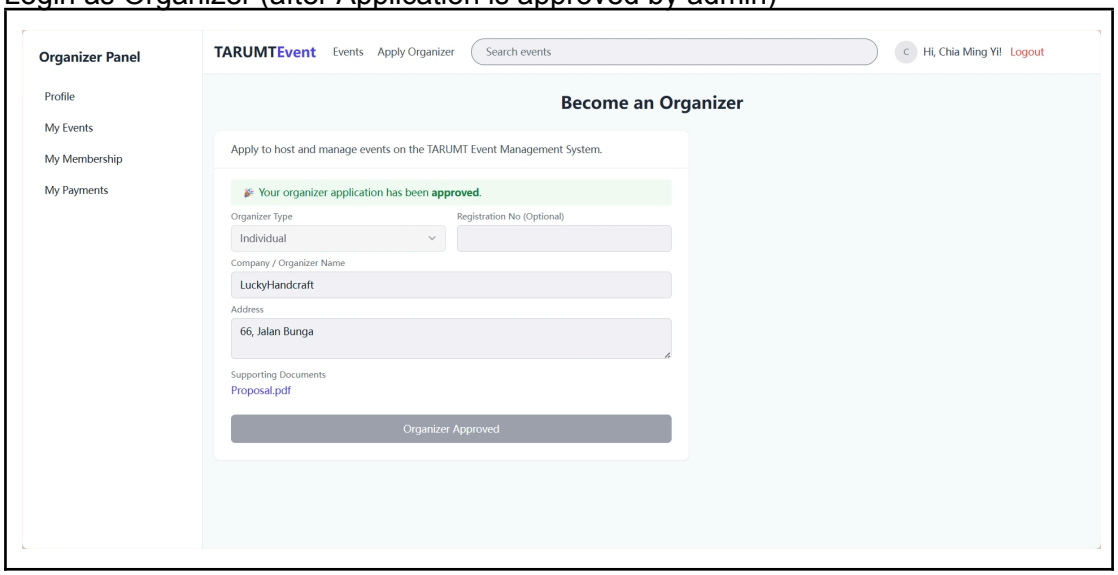


Figure 2.1.9: Approved Member Able to Login as Organizer (Class Path: resources/views/member/apply-organizer.blade.php)

After the admin approves the application, the member is officially granted organizer access.
The system displays an approval message and enables the Organizer Panel, allowing the user to manage events, view membership details, and handle payments.
The application form becomes read-only, indicating that the organizer account has been successfully activated.

- Organizer Profile Management: edit profile, update password, delete account

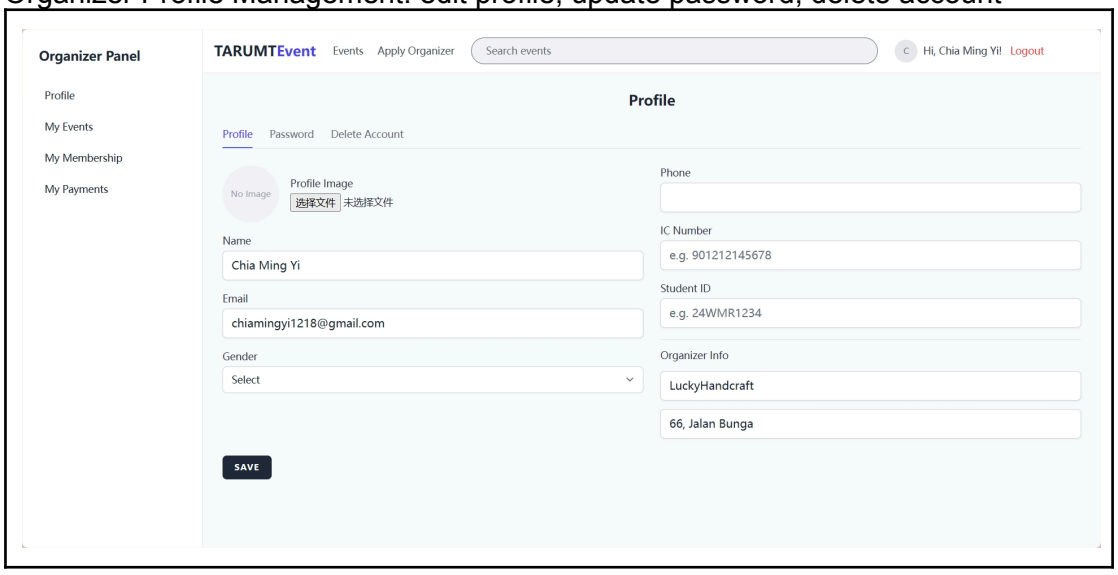


Figure 2.1.10: Organizer Profile Management (Class Path: resources/views/profile/edit.blade.php)

The organizer profile functions similarly to the member profile, allowing users to edit

personal information such as name, email, phone number, gender, student ID, and profile image.
It includes extra organizer-specific fields, such as organizer name and address, which are used for managing events and organizer-related activities.
All updated information can be saved and reflected immediately in the organizer’s account.

2.1.4 Admin

Admins are responsible for managing users and organizers within the system.

Functions handled:

- Admin Dashboard

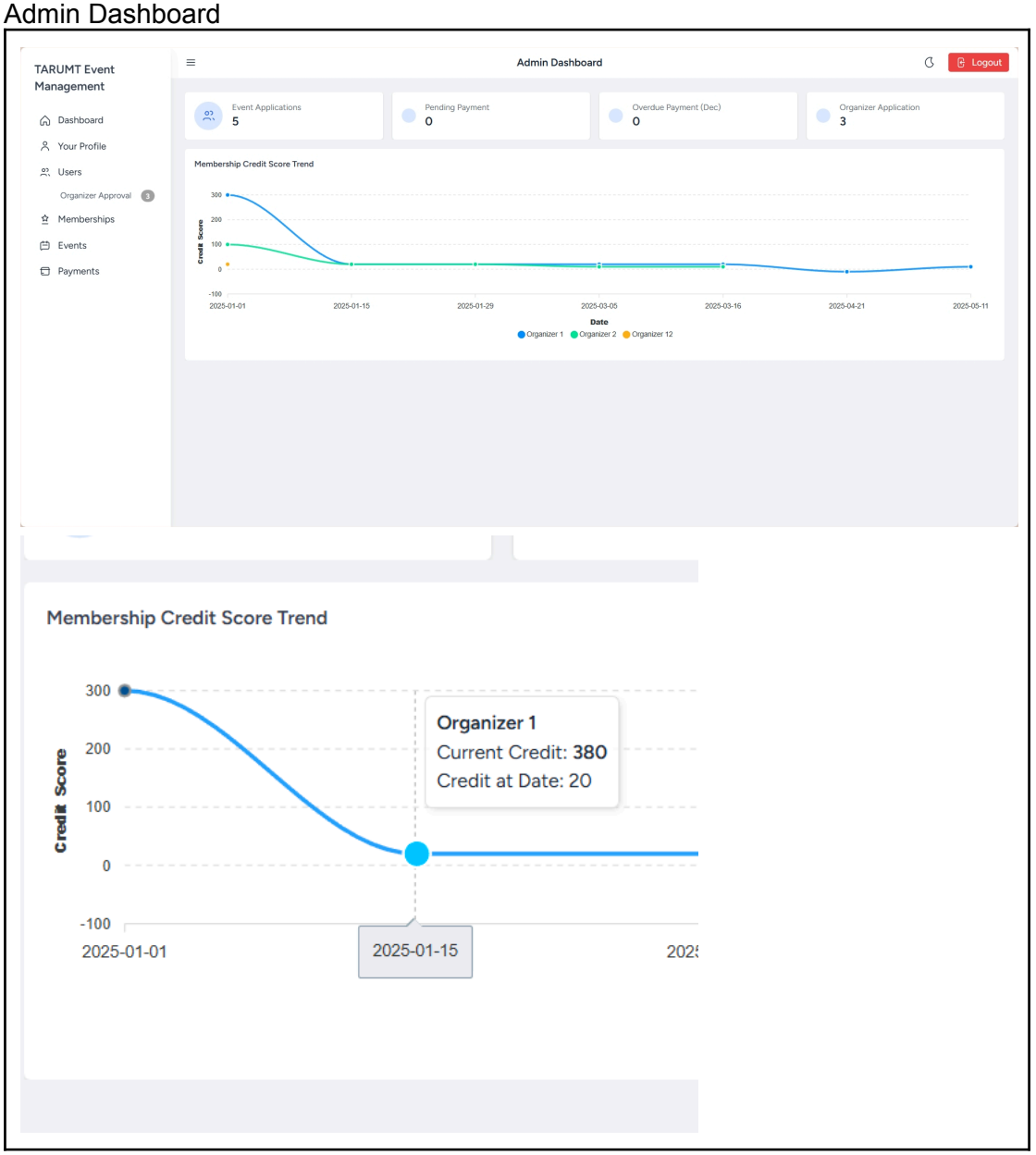


Figure 2.1.11 Admin Dashboard (Class Path: resources/views/admin/dashboard.blade.php)

The dashboard summary cards provide a quick overview of key system statuses, including the number of pending event applications, pending payments, overdue payments for the current month, and organizer applications awaiting approval.

The Membership Credit Score Trend displays a line chart on the Admin Dashboard showing the accumulated credit score of each organizer over time. Each colored line represents a different organizer. The chart updates cumulatively based on credit rewards and penalties. When hovering over a data point, the system displays the organizer name, the current accumulated credit score, and the credit score at that specific date. This allows administrators to quickly monitor organizer performance

and credit changes.

- View list of users

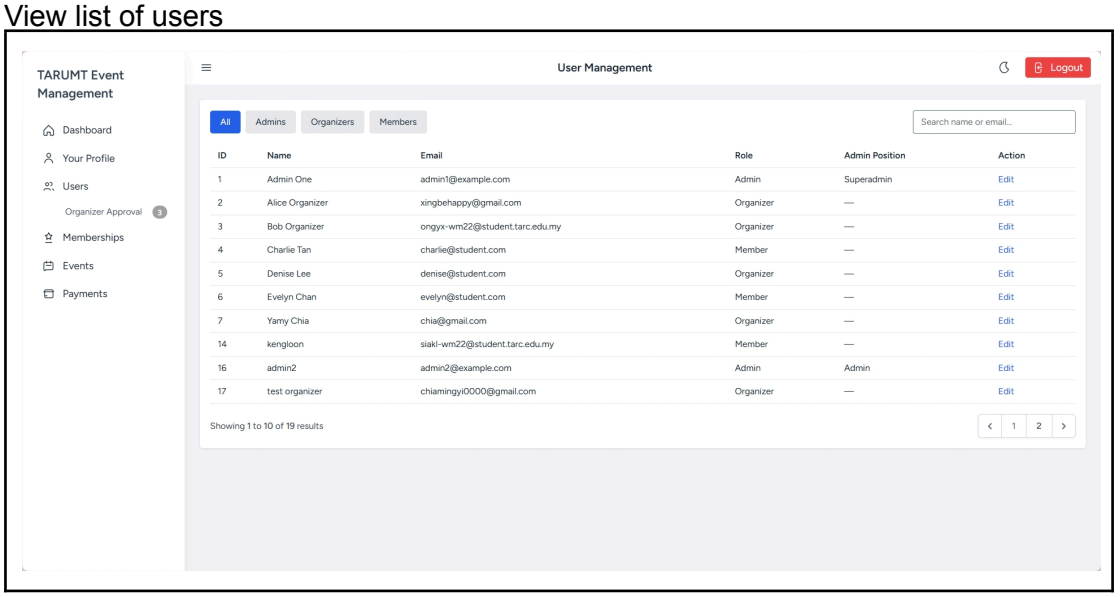


Figure 2.1.12 View User List (Class Path: resources/views/livewire/admin/admin-user-table.blade.php)

The User Management interface allows administrators to view basic information of all system users in a centralized table, where only super administrators are permitted to manage user details. Users can be filtered by role (All, Admins, Organizers, Members) using role tabs and searched by name or email for quick access. The table displays key information such as user ID, name, email, role, and admin position, with an Edit action available for authorized management. Pagination is implemented to efficiently handle large user lists.

- Review Application of Member to become Organizer

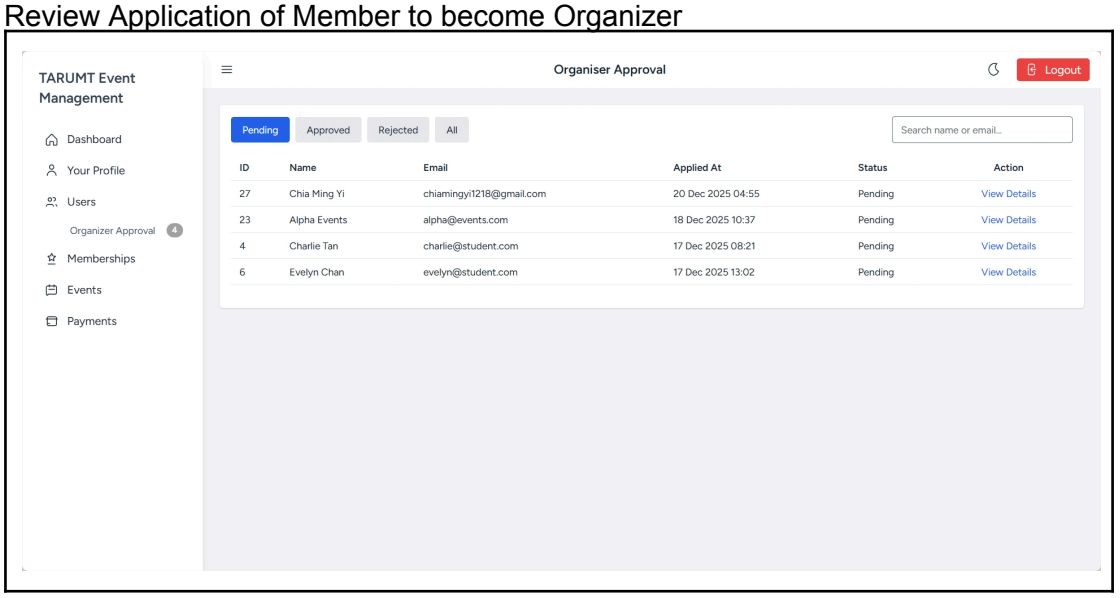


Figure 2.1.13: Review Application of Member to become Organizer (Class Path: resources/views/livewire/admin/admin-organizer-approval-table.blade .php)

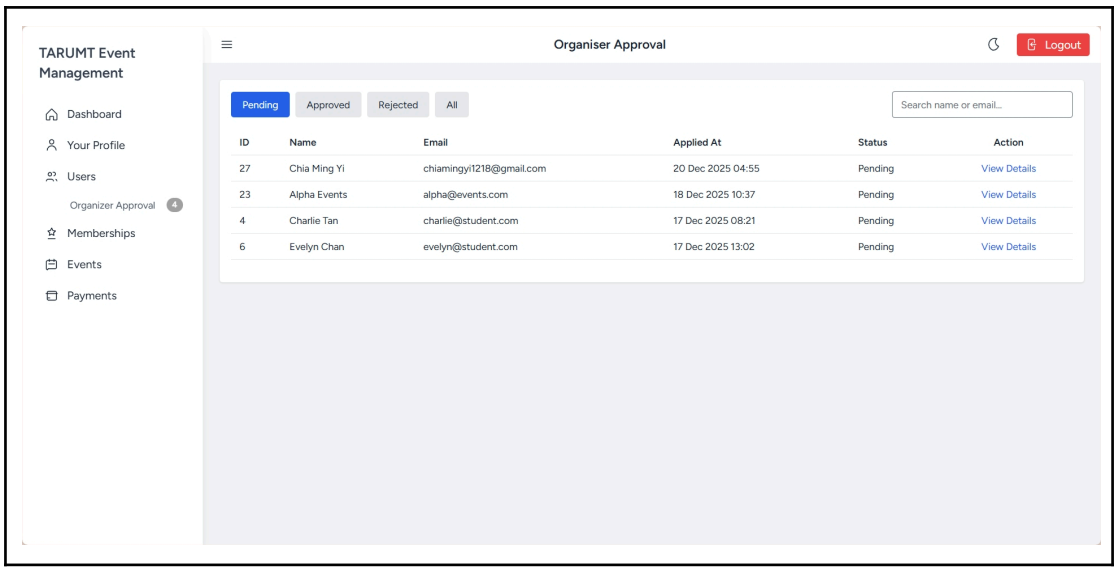


Figure 2.1.14: Review Application Details of Member to become Organizer (Class Path: resources/views/livewire/admin/pending-detail.blade.php)

This interface allows administrators to review and manage applications submitted by members who wish to become organizers. The listing page displays all applications in a structured table, with tabs to filter applications by status (Pending, Approved, Rejected, or All) and a search bar for quick lookup by name or email. Key information such as applicant name, email, application date, and current status is clearly presented, with a View Details action for further review.
Upon selecting an application, the detail page shows the applicant's full information, including personal or company details, registration number, submission time, and supporting documents. Administrators can review the documents directly and take action using the Approve or Reject buttons, enabling an efficient and controlled approval workflow.

2.1.5 Super Admin

Super Admins have the highest level of authority in the Users Module.

Functions handled:

- Create and Edit User

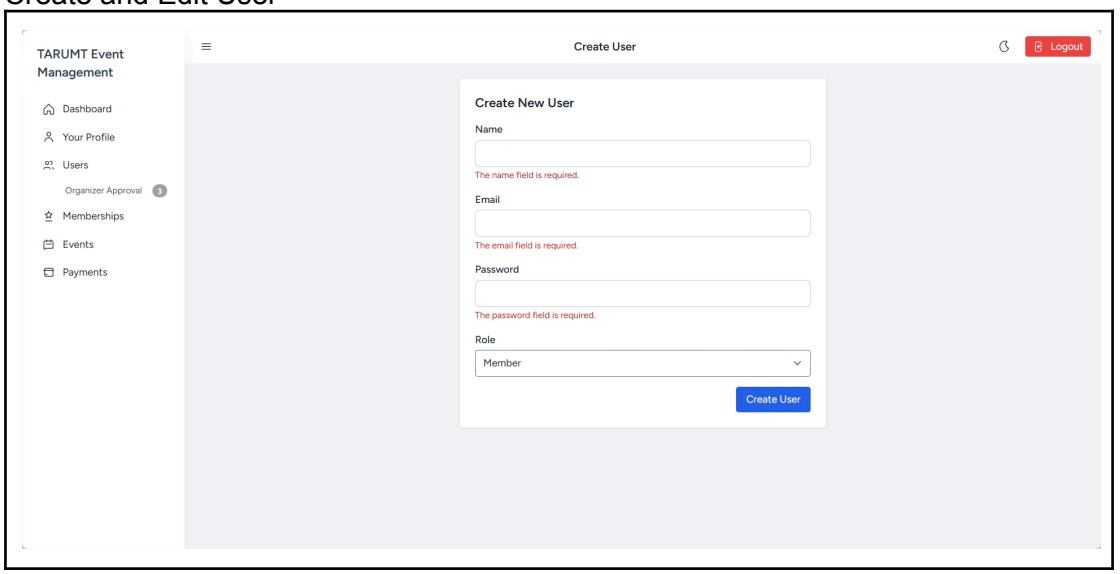


Figure 2.1.15: Super Admin Create User (Class Path: resources/views/livewire/admin/create-user.blade.php)

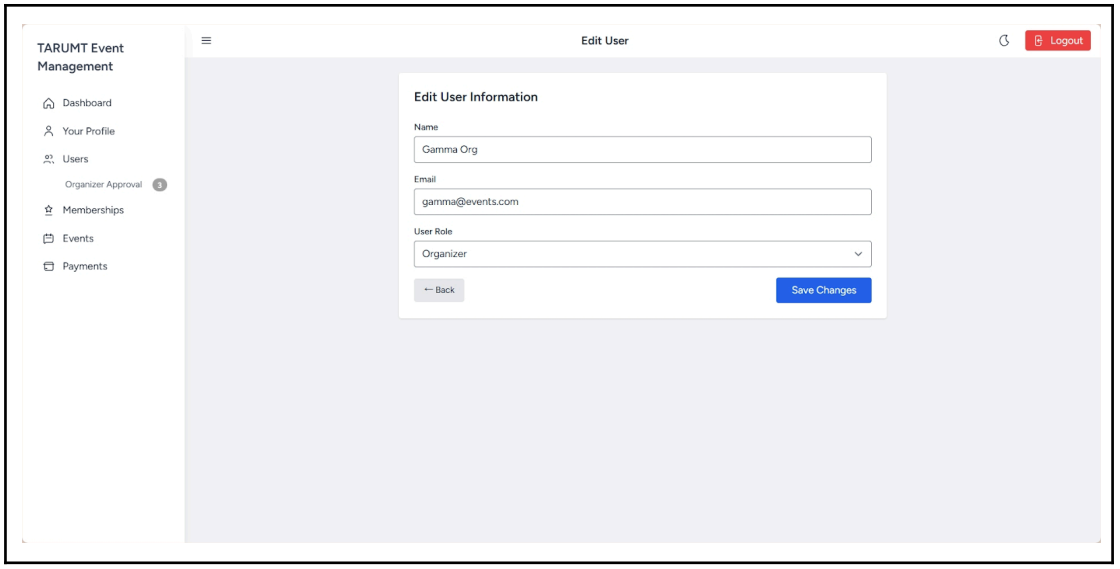
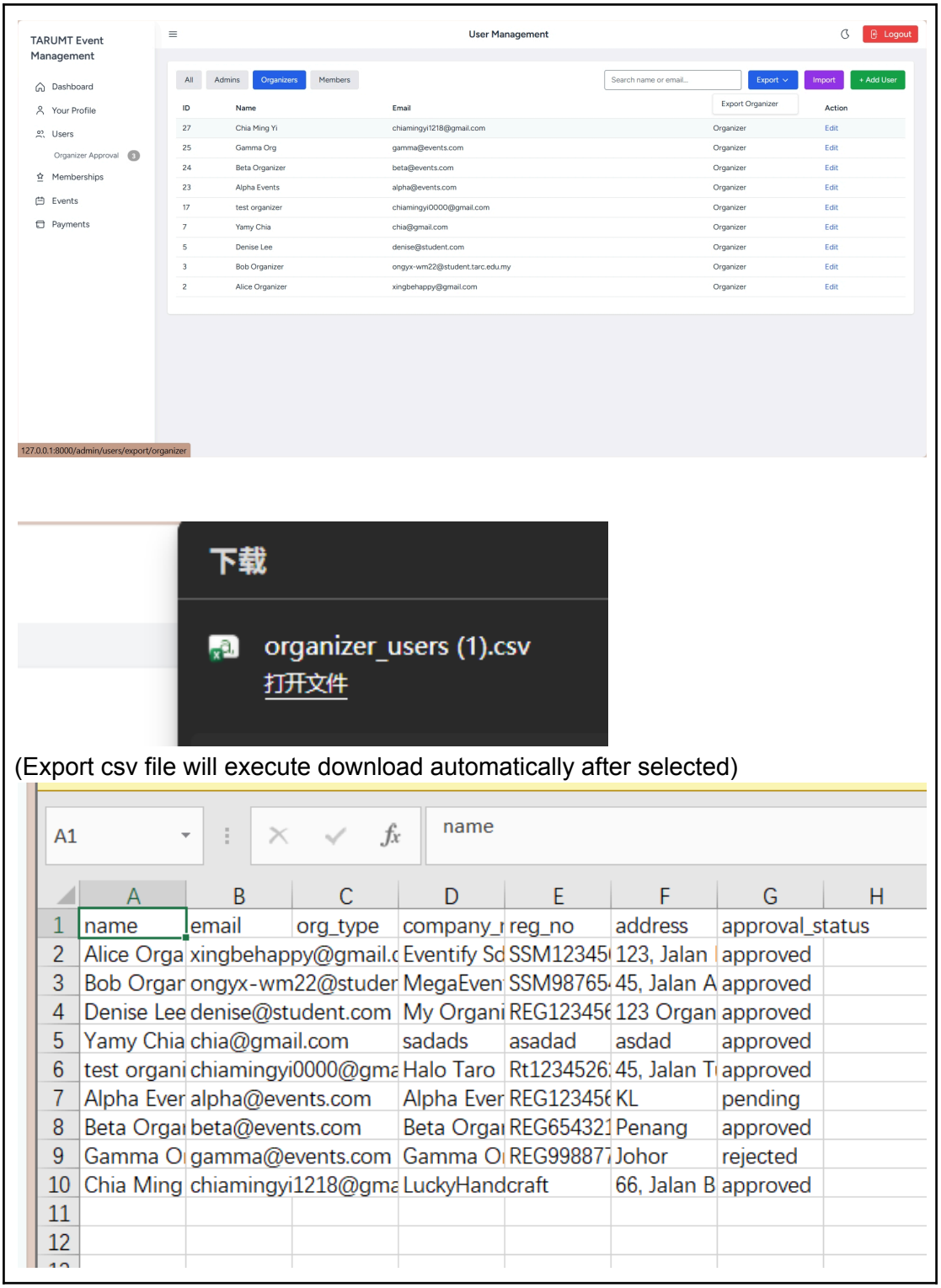


Figure 2.1.16: Super Admin Edit User (Class Path: resources/views/livewire/admin/edit-user.blade.php)

The User Management module allows only super administrators to create new users, edit existing user information. The Create User form enables admins to register users by providing name, email, password, and role. The Edit User page allows updating basic user details such as name, email, and assigned role.

- Export Users



(Export csv file will execute download automatically after selected)

Figure 2.1.17: Super Admin Export User (Class Path: resources/views/livewire/admin/admin-user-table.blade.php)

The User Management interface provides an export function that allows super administrators to download user data in CSV format. By selecting the user tab and using the Export action, the system generates a spreadsheet containing key information such as name, email, organization type, company details, registration number, address, and approval status. This feature supports tasks such as reporting, auditing, and offline data review.

- Import Users

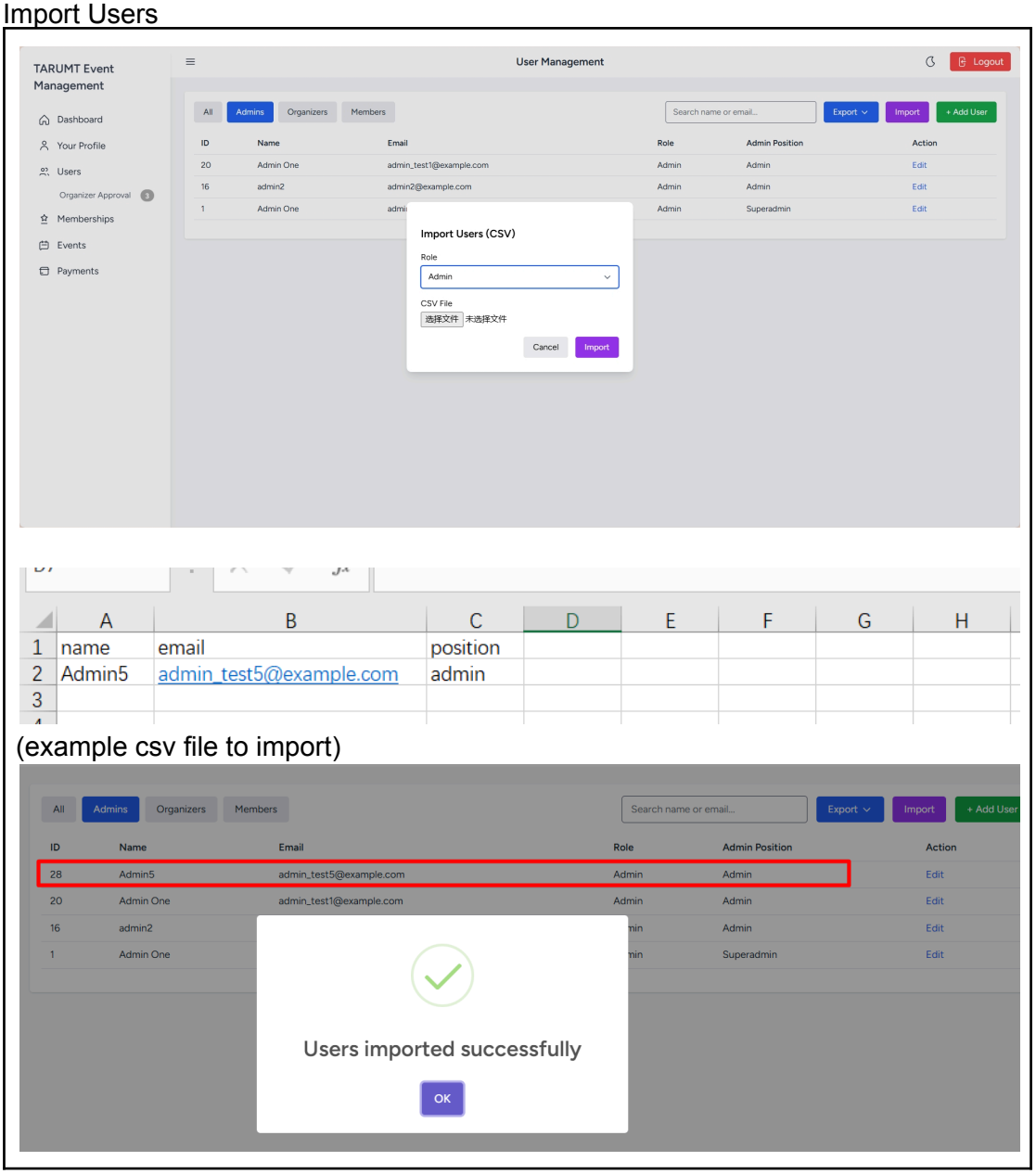


Figure 2.1.18: Super Admin Import Users (Class Path: resources/views/livewire/admin/admin-user-table.blade.php)

The User Management module includes an import function that allows super administrators to bulk add users using a CSV file. Through the Import button, admins select the target user role (e.g., Admin, Organizer, Member) and upload a properly formatted CSV file containing user details. Upon successful import, the system automatically creates the user accounts and displays a confirmation message, with the newly imported users immediately visible in the user list. This feature improves efficiency when managing large numbers of users.

2.2 Memberships Module

The Memberships Module manages membership plans, membership credits, and membership activity logs. This module is accessible only by Organizers and Admins.

2.2.1 Membership Management

This function allows Admins(Admin and Super Admin) to manage membership plans offered to organizers.

- Manage Membership of Organizer

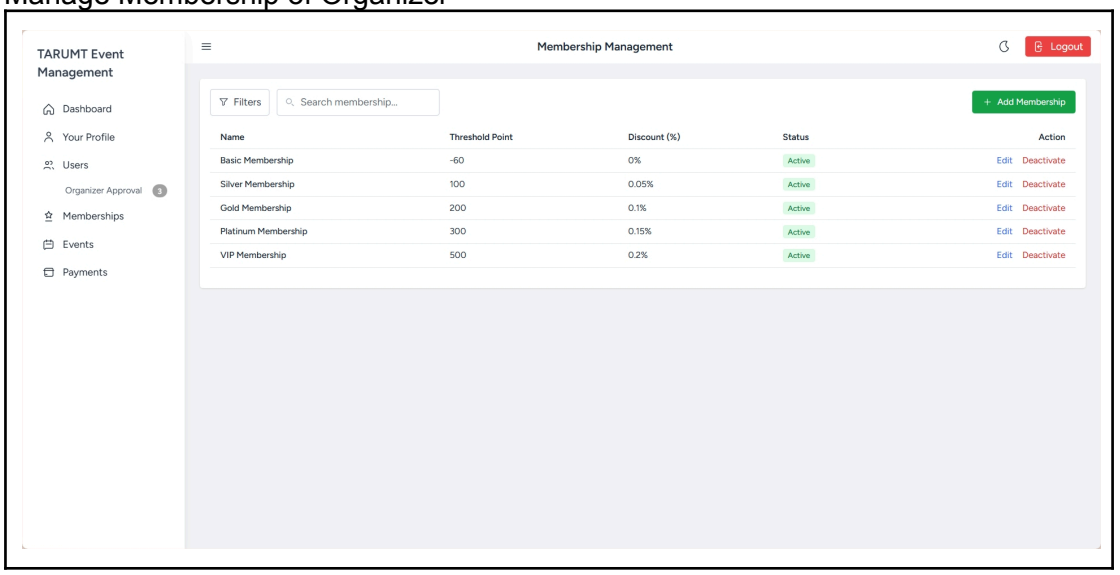


Figure 2.2.1: Membership Management for Admins (Class Path: resources/views/livewire/admin/membership-table.blade.php)

These interfaces allow administrators to manage organizer membership levels through a centralized table and form-based pages. The membership list displays key details such as membership name, credit threshold, discount rate, and current status, with actions to edit or deactivate each level. Administrators can create new membership levels or modify existing ones by defining threshold points, discount percentages, duration, and status. This ensures organizer privileges and benefits are consistently controlled based on their accumulated credit score.

2.2.2 Membership Logs and Credit Management

This function tracks organizer membership activity and calculates membership levels based on accumulated credit scores.

Functions handled:

- Membership of Organizer

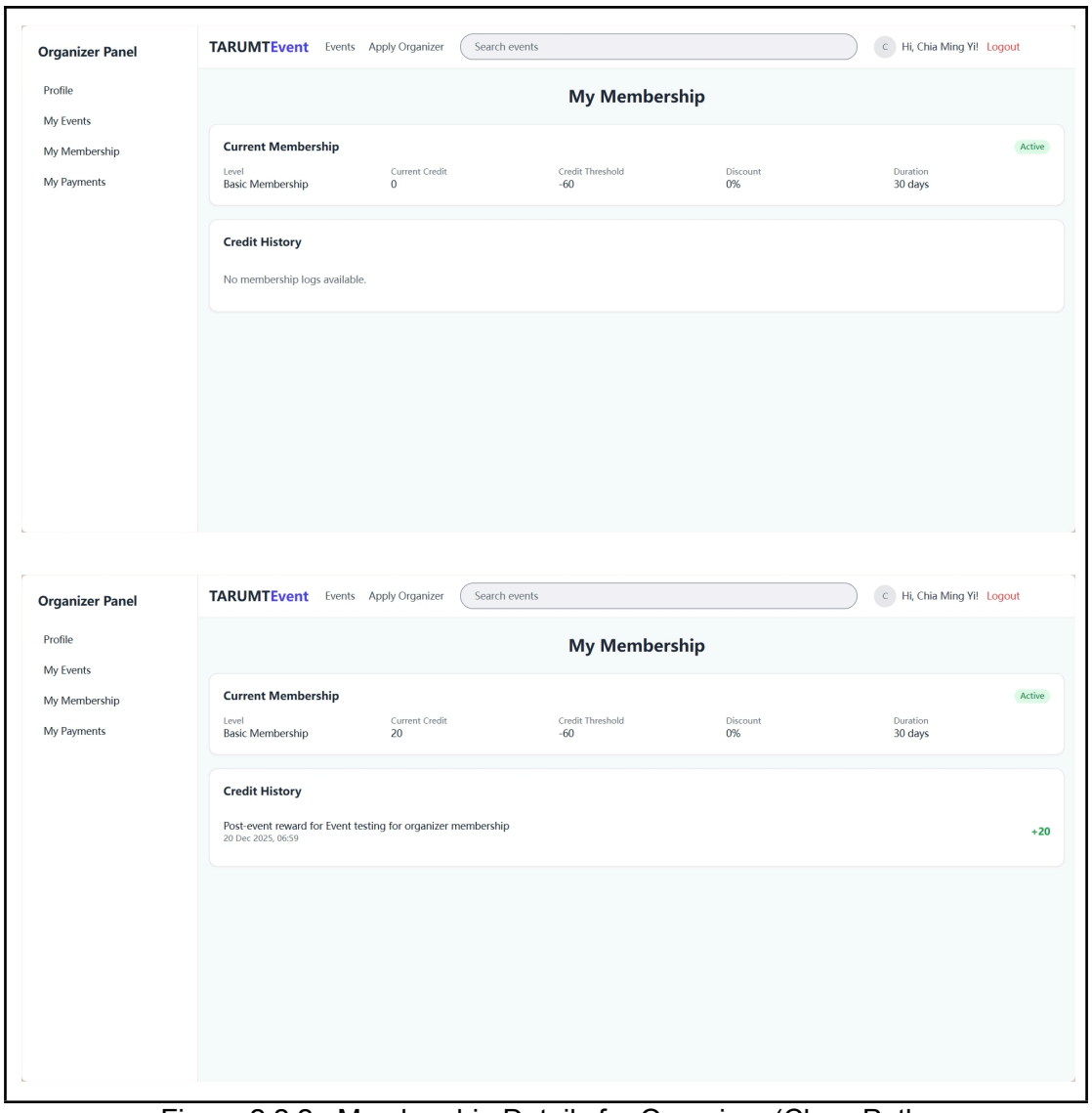


Figure 2.2.2: Membership Details for Organizer (Class Path: resources/views/organizer/membership/index.blade.php)

Organizers receive +20 membership credit points upon successful event completion. Penalty deductions are applied according to severity: -50 points (serious), -20 points (medium), and -10 points (low).

3. Entity Classes

Entity Class Diagram

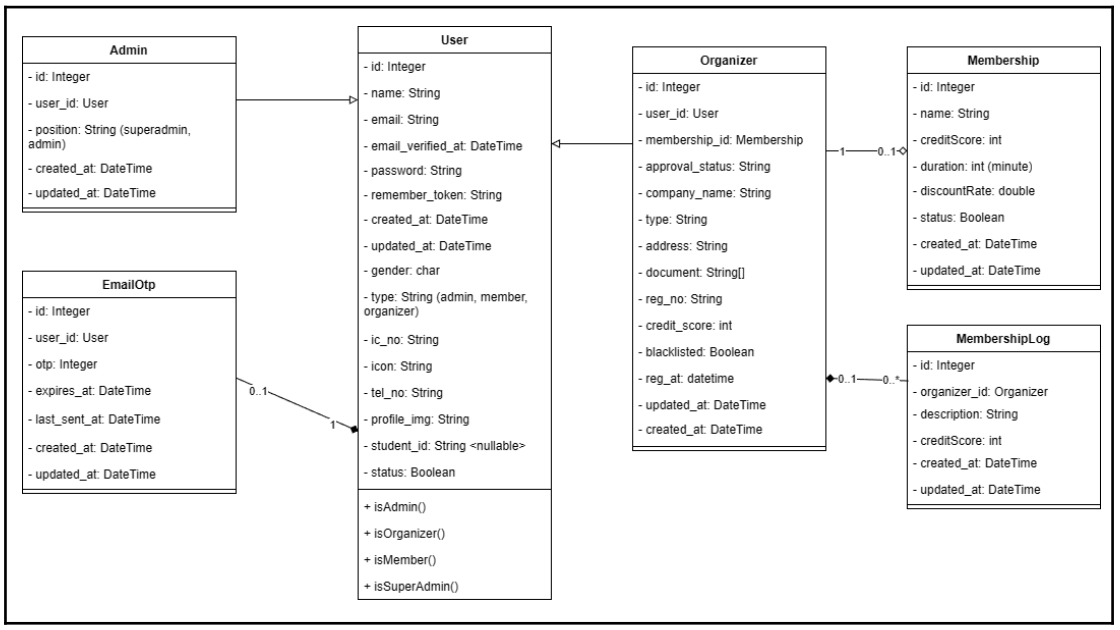


Figure 3.1: Entity Class Diagram of User and Membership Module

Entity Class Descriptions

The entity classes under my responsibility belong to the User Module, which manages identity, authentication, role assignment, organizer membership, and email verification within the TARUMT Event Management System. These entity classes include User, Admin, Organizer, Membership, MembershipLog, and EmailOtp. The design and implementation strictly follow entity class diagram principles, where relationships are represented using object references rather than foreign keys, ensuring a clear separation between domain modeling and database concerns.

In the entity class diagram, associations between entities are modeled using class-to-class relationships with defined multiplicities, rather than database-level foreign keys. This approach is reflected in the Laravel implementations through Eloquent relationship methods such as hasOne, belongsTo, and hasMany.

For example, the User entity maintains object references to Admin and Organizer entities, indicating that a user may optionally assume an administrative or organizer role. This is implemented using one-to-one object relationships, allowing role-specific behavior to be encapsulated within their respective entity classes. Similarly, the Organizer entity holds a direct object reference to the Membership entity, representing the organizer's current membership tier, while avoiding direct manipulation of membership identifiers within business logic.

The MembershipLog entity is associated with the Organizer entity through a one-to-many relationship, enabling each organizer to maintain a history of credit score changes and membership-related updates. This design ensures that membership progression and score adjustments are traceable and logically grouped under the organizer domain object. The EmailOtp entity is associated with the User

entity through an object reference, allowing OTP records to be linked to a single user for email verification and password recovery workflows.

4. Design Pattern

4.1 Description of Design Pattern

The Factory design pattern is implemented within the User module to manage the creation of role-specific objects and user import/export handlers. Its purpose is to centralize object creation logic and ensure that the correct implementation is instantiated based on the user role, without exposing concrete class details to the calling code.

By delegating object creation to factory classes, the system avoids scattered conditional logic and ensures consistent behavior when handling different user roles or import/export strategies. This design improves maintainability and allows new roles or strategies to be added with minimal impact on existing code.

UserRoleFactory

- UserRoleFactory serves as the central factory responsible for creating user role objects. It exposes a factory method create() that returns a UserRole interface implementation based on the provided role type.
- When a role string such as admin, organizer, or member is passed into the factory, the factory instantiates and returns the corresponding concrete role class (AdminRole, OrganizerRole, or MemberRole). This ensures that role-specific permissions and behaviors are encapsulated within their respective classes rather than being handled through conditional logic in controllers or services.

UserImportExportFactory

- UserImportExportFactory applies the same factory approach to user import and export operations. The factory method make() returns a concrete implementation of the UserImportExport interface according to the selected user role.
- Each concrete class (AdminUserImportExport, OrganizerUserImportExport, MemberUserImportExport) handles role-specific import and export logic, while the calling code interacts only with the common interface. This design ensures consistent handling of CSV import and export operations while supporting role-based customization.

4.2 Implementation of Design Pattern

In this system, the Factory pattern is implemented to handle user roles and user import/export operations dynamically.

User Role Factory Class Diagram

The UserRoleFactory class is responsible for creating different user role objects (AdminRole, OrganizerRole, MemberRole) based on the provided role string.

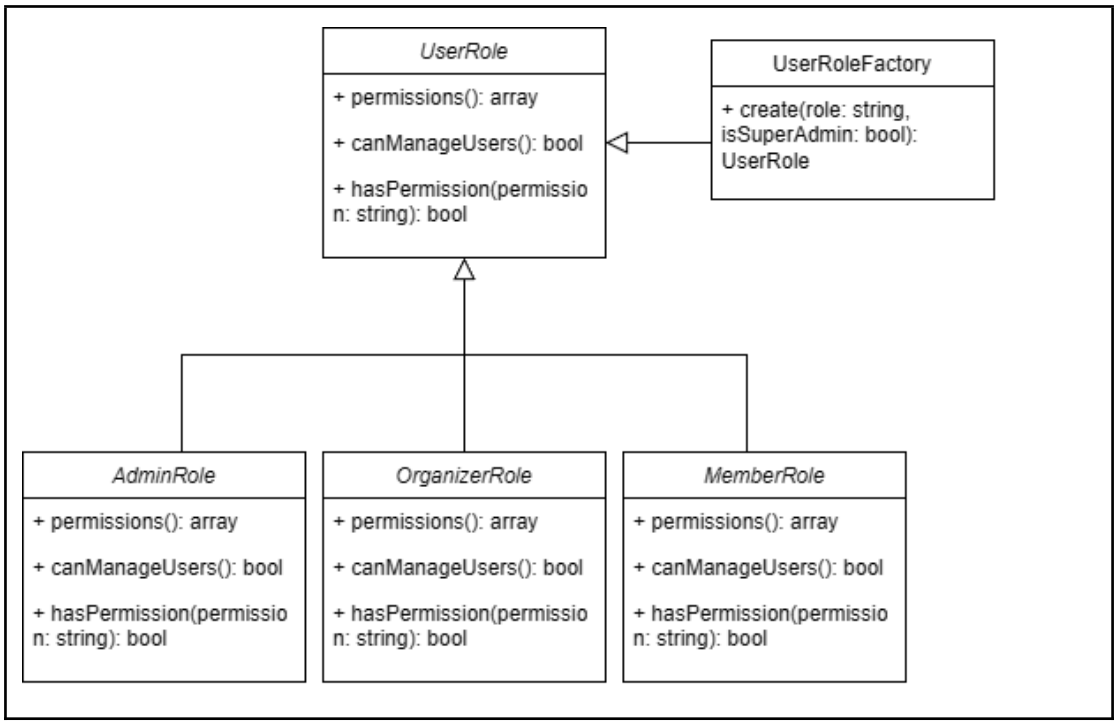


Figure 4.2.1: Class Diagram of User Role Factory

The class diagram illustrates how the factory classes create objects through a common interface. Concrete product classes implement the interface, while the factory depends on the abstract type rather than concrete implementations.

User Import/Export Factory Class Diagram

The UserImportExportFactory class creates the correct import/export handler according to the selected user role during CSV import or export operations.

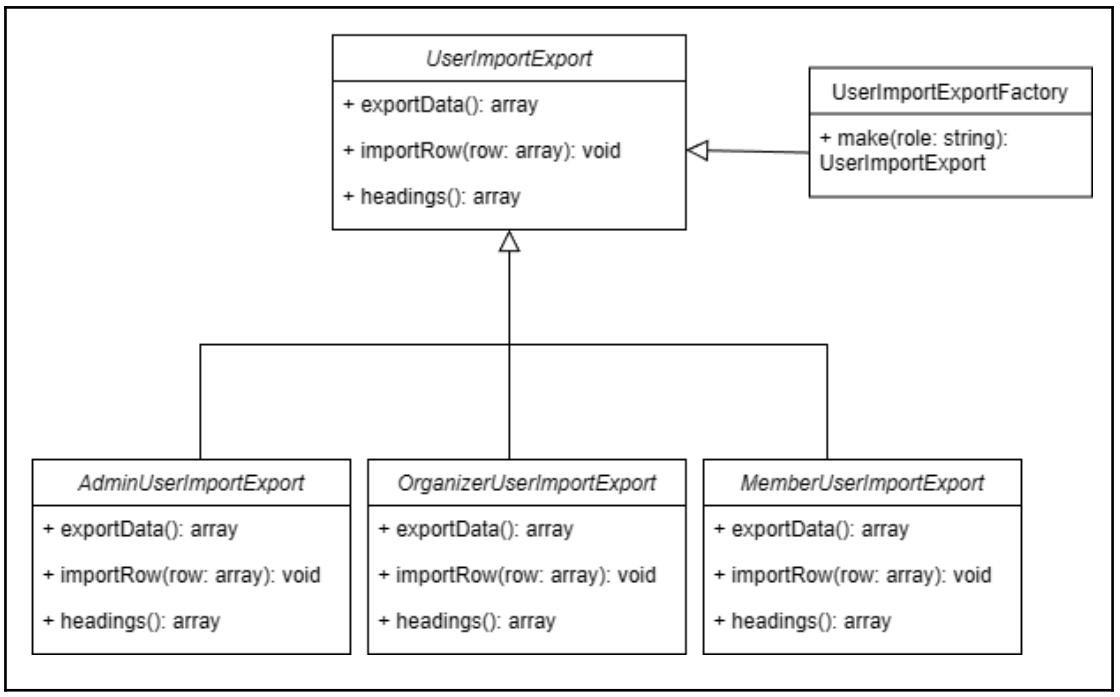


Figure 4.2.2: Class Diagram of User Import/Export Factory

The class diagram illustrates how the factory classes create objects through a common interface. Concrete product classes implement the interface, while the factory depends on the abstract type rather than concrete implementations.

Justification for Using Factory Pattern
The Factory pattern was chosen because:

- Centralized Object Creation:
  The system supports multiple user roles and role-specific behaviors. The Factory Method pattern centralizes the creation of these objects, preventing repeated conditional logic across controllers and services.

- Decoupling of Business Logic:
  Controllers and services depend only on interfaces (UserRole, UserImportExport) rather than concrete classes. This reduces tight coupling and makes the system easier to modify and extend.

- Support for Role-Based Variations:
  Different user roles require different permissions and import/export behavior. The Factory Method pattern allows these variations to be handled cleanly through separate concrete classes without complicating the calling code.

- Extensibility for Future Requirements:
  New user roles or import/export strategies can be introduced by adding new concrete classes and updating the factory, without modifying existing business logic. This follows the Open–Closed Principle and minimizes regression risks.

5. Software Security

5.1 Potential Threat/Attack

1. Credential Theft

Credential theft refers to attacks where malicious actors obtain valid user authentication credentials, such as usernames and passwords, through unauthorized means. This threat may occur due to database compromise, credential harvesting, or the reuse of leaked credentials from other platforms (credential stuffing). If user passwords are stored insecurely or authentication mechanisms reveal excessive information, attackers may successfully obtain credentials and gain unauthorized access to user accounts.

In the context of the Users and Membership module, credential theft poses a significant risk because compromised credentials can allow attackers to impersonate legitimate users, access sensitive personal information, and misuse membership-related privileges. Once credentials are stolen, attackers may bypass authentication controls entirely, leading to unauthorized actions within the system. Therefore, protecting stored credentials and minimizing opportunities for credential harvesting are critical to maintaining system security.

2. Session Hijacking

Session hijacking is an attack in which an attacker gains access to a valid session identifier and uses it to impersonate an authenticated user without needing to know the user's credentials. This can occur if session identifiers are exposed through insecure transmission, predictable session values, improper session handling, or failure to invalidate sessions when authentication states change.

For the Users and Membership module, session hijacking represents a serious threat because authenticated sessions grant direct access to user accounts and membership functionalities. If an attacker successfully hijacks a session, they may perform actions on behalf of the user, such as viewing protected data or modifying account-related information. Effective session management controls are therefore required to ensure that session identifiers are securely generated, regenerated, and invalidated throughout the authentication lifecycle.

5.2 Secure Coding Practice

1. Authentication & Password Management

To mitigate the threat of credential theft, the system implements secure authentication and password management controls using Laravel's built-in authentication framework. The system ensures that user passwords are never stored in plain text by applying cryptographically strong one-way hashing on the server side. In addition, authentication failure responses are designed to be generic and do not disclose whether the email address or password is incorrect, preventing attackers from identifying valid user accounts. Login rate limiting is also enforced to restrict automated attempts to test stolen credentials, reducing the effectiveness of credential stuffing attacks.

By combining secure password hashing, generic authentication error handling, and login rate limiting, the system effectively mitigates credential theft. These layered authentication controls significantly reduce the risk of unauthorized access to user

accounts within the Users and Membership module.

```
$request->user()->update([
    'password' => Hash::make($validated['password']),
]);
```

Figure 5.1.1: Code snippet from app/Http/Controllers/Auth/PasswordController.php

In alignment with OWASP Secure Coding Practices [30] and [31], the application stores only cryptographically strong one-way salted password hashes. Password hashing is performed exclusively on a trusted server-side environment using Laravel's Hash facade, which applies the bcrypt algorithm. This ensures that sensitive authentication credentials remain protected even if the database is compromised.

```
throw ValidationException::withMessages([
    'email' => trans('auth.failed'),
]);
```

Figure 5.1.2: Code snippet from app/Http/Requests/Auth/LoginRequest.php

According to OWASP Secure Coding Practice [33], authentication failure responses should not reveal which part of the credentials is incorrect. The system returns a generic authentication error message regardless of whether the email or password is invalid, preventing username enumeration and reducing the risk of credential harvesting attacks.

```
if (! RateLimiter::tooManyAttempts($this->throttleKey(), 5)) {
    return;
}
```

Figure 5.1.3: Code snippet from app/Http/Requests/Auth/LoginRequest.php

In accordance with OWASP Secure Coding Practice [41], the system enforces rate limiting on authentication attempts. This control restricts repeated login attempts, making it more difficult for attackers to test stolen credentials through automated credential stuffing attacks.

2. Session Management

To mitigate the risk of session hijacking, the system implements secure session management controls using Laravel's built-in session handling mechanism. Session hijacking occurs when attackers obtain a valid session identifier and impersonate an authenticated user without knowing their credentials. This threat is particularly critical for the Users and Membership module, as compromised sessions may grant unauthorized access to user accounts and membership-related functionality.

By combining session regeneration, secure session termination, and server-side session handling, the system effectively mitigates session hijacking risks. These session management controls ensure that authenticated sessions within the Users and Membership module remain secure throughout the user authentication lifecycle.

```php
public function store(LoginRequest $request): RedirectResponse
{
    $request->authenticate();
    $request->session()->regenerate();
```

Figure 5.2.1: Code snippet from app/Http/Controllers/Auth/AuthenticatedSessionController.php

In alignment with OWASP Secure Coding Practices [66] and [67], the system regenerates the session identifier immediately after successful authentication. This prevents session fixation attacks, where an attacker attempts to force a user to use a known session identifier prior to login. By issuing a new session identifier upon authentication, any previously exposed or attacker-controlled session ID becomes invalid.

```php
public function destroy(Request $request): RedirectResponse
{
    Auth::guard('web')->logout();

    $request->session()->invalidate();
    $request->session()->regenerateToken();

    return redirect('/');
}
```

Figure 5.2.2: Code snippet from app/Http/Controllers/Auth/AuthenticatedSessionController.php

According to OWASP Secure Coding Practice [62], logout functionality should fully terminate the associated session. The system explicitly invalidates the active session and regenerates the session token during logout, ensuring that session data cannot be reused after the user signs out. This reduces the risk of session reuse by unauthorized parties.

```php
'driver' => env('SESSION_DRIVER', 'database'),
'secure' => env('SESSION_SECURE_COOKIE'),
'http_only' => env('SESSION_HTTP_ONLY', true),
'same_site' => env('SESSION_SAME_SITE', 'lax'),
```

Figure 5.2.3: Code snippet from config/session.php

Laravel's session management framework stores session data on the server side and maintains only a session identifier within an HTTP cookie. As shown in the session configuration, the application uses a server-side session driver and enforces HTTP-only cookies to prevent client-side scripts from accessing session identifiers. Session cookies are also configured with secure and SameSite attributes, ensuring that session identifiers are transmitted only under appropriate conditions.
This implementation aligns with OWASP Secure Coding Practice [69], which recommends preventing session identifiers from being exposed through URLs, logs, or client-side parameters, thereby reducing the risk of session hijacking.

6. Web Services

1. Service Exposure

The Users Module and Membership Module exposes RESTful web services to allow other system modules to access user authentication, authorization, and membership-related information in a standardized and loosely coupled manner. These services are implemented using JSON-based REST APIs, providing a lightweight and flexible communication mechanism suitable for modern web-based systems.

The exposed services are designed to be consumed by other modules such as the Event Module, Payment Module for purposes including user authentication, role validation, and access control enforcement. By centralizing user and membership management, the system ensures consistent identity verification and authorization decisions across all modules.

All exposed web services comply with the Interface Agreement (IFA) standards by enforcing mandatory request timestamps and returning structured JSON responses. This ensures traceability, consistency, and interoperability between modules during inter-module communication.

Compared to SOAP-based services, the RESTful architecture was selected due to its simplicity, lower communication overhead, and ease of integration across heterogeneous platforms, making it more suitable for a scalable and modular system architecture.

```php
// User API Route
Route::get( uri: '/organizers/event', [UserController::class, 'getEventOrganizers'])->name( name: 'eventOrganizers');
Route::get( uri: '/organizers/bulk', [UserController::class, 'getByOrganizerIds']);
Route::get( uri: '/organizers/{organizerId}', [UserController::class, 'getOrganizerDetailById']);
Route::get( uri: '/organizers/{organizerId}/membership', [MembershipController::class, 'getOrganizerMembership']);
Route::get( uri: '/organizers/{organizer}/is-blacklisted', [UserController::class, 'isBlacklisted']);
Route::get( uri: '/users/{user}/organizer', [UserController::class, 'getUserOrganizer']);
Route::post( uri: '/users/batch', [UserController::class, 'getUsersByIds']);
Route::post( uri: '/organizers/blacklistStatus', [UserController::class, 'setBlacklistStatus']);
Route::post( uri: '/organizers/membershipLog', [MembershipController::class, 'setMembershipLog']);
Route::post( uri: '/organizers/addOrDeductCreditScore', [MembershipController::class, 'addOrDeductCreditScore']);
```

Figure 6.1: API Route Provided from User Module (Class Path: \routes\api\php)

2. Webservice Mechanism

Get Event Organizers

	Description
Protocol	RESTFUL
Function Description	*Retrieves a list of organizers who are eligible to be associated with events. The service returns organizer information in a standardized JSON format compliant with the Interface Agreement (IFA) standard.*
Source Module	*Users Module*
Target Module	*Event Module*
URL	*http://127.0.0.1:8000/api/organizers/event*

Function Name	*getEventOrganizers*

Get Organizers by Organizer IDs (Bulk)

	Description
Protocol	RESTFUL
Function Description	*Retrieves user information for multiple organizers based on a list of organizer IDs. This bulk service supports efficient cross-module data retrieval and returns structured JSON responses.*
Source Module	*Users Module*
Target Module	*Event Module, Payment Module*
URL	*http://127.0.0.1:8000/api/organizers/bulk?ids={organizerIds}*
Function Name	*getByOrganizerIds*

Get Organizer Detail by ID

	Description
Protocol	RESTFUL
Function Description	*Retrieves detailed organizer profile information for a specific organizer ID. This service supports organizer verification and detail display in other system modules.*
Source Module	*Users Module*
Target Module	*Event Module*
URL	*http://127.0.0.1:8000/api/organizers/{organizerId}*
Function Name	*getOrganizerDetailById*

Get Organizer Membership Information

	Description
Protocol	RESTFUL
Function Description	*Retrieves the current membership details of an organizer, including membership level and related benefits. This service supports discount calculation and membership-based logic in other modules.*
Source Module	*Users Module*
Target Module	*Payment Module, Event Module*
URL	*http://127.0.0.1:8000/api/organizers/{organizerId}/membership*
Function Name	*getOrganizerMembership*

Check Organizer Blacklist Status

	Description
Protocol	RESTFUL

Function Description	*Checks whether a specific organizer is blacklisted. The service returns a boolean flag indicating the blacklist status to prevent restricted organizers from performing unauthorized actions.*
Source Module	*Users Module*
Target Module	*Payment Module, Event Module*
URL	*http://127.0.0.1:8000/api/organizers/{organizerId}/is-blacklisted*
Function Name	*isBlacklisted*

Get Organizer Profile by User ID

	Description
Protocol	RESTFUL
Function Description	*Retrieves the organizer profile associated with a given user ID. This service is used to validate whether a user has an organizer role and retrieve organizer-related metadata.*
Source Module	*Users Module*
Target Module	*Event Module*
URL	*http://127.0.0.1:8000/api/users/{userId}/organizer*
Function Name	*getUserOrganizer*

Get Users by User IDs (Batch)

	Description
Protocol	RESTFUL
Function Description	*Retrieves basic user profile information for a batch of users based on a list of user IDs. The service supports efficient bulk user data retrieval in cross-module operations.*
Source Module	*Users Module*
Target Module	*Event Module*
URL	*http://127.0.0.1:8000/api/users/batch*
Function Name	*getUsersByIds*

Update Organizer Blacklist Status

	Description
Protocol	RESTFUL
Function Description	*Updates the blacklist status of an organizer. This service enforces administrative control over organizer eligibility and ensures consistent restriction enforcement across modules.*
Source Module	*Users Module*
Target Module	*Users Module (Admin)*
URL	*http://127.0.0.1:8000/api/organizers/blacklistStatus*

Function Name	*setBlacklistStatus*

Create Organizer Membership Log

	Description
Protocol	RESTFUL
Function Description	*Creates a membership activity log for an organizer, recording credit score changes and related descriptions. This service ensures traceable membership history management.*
Source Module	*Users Module*
Target Module	*Payment Module, Users Module (Admin)*
URL	*http://127.0.0.1:8000/api/organizers/membershipLog*
Function Name	*setMembershipLog*

Add or Deduct Organizer Credit Score

	Description
Protocol	RESTFUL
Function Description	*Updates an organizer's credit score by adding or deducting points based on system-defined actions. The service ensures membership level recalculation and consistent credit enforcement.*
Source Module	*Users Module*
Target Module	*Payment Module, Event Module*
URL	*http://127.0.0.1:8000/api/organizers/addOrDeductCreditScore*
Function Name	*addOrDeductCreditScore*

Web Services Request Parameter (provide)

Retrieve Event Organizers

Field Name	Field Type	Mandatory/ Optional	Description	Format
timestamp	String	Mandatory	Time when the request was made	ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ)

Get Organizers by Organizer IDs (Bulk)

Field Name	Field Type	Mandatory/ Optional	Description	Format
timestamp	String	Mandatory	Time when the request was made	ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ)
ids	String	Mandatory	Comma-separated list of organizer IDs	Example: 1,2,3,5

Get Organizer Detail by ID

Field Name	Field Type	Mandatory/ Optional	Description	Format
organizerId	Integer	Mandatory	Unique identifier of the organizer	Numeric
timestamp	String	Mandatory	Time when the request was made	YYYY-MM-DD HH:MM:SS

Get Organizer Membership Information

Field Name	Field Type	Mandatory/ Optional	Description	Format
organizerId	Integer	Mandatory	Unique identifier of the organizer	Numeric
timestamp	String	Mandatory	Time when the request was made	ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ)

Check Organizer Blacklist Status

Field Name	Field Type	Mandatory/ Optional	Description	Format

organizerId	Integer	Mandatory	Unique identifier of the organizer	Numeric
timestamp	String	Mandatory	Time when the request was made	ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ)

Get Organizer Profile by User ID

Field Name	Field Type	Mandatory/ Optional	Description	Format
userId	Integer	Mandatory	Unique identifier of the user	Numeric
timestamp	String	Mandatory	Time when the request was made	ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ)

Get Users by User IDs (Batch)

Field Name	Field Type	Mandatory/ Optional	Description	Format
timestamp	String	Mandatory	Time when the request was made	ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ)
user_ids	Array	Mandatory	List of user IDs to retrieve	Example: [1, 3, 5]
user_ids.*	Integer	Mandatory	Individual user ID	Numeric

Update Organizer Blacklist Status

Field Name	Field Type	Mandatory/ Optional	Description	Format
timestamp	String	Mandatory	Time when the request was made	ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ)
organizerId	Integer	Mandatory	Unique identifier of the organizer	Numeric
blacklisted	Boolean	Mandatory	Blacklist status flag	true / false

Create Organizer Membership Log

Field Name	Field Type	Mandatory/ Optional	Description	Format
timestamp	String	Mandatory	Time when the request was made	ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ)
organizerId	Integer	Mandatory	Unique identifier of the organizer	Numeric
credit_mark	Integer	Mandatory	Credit score to add or deduct	Integer
description	String	Optional	Description of membership activity	Text

Add or Deduct Organizer Credit Score

Field Name	Field Type	Mandatory/ Optional	Description	Format
timestamp	String	Mandatory	Time when the request was made	ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ)
organizerId	Integer	Mandatory	Unique identifier of the organizer	Numeric
credit_mark	Integer	Mandatory	Credit score to add or deduct	Integer
description	String	Optional	Description of membership activity	Text

Web Services Response Parameter (consume)

Event Module - organizerListEvents

Field Name	Field Type	Mandatory/ Optional	Description	Format
timestamp	String	Mandatory	Time when the request was made	ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ)
status	String	Mandatory	Status of the API request	success / failed
data	Object	Mandatory	List of event records associated with organizers.	Array of event objects
data[].id	Integer	Mandatory	Unique identifier of the event.	Positive integer
data[].event_name	String	Mandatory	Name of the event.	Text
data[].status	String	Mandatory	Current event status.	pending / approved / rejected
data[].event_start	DateTime (ISO 8601)	Mandatory	Event start date and time.	ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ)
data[].event_end	DateTime (ISO 8601)	Mandatory	Event end date and time.	ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ)
meta	Object	Mandatory	Pagination metadata returned by the Event Module.	Object
meta.total	Integer	Mandatory	Total number of event records.	Positive integer

Payment Module - getByFilter

Field Name	Field Type	Mandatory/ Optional	Description	Format
timestamp	String	Mandatory	Time when the request was made	ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ)

status	String	Mandatory	Status of the API request	success / failed
data	Array	Mandatory	List of payment records matching the filter criteria.	Array of payment objects
data[].id	Integer	Mandatory	Unique identifier of the payment record.	Positive integer
data[].organizer_id	Integer	Mandatory	Identifier of the organizer associated with the payment.	Positive integer
data[].event_id	Integer	Mandatory	Identifier of the related event.	Positive integer
data[].type	String	Mandatory	Type of payment.	deposit / penalty
data[].status	String	Mandatory	Current payment status.	pending / paid / overdue
data[].total_amt	Decimal	Mandatory	Total amount required for the payment.	Decimal number (2 decimal places)
data[].received_amt	Decimal	Optional	Amount received from the payer.	Decimal number (2 decimal places)

3. Service Consumption

The Admin Dashboard (Users and Membership context) consumes web services exposed by other system modules to support cross-module data aggregation and administrative monitoring. Instead of directly accessing external module databases, the system communicates through RESTful JSON-based web services, ensuring loose coupling, scalability, and clear module boundaries.

All consumed web services comply with the Interface Agreement (IFA) standards by enforcing mandatory timestamp parameters in requests and returning structured JSON responses containing timestamp and status fields. This approach ensures reliable request–response tracking, traceability, and consistent inter-module communication.

The Laravel HTTP client is used to consume external APIs securely, with authentication handled via Sanctum tokens or internal service headers depending on the target module.

Consuming Event Module Web Service

One key example of service consumption in the Admin Dashboard is the retrieval of pending event applications submitted by organizers. The Admin Dashboard consumes a web service exposed by the Event Module to obtain event application data and calculate summary statistics for administrative decision-making.

The Admin Dashboard sends a GET request to the Event Module API with a mandatory timestamp parameter to comply with IFA standards. The response metadata is then used to determine the total number of pending event applications, which is displayed as a summary metric on the dashboard.

Consumed Web Service Details

- Service Provider Module: Event Module
- Consuming Module: Admin Dashboard (Users & Membership context)
- API Route: /api/events/organizer/list
- HTTP Method: GET
- Purpose: Retrieve a list of organizer events filtered by status for administrative monitoring

```php
// Pending event applications
$eventResponse = $response = Http::withToken($token)
    ->acceptJson()
    ->get(
        url: config( key: 'services.event.url') . '/api/events/organizer/list',
        [
            'timestamp' => now()->toIso8601String(),
            'status' => 'pending',
            'page' => 1,
        ]
    );

$eventApplications = ($eventResponse->successful() || $eventResponse->json( key: 'status') !== 'success')
    ? (int)($eventResponse->json( key: 'meta.total') ?? 0)
    : 0;
```

Figure 6.2: Consuming Event Module API at User Module(Admin Dashboard) (Class Path: app/Http/Controllers/Admin/AdminDashboardController.php)

Consuming Payment Module Web Service

Another important example of service consumption is the retrieval of payment records from the Payment Module. This integration enables the Admin Dashboard to display the number of pending and overdue payments without duplicating financial data.

The Admin Dashboard invokes the Payment Module's REST API using internal service authentication headers. The request includes filter parameters and a mandatory timestamp, and the returned payment data is processed to compute the number of overdue payments for the current month. This allows administrators to monitor financial risks in real time.

Consumed Web Service Details

- Service Provider Module: Payment Module
- Consuming Module: Admin Dashboard
- API Route: /api/payments/filter
- HTTP Method: GET
- Purpose: Retrieve filtered payment records for administrative reporting

```php
// Pending payments
$response = Http::withHeaders([
    'X-INTERNAL-KEY' => config( key: 'services.internal.key'),
    'X-INTERNAL-SECRET' => encrypt(config( key: 'services.internal.secret')),
])->get(
    url: config( key: 'services.finance.url') . '/api/payments/filter',
    ['status' => 'pending']
);

$pendingPayments = $response->successful()
    ? count( value: $response->json( key: 'data') ?? [])
    : 0;
```

Figure 6.3: Consuming Payment Module API at User Module(Admin Dashboard) (Class Path: app/Http/Controllers/Admin/AdminDashboardController.php)

7. Index

Figure No.	Title	View Path	Class Path
2.1.1	User Registration Form	resources/views/auth/register.blade.php	app/Http/Controllers/Auth/RegisteredUserController.php
2.1.2	Email Verification Form After User Registration	resources/views/auth/verify-email.blade.php	app/Http/Controllers/Auth/VerifyEmailController.php
2.1.3	Send Reset Password Link Form	resources/views/auth/forgot-password.blade.php	app/Http/Controllers/Auth/PasswordResetLinkController.php
2.1.4	Reset Password Form	resources/views/auth/reset-password.blade.php	app/Http/Controllers/Auth/PasswordResetLinkController.php
2.1.5	Login Form	resources/views/auth/login.blade.php	app/Http/Requests/Auth/LoginRequest.php
2.1.6	Member Profile Management	resources/views/profile/edit.blade.php	app/Http/Controllers/ProfileController.php
2.1.7	Application to Become Organizer Form	resources/views/member/apply-organizer.blade.php	app/Http/Controllers/Auth/OrganizerApplicationController.php
2.1.8	Resubmit Application to Become Organizer Form	resources/views/member/apply-organizer.blade.php	app/Http/Controllers/Auth/OrganizerApplicationController.php
2.1.9	Approved Member Able to Login as Organizer	resources/views/member/apply-organizer.blade.php	app/Http/Controllers/Auth/OrganizerApplicationController.php
2.1.10	Organizer Profile Management	resources/views/profile/edit.blade.php	app/Http/Controllers/ProfileController.php
2.1.11	Admin Dashboard	resources/views/admin/dashboard.blade.php	app/Http/Controllers/Admin/AdminDashboardController.php
2.1.12	View User List	resources/views/livewire/admin/admin-user-table.blade.php	app/Http/Controllers/Admin/AdminUserController.php
2.1.13	Review Organizer Application List	resources/views/livewire/admin/admin-organizer-approval-table.blade.php	app/Http/Controllers/Admin/AdminMembershipController.php

2.1.14	Review Organizer Application Details	resources/views/livewire/admin/pending-detail.blade.php	app/Services/UserService/OrganizerApprovalService.php
2.1.15	Super Admin Create User	resources/views/livewire/admin/create-user.blade.php	app/Http/Controllers/Admin/AdminUserController.php
2.1.16	Super Admin Edit User	resources/views/livewire/admin/edit-user.blade.php	app/Http/Controllers/Admin/AdminUserController.php
2.1.17	Super Admin Import Users	resources/views/livewire/admin/admin-user-table.blade.php	app/Http/Controllers/UserImportExportController.php
2.1.18	Super Admin Export Users	resources/views/livewire/admin/admin-user-table.blade.php	app/Http/Controllers/UserImportExportController.php
2.2.1	Membership Management for Admins	resources/views/livewire/admin/membership-table.blade.php	app/Http/Controllers/Admin/AdminMembershipController.php
2.2.2	Membership Details for Organizer	resources/views/organizer/membership/index.blade.php	app/Http/Controllers/Admin/AdminMembershipController.php
3.1	Entity Class Diagram of User and Membership Module	-	-
4.2.1	Class Diagram of User Role Factory	-	-
4.2.2	Class Diagram of User Import/Export Factory	-	-
5.1.1	Code Snippet: Password Controller	-	app/Http/Controllers/Auth/PasswordController.php
5.1.2	Code Snippet: Login Request Validation (1)	-	app/Http/Requests/Auth/LoginRequest.php
5.1.3	Code Snippet: Login Request Validation (2)	-	app/Http/Requests/Auth/LoginRequest.php
5.2.1	Code Snippet: Authenticated Session Controller (1)	-	app/Http/Controllers/Auth/AuthenticatedSessionController.php
5.2.2	Code Snippet: Authenticated Session Controller (2)	-	app/Http/Controllers/Auth/AuthenticatedSessionController.php

5.2.3	Code Snippet: Session Configuration	-	config/session.php
6.1	API Routes Provided by User Module	-	routes/api.php
6.2	Consuming Event Module API in Admin Dashboard	-	app/Http/Controllers/Admin/AdminDashboardController.php
6.3	Consuming Payment Module API in Admin Dashboard	-	app/Http/Controllers/Admin/AdminDashboardController.php

8. References

Visual Paradigm. (n.d.). *Factory method design pattern*. Retrieved December 20, 2025, from https://www.visual-paradigm.com/tutorials/factorymethod.jsp

OWASP Foundation. (2021). *OWASP secure coding practices quick reference guide (v2.1)*.Retrieved December 20, 2025, from https://owasp.org/www-project-secure-coding-practices/