# **CEBU INSTITUTE OF TECHNOLOGY UNIVERSITY**

COLLEGE OF COMPUTER STUDIES

**Software Design Description**

*for*

PortfoliX: Student Portfolio Tracker Using PowerApps

## **Signature**

## **Change History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Time | Description | Author |
| 0.1 | 22/03/2025 | 3:00 PM | Initial Draft | Sacamay, Hannah Grace |
| 0.1.1 | 22/03/2025 |  | Module 2 | Handayan,Amiel Kenth |
| 0.2.1 | 22/03/2025 | 11:00 PM | User Management Module: Login | Pepito, John Luis |
| 0.2.2 | 23/03/2025 |  | User Management Module: Signup | Pepito, John Luis |
| 0.3.1 | 27/03/2025 | 12:00 AM | Dashboard Module: Student | Pepito, John Luis |
| 0.3.2 | 27/03/2025 |  | Dashboard Module: Faculty | Pepito, John Luis |
| 0.3.3 | 27/03/2025 | 2:00 AM | Dashboard Module ER Diagram | Pepito, John Luis |

## **Preface**

PortfoliX redefines how CIT-U students document, validate, and share their academic journeys. By replacing fragmented tools with a secure, Microsoft 365-integrated platform, we empower learners to build professional portfolios, equip faculty to guide growth, and connect recruiters to emerging talent—all while reducing administrative overhead. This centralized approach aligns perfectly with the College of Computer Studies' mission to prepare students for professional success through technological excellence and validated skill development.

At the core of PortfoliX is the Microsoft Power Platform ecosystem, which transforms portfolio management from an individual burden into an institutional advantage. PowerApps delivers intuitive, role-optimized interfaces that adapt to students, faculty, and recruiters. SharePoint Lists provide structured data storage for projects, certifications, and achievements, ensuring consistency and searchability. Power Automate orchestrates critical workflows—from faculty validation of submissions to automated recruiter notifications—creating a seamless experience that respects institutional protocols while minimizing manual effort.

This Software Design Description serves as the definitive implementation guide for bringing PortfoliX to life. For technical teams, it offers detailed specifications for PowerApps configurations, SharePoint schema designs, and security models that balance accessibility with data protection. For administrators, it demonstrates how the system scales naturally within CIT-U's existing Microsoft cloud environment, requiring minimal infrastructure investment. For institutional leaders, it showcases how student career preparation, faculty assessment, and industry engagement converge in a single platform that enhances the university's reputation while streamlining operations. The following sections translate this vision into actionable architecture (Section 2) and detailed component specifications (Section 3), providing everything needed to implement a portfolio solution that grows with the institution and adapts to evolving educational needs.

## **Table of Contents**

**CEBU INSTITUTE OF TECHNOLOGY UNIVERSITY 1**

[**Signature 2**](#_ksvrucejeafa)

[**Change History 3**](#_94zl9bxoo1dg)

[**Preface 4**](#_741b18u795d0)

[**Table of Contents 6**](#_vthazy53yk8)

[**1. Introduction 7**](#_e9exluj7lerd)

[1.1. Purpose 8](#_n0rtxs8c7hmz)

[1.2. Scope 9](#_ad7r9fgo2dx4)

[1.3. Definition and Acronyms 12](#_cgb0bgcci8h)

[1.4. References 14](#_h121tzi4x3a)

[**2. Architectural Design 16**](#_4ol2x9xdd6pz)

[Architecture Overview 16](#_94b3f2qhkm1c)

[**3. Detailed Design 19**](#_ohuzmijv6n6q)

[Module 1: Portfolio Management 20](#_n4ef5lgj7eem)

[1.1 Create Portfolio 20](#_mj8jhgeb4lk8)

[1.2 Update Portfolio 23](#_f32k9g11yr2n)

[1.3 Upload Supporting Documents 26](#_q2mtiphzdne0)

[Module 2: User Management 29](#_9ny4azyg6i4w)

[2.1 User Registration & Login 29](#_zd5y5tfh7qzx)

[2.2 Manage User Profiles 36](#_u8m5nju7ujm7)

[Module 3: Dashboard 40](#_l3hl25volvyx)

[3.1 Student Dashboard 40](#_cytl353pdszq)

[3.2 Faculty Dashboard 46](#_8jvy3x8xra0n)

[Module 4: Shareable Links 50](#_w4naulh9ni4u)

[4.1 Generate Shareable Link 50](#_pkeae9b7hw66)

[4.2 View Portfolio via Shareable Link 56](#_hrmenj2x16oj)

# **Introduction**

In today’s rapidly evolving digital landscape, students face increasing challenges in effectively managing and presenting their academic achievements. Many rely on personal cloud storage, offline documents, and social media platforms, which lack a structured, professional approach to portfolio management. This fragmentation makes it difficult for students to showcase their skills, receive faculty feedback, and gain recognition from potential employers.

To address these issues, PortfoliX: Student Portfolio Tracker is designed as a centralized digital portfolio system that enables students to efficiently store, organize, and present their projects, certifications, and skills. Built on Microsoft PowerApps, PortfoliX integrates faculty validation, recruiter access, and analytics tracking to enhance student career readiness and streamline academic assessments.

This document outlines the design and architecture of PortfoliX, detailing how its core features—profile creation, project submission, faculty evaluation, recruiter access, and data-driven insights—work together to provide a seamless and structured portfolio management experience. By integrating secure authentication, automated data retrieval, and interactive dashboards, the system ensures efficiency, accessibility, and usability for students, faculty, and recruiters.

PortfoliX is more than just a storage system—it is a comprehensive academic and career development tool that enhances student engagement, simplifies faculty assessments, and streamlines recruiter evaluations.

# **1.1.** **Purpose**

The primary objectives of this Software Design Description (SDD) are to translate the requirements detailed in the Software Requirements Specification (SRS) into a comprehensive technical blueprint for PortfoliX, a centralized portfolio management system for the College of Computer Studies at Cebu Institute of Technology - University. Specifically, this SDD serves as a guide for implementing PortfoliX using Microsoft Power Platform technologies, detailing how Microsoft PowerApps provides both frontend interfaces and backend business logic, while SharePoint Lists deliver structured data storage. This low-code approach accelerates development while ensuring the scalability and security needed for institutional requirements.

The SDD bridges the gap between high-level requirements and practical implementation by documenting the three-tier architecture (User → PowerApps → SharePoint) illustrated in Section 2, and providing detailed modules specifications for Portfolio Management, User Management, Dashboards, and Shareable Links. It serves as a developer guide for configuring PowerApps formulas, designing SharePoint list schemas, and implementing custom logic for features such as Faculty Validation (Module 3.2), Recruiter Access via secure links (Module 4.2), and Real-Time Dashboards (Module 3.1). By providing this technical foundation, the SDD helps transform CIT-U's fragmented portfolio management approach (relying on personal cloud storage, social media, and offline documents) into a unified, validated portfolio hub where students can showcase their achievements, faculty can provide structured assessment, and recruiters can access verified student accomplishments.

This document is essential for both low-code configuration (PowerApps UI components, SharePoint columns, Power Automate workflows) and custom development tasks (OAuth 2.0 implementation, JWT validation). It establishes clear maintenance procedures for SharePoint permission models and data governance, ensuring the system's long-term sustainability. By aligning technical design decisions with CCS's portfolio tracking needs, this SDD provides traceability between architectural diagrams (Figures 1.1-1.3) and module implementations, setting clear expectations for developers building a scalable, secure, and user-friendly platform that enhances student career readiness and streamlines academic assessments.

# **1.2.** **Scope**

This document defines the design and architectural scope of PortfoliX, a digital portfolio management system developed using Microsoft Power Platform technologies. The system is intended for use by students and faculty of the College of Computer Studies at Cebu Institute of Technology – University. It enables students to build and manage structured digital portfolios, and allows faculty to assess, monitor, and generate insights from student submissions.

The design scope includes all software components and architectural layers necessary for implementing the functional and non-functional requirements as outlined in the Software Requirements Specification (SRS). The system follows a three-tier architecture:

1. User Layer: Microsoft PowerApps frontend interfaces
2. Processing Layer: Microsoft PowerApps business logic
3. Data Layer: Microsoft SharePoint Lists for data storage

This Software Design Description covers the following functional areas:

1. **Portfolio Management System** (Section 3.1)

* PowerApps interfaces for creating, updating, and managing student portfolios.
* Support for uploading micro-credentials, projects, and supporting documents.
* Integration with SharePoint Lists (Portfolio, Projects, Certifications, Documents) as shown in Fig. 1.2.
* PowerApps connectors for seamless data operations between frontend and SharePoint.

1. **User Management** (Section 3.2)

* Single Sign-On (SSO) authentication using institutional email accounts
* Role-based access control to distinguish between student and faculty users
* User profile management with customizable settings
* Account validation to ensure only authorized CIT-U CCS students and faculty can access the system

1. **Dashboard Modules** (Section 3.3)

* Student Dashboard with real-time data binding to SharePoint Lists
* Faculty Dashboard for viewing, filtering, and evaluating student portfolios
* Interactive data visualization components powered by PowerApps
* Aggregated portfolio views with insights into student achievements and competencies

1. **Shareable Links Module** (Section 3.4)

* Generation of secure, time-limited portfolio access links
* Customizable link settings including expiration dates and password protection
* SharePoint "Shareable Links List" for tracking shared portfolio information
* Public portfolio viewing interface for recruiters and external stakeholders

Non-functional aspects addressed in the design include:

* **Scalability**: Leveraging SharePoint Online's cloud storage capabilities and PowerApps' low-code scalability to accommodate growing numbers of students and portfolio submissions
* **Security**: Implementation of OAuth 2.0 authentication, role-based access control, and SharePoint's built-in permission management
* **Usability**: Intuitive, drag-and-drop PowerApps UI components
* **Performance**: Efficient data retrieval and rendering through optimized PowerApps formulas and SharePoint data connections
* **Maintainability**: Low-code architecture allows for rapid updates and enhancements with minimal development overhead

The scope explicitly excludes:

* Integration with external job boards or recruitment platforms
* Advanced analytics requiring dedicated data warehousing solutions
* Mobile application development beyond PowerApps' responsive design capabilities
* Portfolio content creation tools beyond basic text and media uploading

This scope defines the system design boundaries and ensures alignment with institutional requirements, usability goals, and technical constraints, forming the foundation for the implementation of a secure, scalable, and maintainable digital portfolio platform.

# **1.3.** **Definition and Acronyms**

* **PortfoliX:** A student portfolio management system designed to help students store, organize, and present their academic and professional achievements in a structured and validated manner.
* **CIT-U (Cebu Institute of Technology - University):** The institution where PortfoliX will be implemented, specifically within its College of Computer Studies (CCS) to support students, faculty, and recruiters.
* **Microsoft PowerApps:** A low-code development platform used to build PortfoliX, allowing for easy customization and seamless integration with other Microsoft services.
* **OAuth 2.0 / Single Sign-On (SSO)**: A secure authentication method allowing students, faculty, and recruiters to log in using their institutional Microsoft Teams credentials.
* **Faculty Validation:** A process where faculty members review, approve, and provide feedback on student submissions before they are officially added to their portfolios.
* **Recruiter Access:** A feature that allows recruiters to search, filter, and view student portfolios, streamlining the talent acquisition process by assessing student skills, certifications, and project history.
* **Analytics Dashboard:** A data visualization tool that provides insights into student progress, faculty engagement, and recruiter interactions, helping users make informed decisions.
* **Project and Certification Submission:** A core feature enabling students to upload projects, research work, and certifications into categorized sections for structured portfolio management.
* **Public Portfolio Sharing:** A functionality that allows students to share their portfolios with recruiters and industry partners, with customizable privacy settings for controlled access.
* **Role-Based Access Control (RBAC):** A security framework that defines different access levels for students, faculty, and recruiters, ensuring appropriate permissions for each user role.
* **Stakeholders:** Individuals or groups who have an interest in the development and implementation of PortfoliX. This includes students, faculty, recruiters, and developers.

# **1.4.** **References**

[1] D. Megan, “College Students’ Motivation and Confidence for ePortfolio Use.,”

International Journal of ePortfolio, vol. 9, no. 1, pp. 1–16, 2019, Available:

<https://eric.ed.gov/?id=EJ1214509>

[2] M. Ciesielkiewicz, “The use of e-portfolios in higher education: From the students’

perspective,” Issues in Educational Research, vol. 29, no. 3, p. 2019, 2019, Available:

Project Proposal 18

https://digiuv.villanueva.edu/bitstream/handle/20.500.12766/483/The%20use%20of%20

e-portfolios%20in%20higher%20education.pdf?sequence=1

[3] Z. Syzdykova, K. Koblandin, N. Mikhaylova, and O. Akinina, “Assessment of

E-Portfolio in Higher Education,” International Journal of Emerging Technologies in

Learning (iJET), vol. 16, no. 02, p. 120, Jan. 2021, doi:

<https://doi.org/10.3991/ijet.v16i02.18819>.

[4] H. Yang and R. Wong, “An in-depth literature review of e-portfolio implementation in

higher education: Processes, barriers, and strategies,” Issues and Trends in Learning

Technologies, vol. 12, no. 1, Apr. 2024, doi: https://doi.org/10.2458/itlt.5809.

[5] M. M. Alajmi, “The impact of e-portfolio use on the development of professional

standards and life skills of students: a case study,” Entrepreneurship and Sustainability

Issues, vol. 6, no. 4, pp. 1714–1735, Jun. 2019, doi:

<https://doi.org/10.9770/jesi.2019.6.4(12)>.

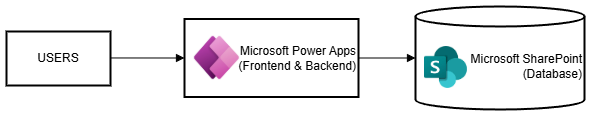
[6] O. Farrell, “From Portafoglio to Eportfolio: The Evolution of Portfolio in Higher

Education,” Journal of Interactive Media in Education, vol. 2020, no. 1, 2020, doi:

https://doi.org/10.5334/jime.574.

# **Architectural Design**

## **Architecture Overview**

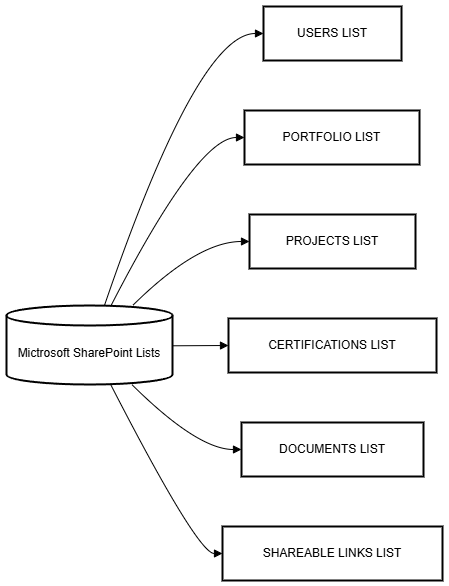
The PortfoliX system uses a three-tier architecture built primarily on Microsoft's ecosystem. From the diagrams, we can see:  


*PortfolioX Architectural Diagram 1.1*

1. User-facing layer: Users interact with the Microsoft PowerApps frontend.
2. Processing layer: Microsoft PowerApps handles both frontend UI and backend business logic.
3. Data storage layer: Microsoft SharePoint serves as the database.

This architecture follows a client-server model where PowerApps serves as both the client interface and application server, while SharePoint acts as the database server.

**Microsoft SharePoint Lists**



*PortfolioX Architectural Diagram 1.2*

The diagram shows that SharePoint serves as the database backbone for PortfoliX with several structured lists:

1. Users List - Stores user account information
2. Portfolio List - Contains portfolio metadata
3. Projects List - Stores project information
4. Certifications List - Records certification data
5. Documents List - Stores supporting documents
6. Shareable Links List - Manages portfolio sharing

SharePoint provides a ready-made data storage solution with built-in security, version control, and document management capabilities.

**Microsoft PowerApps Components**

## 

*PortfolioX Architectural Diagram 1.3*

Microsoft PowerApps serves multiple functions:

1. UI Components: Provides screens and forms for users to interact with the system
2. Login Screen - Handles authentication
3. Dashboard - Displays summary information
4. Portfolio Management - Interface for managing portfolios
5. Link Sharing - Controls portfolio sharing functionality
6. Business Logic: Contains formulas and functions that implement business rules
7. Data validation
8. Processing logic
9. Workflow execution
10. SharePoint Connector: Serves as the bridge between the UI and database
11. Handles database operations
12. Manages data retrieval and persistence

**Component Interactions**

The flow of data through the system appears to follow this pattern:

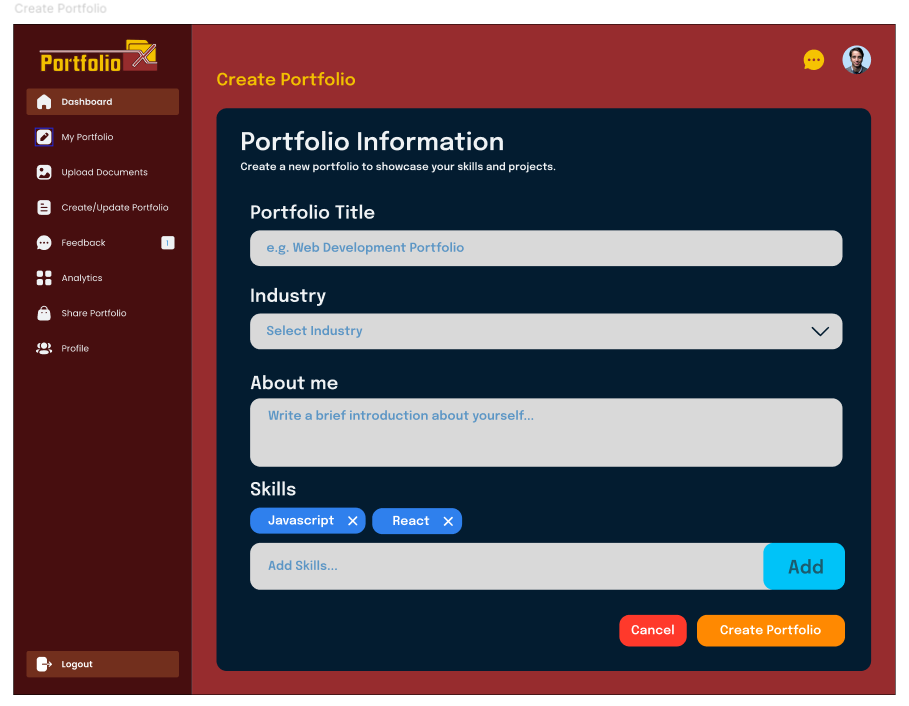
1. Users interact with the PowerApps frontend
2. The UI components collect user input
3. Business logic components process that input and determine necessary actions
4. The SharePoint connector executes database operations on the relevant SharePoint lists
5. Results are returned through the connector to the business logic
6. The UI is updated to reflect changes or display requested information

# **Detailed Design**

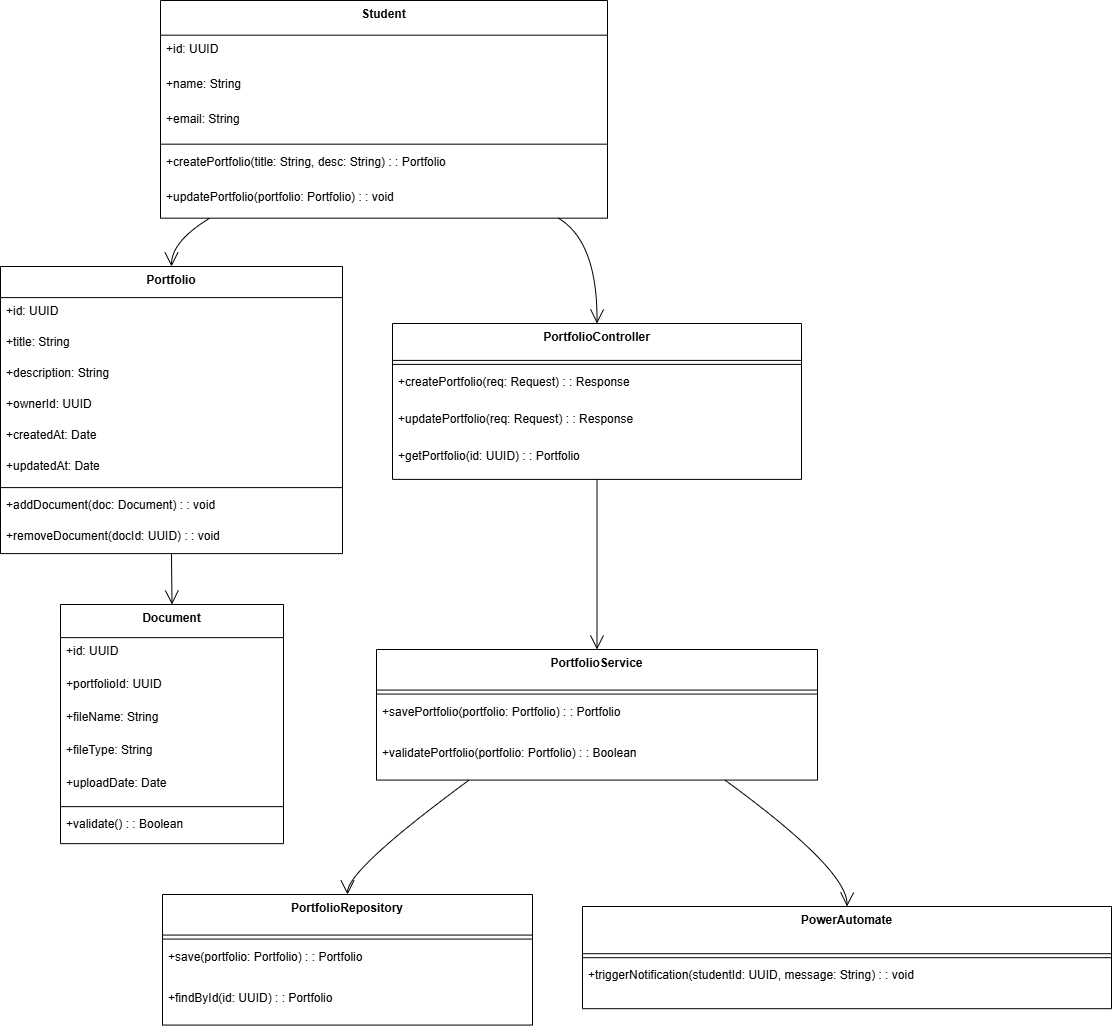
## ***Module 1: Portfolio Management***

### ***1.1 Create Portfolio***

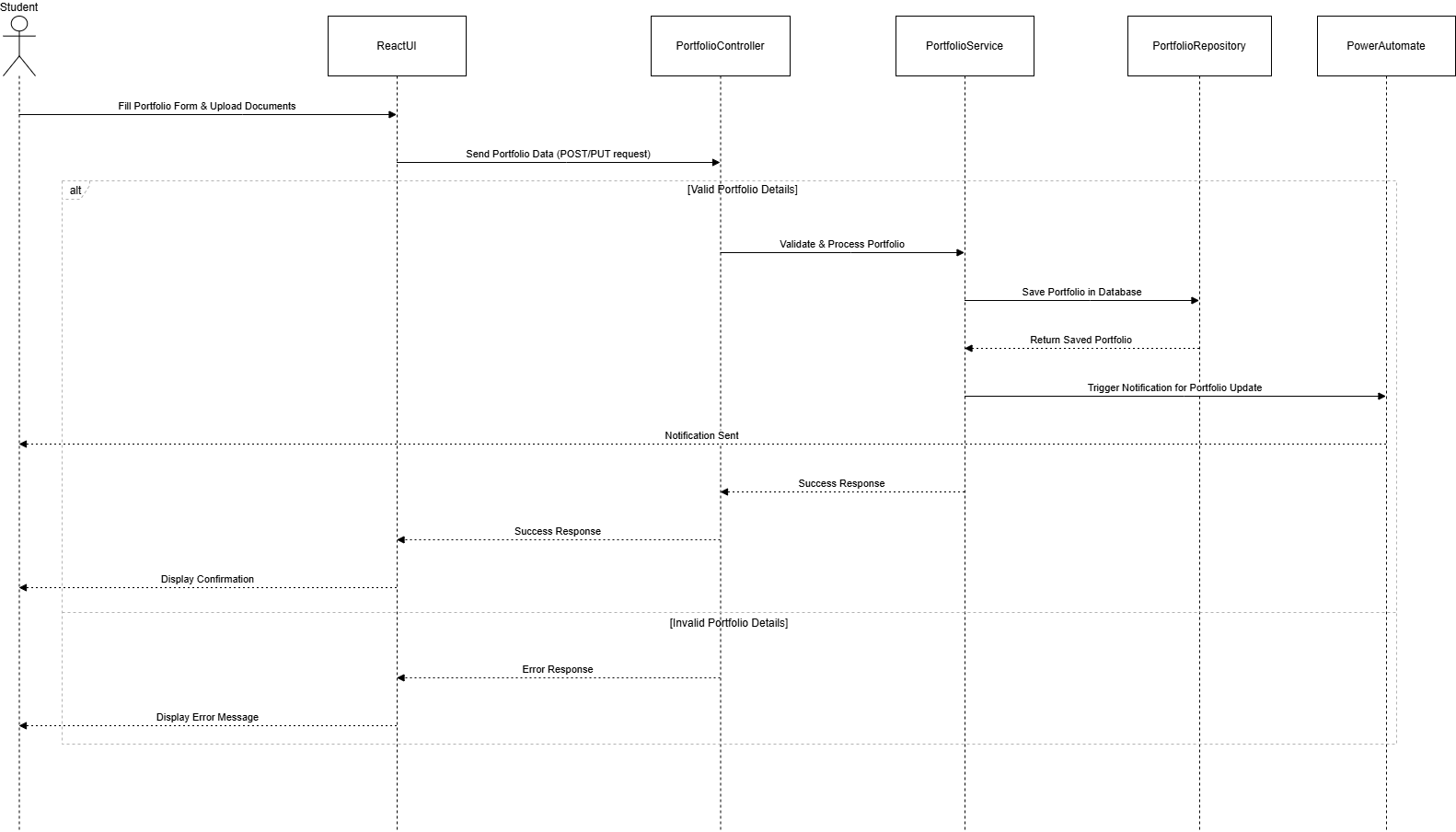
* User Interface Design



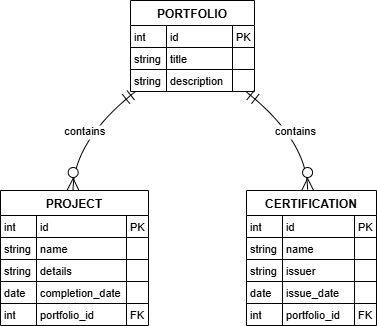
* Front-end component(s)
  + Component Name
    - Description and purpose
    - Component type or format
* Back-end component(s)
  + Component Name
    - Description and purpose
    - Component type or format
* Object-Oriented Components
  + Class Diagram

****

* + Sequence Diagram

****

* Data Design
  + ERD or schema

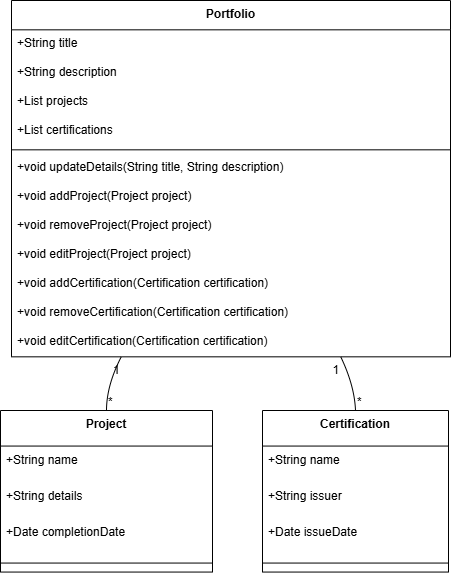


### ***1.2 Update Portfolio***

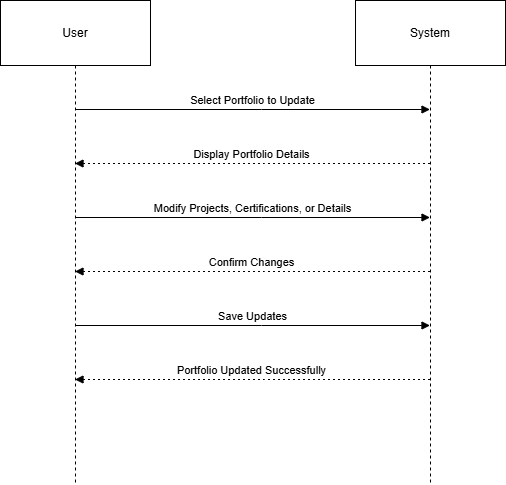
* User Interface Design



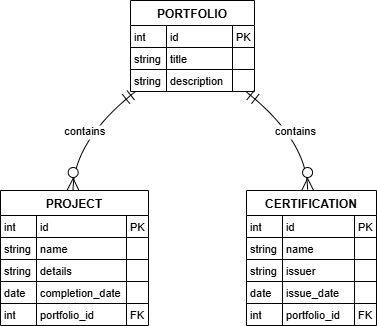
* Front-end component(s)
  + Component Name
    - Description and purpose
    - Component type or format
* Back-end component(s)
  + Component Name
    - Description and purpose
    - Component type or format
* Object-Oriented Components
  + Class Diagram



* + Sequence Diagram

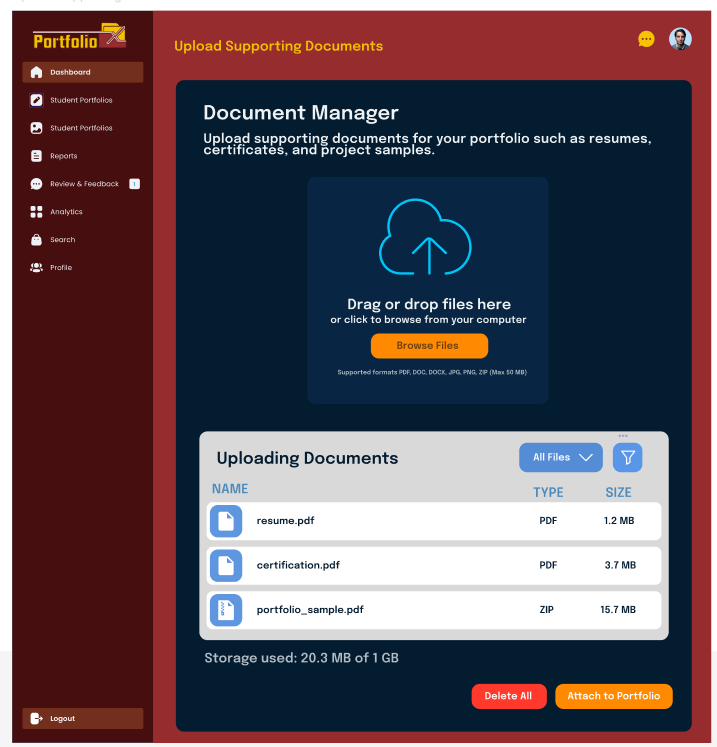


* Data Design
  + ERD or schema

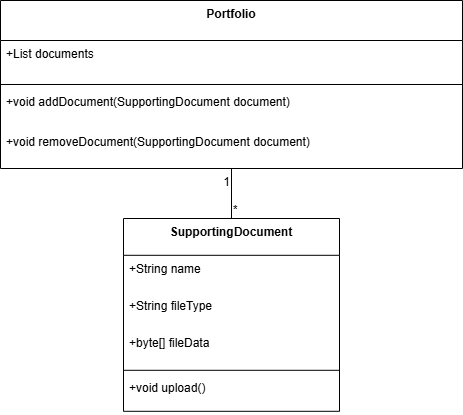


### ***1.3 Upload Supporting Documents***

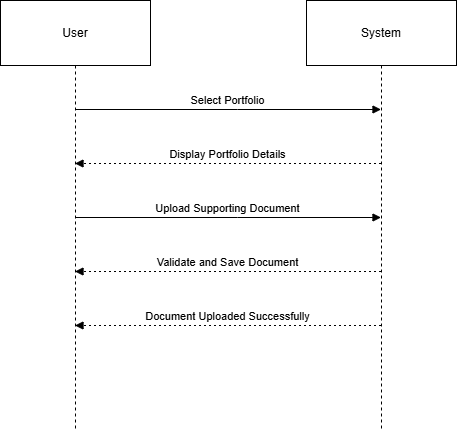
* User Interface Design



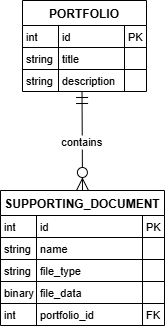
* Front-end component(s)
  + Component Name
    - Description and purpose
    - Component type or format
* Back-end component(s)
  + Component Name
    - Description and purpose
    - Component type or format
* Object-Oriented Components
  + Class Diagram



* + Sequence Diagram



* Data Design
  + ERD or schema



## ***Module 2: User Management***

**2.1 User Registration & Login**

This document outlines the updated design of the User Management module, replacing Microsoft Single Sign-On (SSO) with GitHub OAuth for user authentication while retaining the existing registration and error handling components.

## **User Interface Design**

### **Front-end Components**

#### **Login Page**

* **Purpose**: Facilitates user authentication via GitHub OAuth and redirects users to their respective dashboards (Student or Faculty) upon successful login.
* **Features**:
  + Displays a "Login with GitHub" button, which initiates the GitHub OAuth flow.
  + Redirects users to GitHub’s authorization page for authentication.
  + Upon successful authentication, retrieves user details (e.g., GitHub username, email) and directs users to the appropriate dashboard based on their role.
  + Handles authentication failures by displaying an error message and redirecting to the Error Page.
* **Technology**: Built with React.js, using Axios for HTTP requests and React Router for navigation.

#### **RegistrationForm**

* **Purpose**: Provides an interface for new users (students and faculty) to sign up.
* **Features**:
  + Includes input fields for name, institutional email, password, role selection (Student/Faculty), and department.
  + Submits data to the backend for validation and storage.
  + Displays validation errors (e.g., invalid email format, duplicate email).
* **Status**: Unchanged from the original design.
* **Technology**: Built with React.js, using form validation libraries like Formik.

#### **Error Page**

* **Purpose**: Displays specific error messages during login or registration failures.
* **Features**:
  + Shows user-friendly messages for issues like failed GitHub authentication (e.g., "Authentication failed. Please try again.") or network errors.
  + Provides a link to return to the Login Page.
* **Status**: Unchanged but updated to handle GitHub-specific error messages.
* **Technology**: Built with React.js.

## **Back-end Components**

### **Authentication Controller**

* **Purpose**: Manages user authentication by processing GitHub OAuth login requests.
* **Functionality**:
  + Initiates the OAuth flow by redirecting users to GitHub’s authorization endpoint.
  + Handles the callback from GitHub, validating the authorization code and exchanging it for an access token.
  + Retrieves user details (e.g., username, email) from GitHub’s API.
  + Authenticates the user using Spring Security and assigns a role (default: Student, or Faculty if specified).
  + Redirects to the appropriate dashboard or the Error Page on failure.
* **Technology**:
  + Implemented using Java with the Spring Boot framework.
  + Leverages Spring Security’s OAuth2 client support for GitHub integration.
  + Configured via application.yml with GitHub OAuth client ID, client secret, and scopes (user:email).
* **Change**: Replaced Microsoft Entra ID (Azure AD) integration with GitHub OAuth, updating dependencies and configuration.

### **Error Handler**

* **Purpose**: Monitors, logs, and manages errors during authentication.
* **Functionality**:
  + Captures GitHub OAuth-specific errors (e.g., invalid tokens, API rate limits).
  + Logs detailed error information for debugging.
  + Returns user-friendly error messages to the front-end Error Page.
* **Status**: Updated to handle GitHub OAuth error codes and messages.
* **Technology**: Implemented using Spring Boot’s exception handling mechanisms.

### **UserRegistrationController**

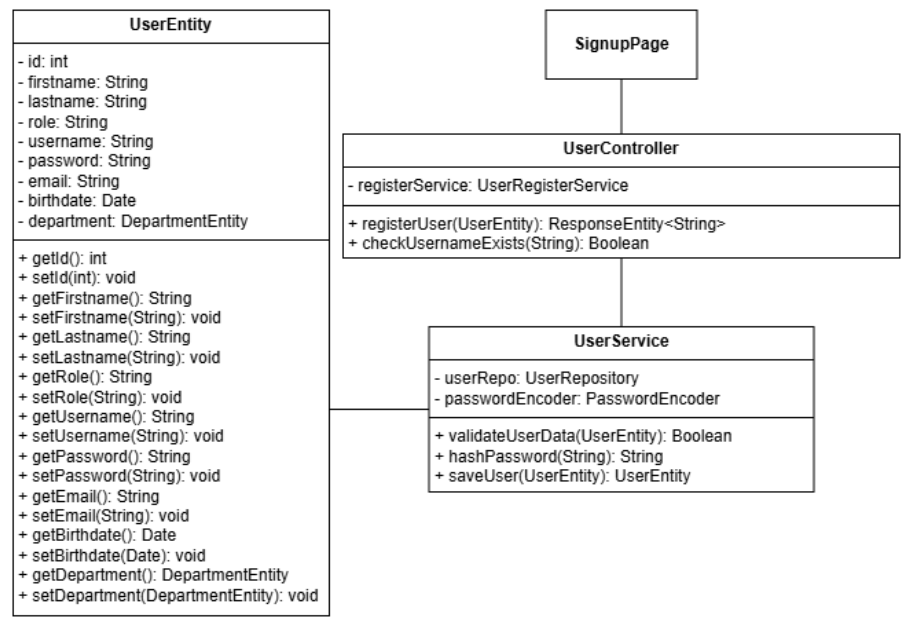
* **Purpose**: Handles HTTP requests for registering new users (POST /register).
* **Functionality**:
  + Receives user data from the RegistrationForm.
  + Calls UserRegistrationService to validate and store user details.
  + Returns success responses or error messages (e.g., "Email already exists").
* **Status**: Unchanged from the original design.
* **Technology**: Built with Spring Boot, using RESTful API endpoints.

### **UserRegistrationService**

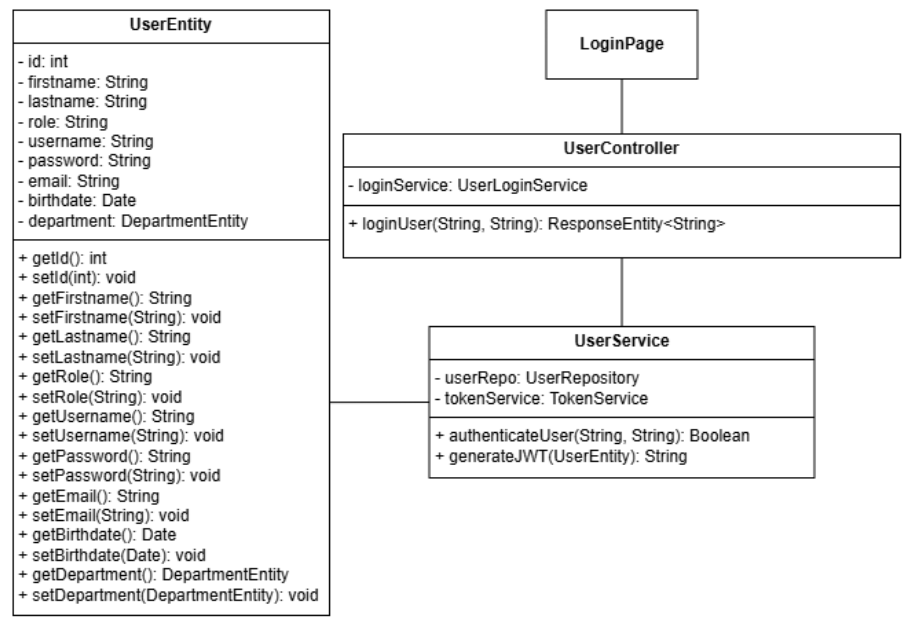
* **Purpose**: Validates and processes user registration data.
* **Functionality**:
  + Validates input data (e.g., email format, password strength).
  + Checks for duplicate email addresses.
  + Hashes passwords using a secure algorithm (e.g., BCrypt).
  + Calls UserRepository to store the user in the database.
* **Status**: Unchanged from the original design.
* **Technology**: Implemented as a Spring Service component.

### **UserRepository**

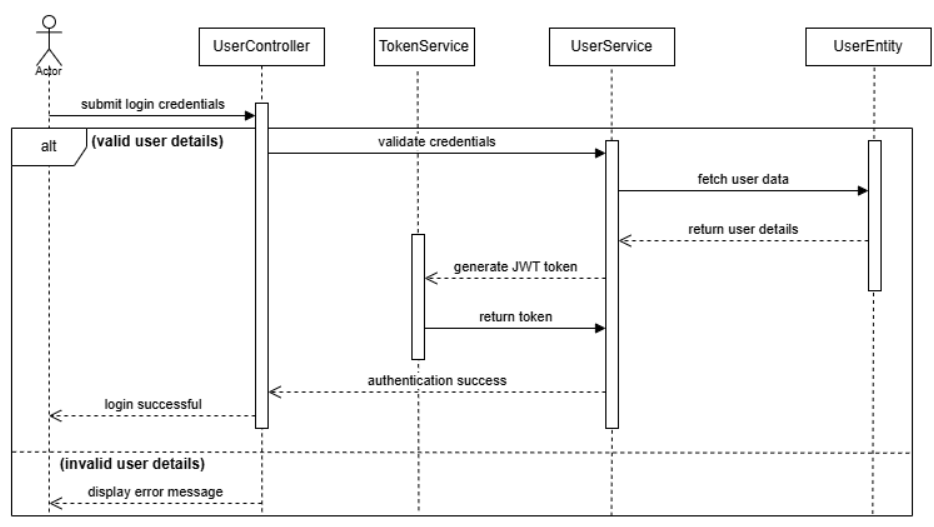
* **Purpose**: Interfaces with the database for user creation and retrieval.
* **Functionality**:
  + Stores user details (name, email, hashed password, role, department).
  + Ensures email uniqueness through database constraints.
  + Retrieves user data for authentication and profile management.
* **Status**: Unchanged from the original design.
* **Technology**: Uses Spring Data JPA with a relational database (e.g., MySQL, PostgreSQL).
* Object-Oriented Components
  + Class Diagram
  + Signup Page



* Login Page



* + Sequence Diagram
  + Login



* + Signup

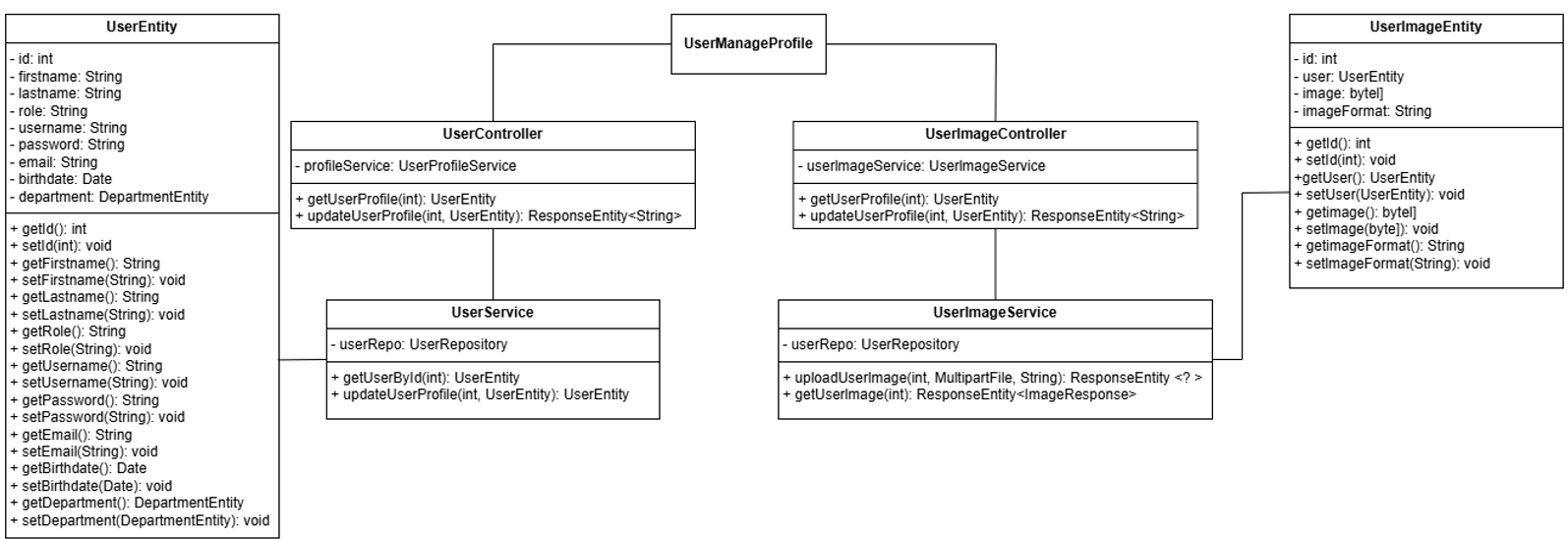


* Data Design
  + ERD or schema

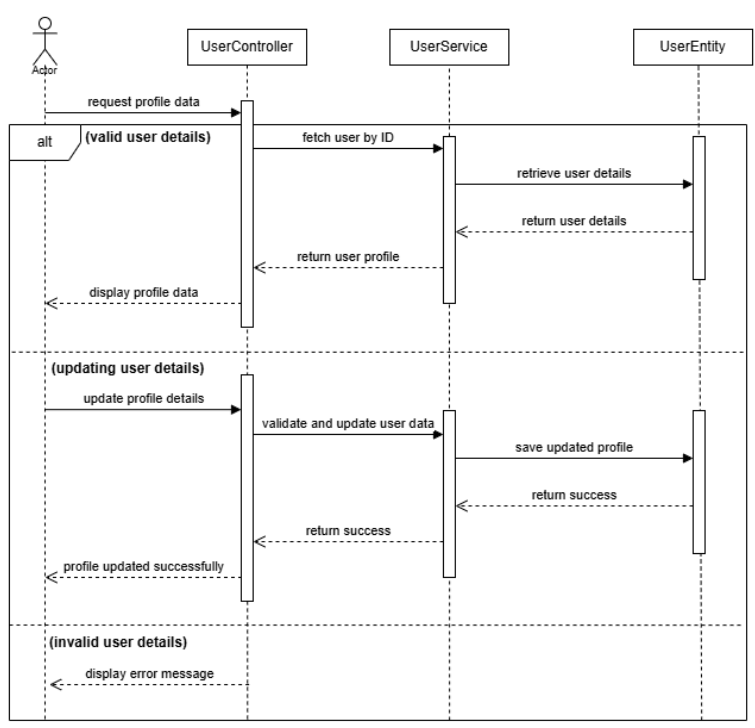


### ***2.2 Manage User Profiles***

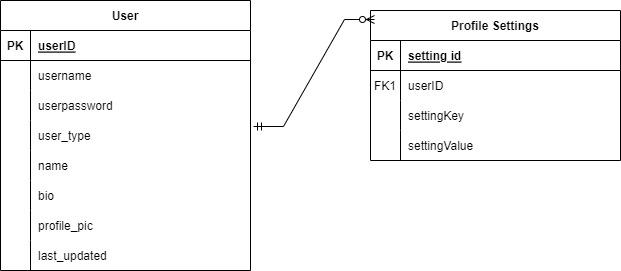
* User Interface Design
* Front-end component(s)
  + UserProfilePage
    - Allows students and faculty to update profile details.
    - Fields: name, bio, profile picture, skills, department.
    - Drag and drop components built-in inside the microsoft powerapp.
* Back-end component(s)
  + UserProfileController
    - Allows students and faculty to update profile details.
    - Fields: name, bio, profile picture, skills, department.
  + UserProfileService
    - Validates and updates user details.
    - Calls UserRepository to store updated information.
  + UserProfileService
    - Stores and retrieves user profile data.
* Object-Oriented Components
  + Class Diagram



* + Sequence Diagram



* Data Design
  + ERD or schema



## ***Module 3: Dashboard***

### ***3.1 Student Dashboard***

* + - User Interface Design



* + - Front-end component(s)
      * StudentDashboardPage
        + Displays student progress, achievements, micro-credentials, and uploaded portfolios.
        + Provides filtering options for achievements, projects, and skills.
        + Uses React and Material-UI for an interactive and responsive UI.
        + Description and purpose
        + Component type or format
* PortfolioList
  + - * + Displays a list of student portfolios with project details and certifications.
        + Uses Material-UI List and Card components for better UI structure.
* ProgressChart
  + - * + Visual representation of student progress based on achievements and completed projects.
        + Uses Chart.js with Material-UI Card for visualization.
* RecentActivity
  + - * + Displays the latest activities like faculty feedback and recent project updates.
        + Uses Material-UI Table for structured display.
    - Back-end component(s)

StudentDashboardController (GET /student/dashboard/{id})

Retrieves student portfolio data including projects, certifications, and progress.

Calls StudentDashboardService to process and structure data.

* + - * StudentDashboardService

Displays a list of student portfolios with project details and certifications.

Uses Material-UI List and Card components for better UI structure.

* + - * ProgressChart

Fetches student achievements, uploaded projects, and skills.

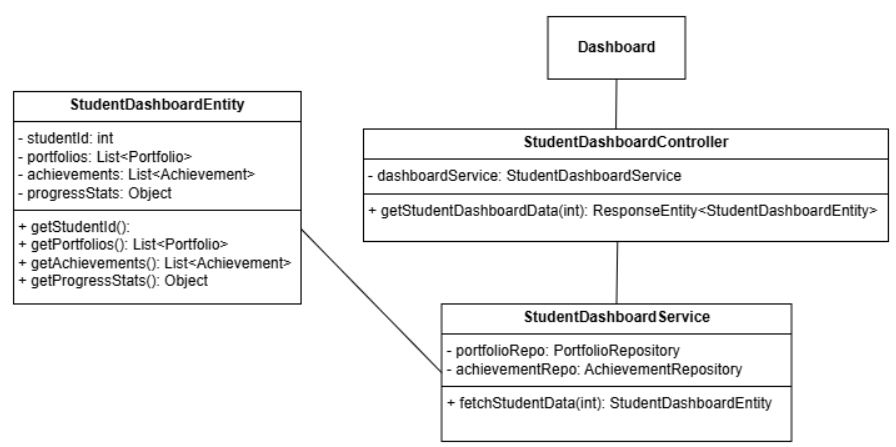
Computes progress statistics for visualization.

Returns structured dashboard data.

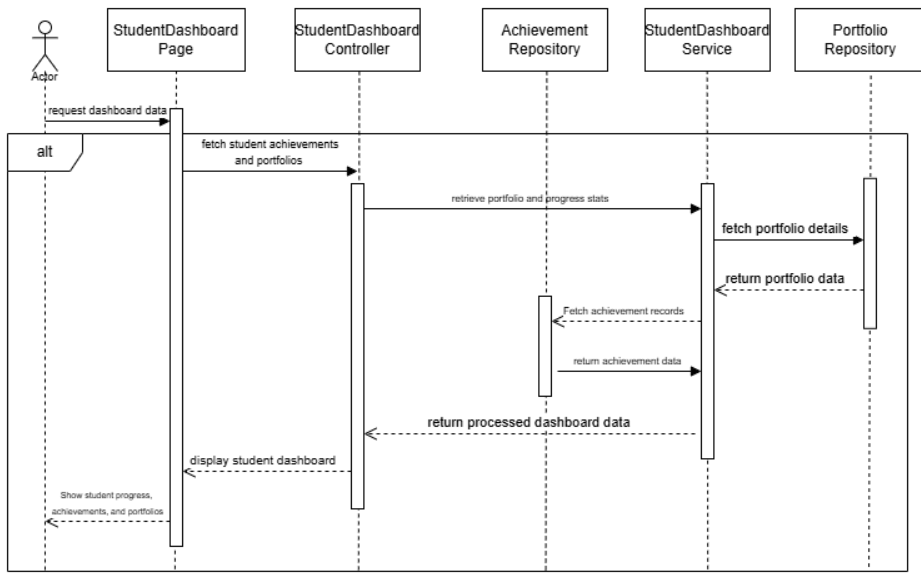
* + - * PortfolioRepository

Retrieves student portfolios stored in the database.

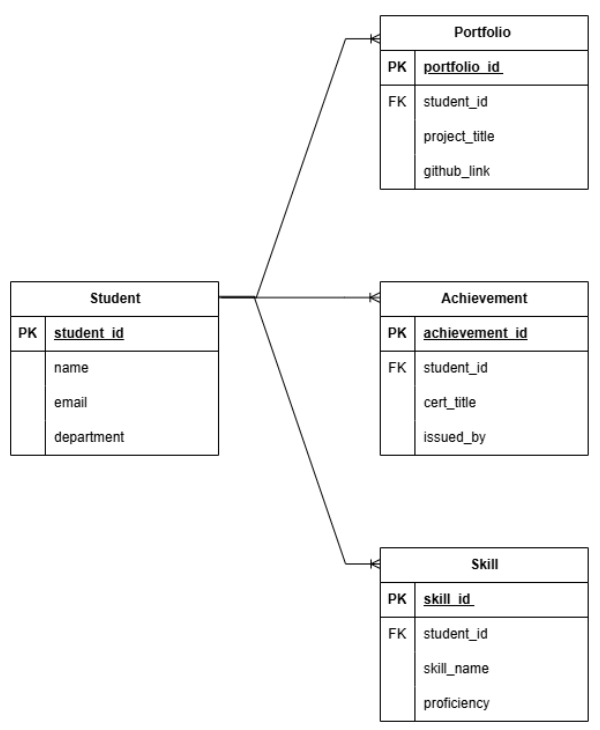
* AchievementRepository
  + - * + Fetches student achievement data and certificates.
    - Object-Oriented Components
      * Class Diagram



* + - * Sequence Diagram



* Data Design
  + ERD or schema



### ***3.2 Faculty Dashboard***

* + - User Interface Design



* + - Front-end component(s)

FacultyDashboardPage

Displays summarized student progress reports and portfolio insights.  
Uses React and Material-UI for a visually structured UI.

* + - * StudentList

Displays a list of students with portfolio summaries.

Allows faculty to filter students based on skills, projects, and department.

* + - * AnalyticsSection

Provides graphical insights on student achievements and competency levels.

Uses Chart.js for performance visualization.

* + - * FeedbackPanel

Allows faculty to provide feedback on student portfolios.

Uses Material-UI Dialog for structured feedback submission.

* + - Back-end component(s)

FacultyDashboardController (GET /faculty/dashboard)

Retrieves student data for faculty reports and insights.

Calls FacultyDashboardService to structure and analyze student performance.

* + - * FacultyDashboardService

Aggregates student portfolio data and achievements.

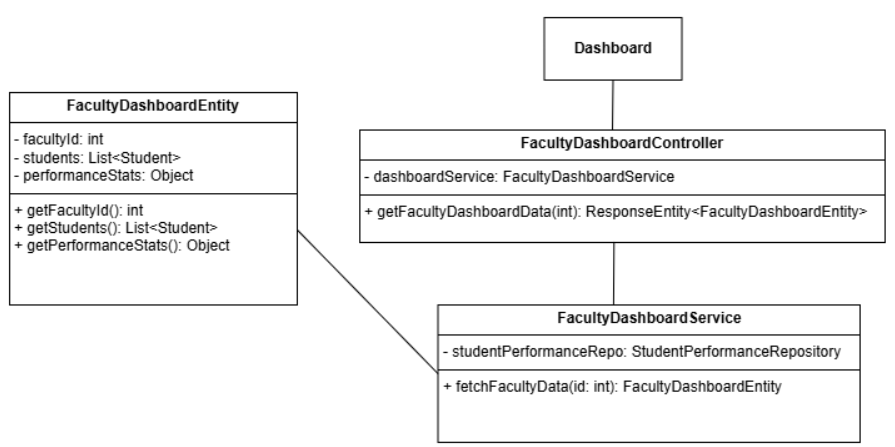
Generates summary statistics for faculty evaluation.

Allows filtering based on student skills, projects, and grades.

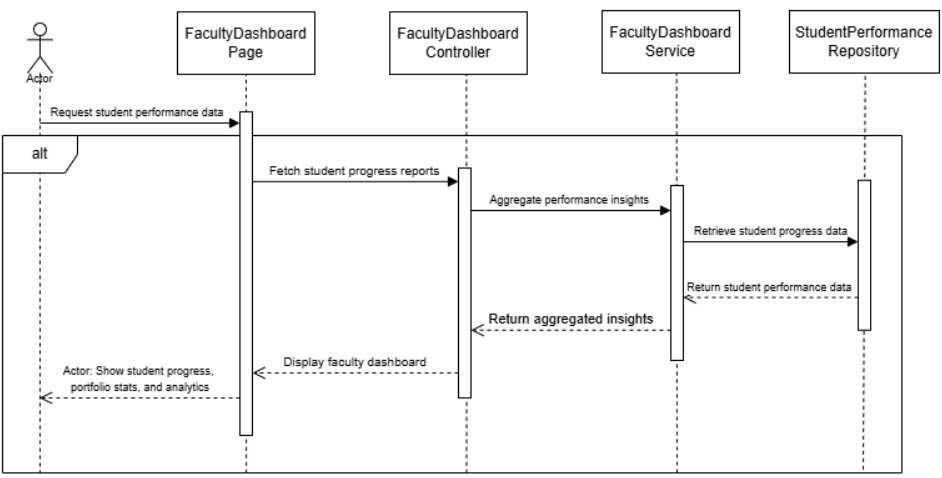
* + - * StudentPerformanceRepository

Fetches student progress metrics from the database.

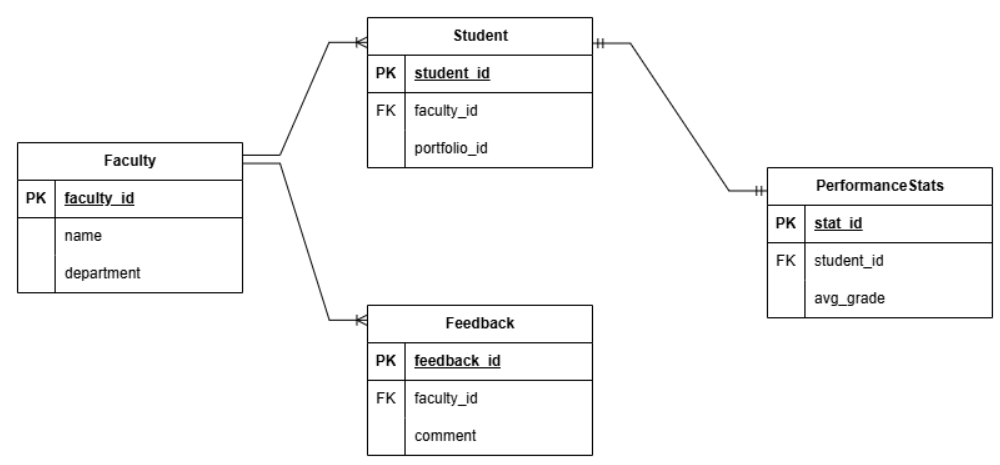
* + - Object-Oriented Components
      * Class Diagram



* + - * Sequence Diagram



* + - Data Design
      * ERD or schema

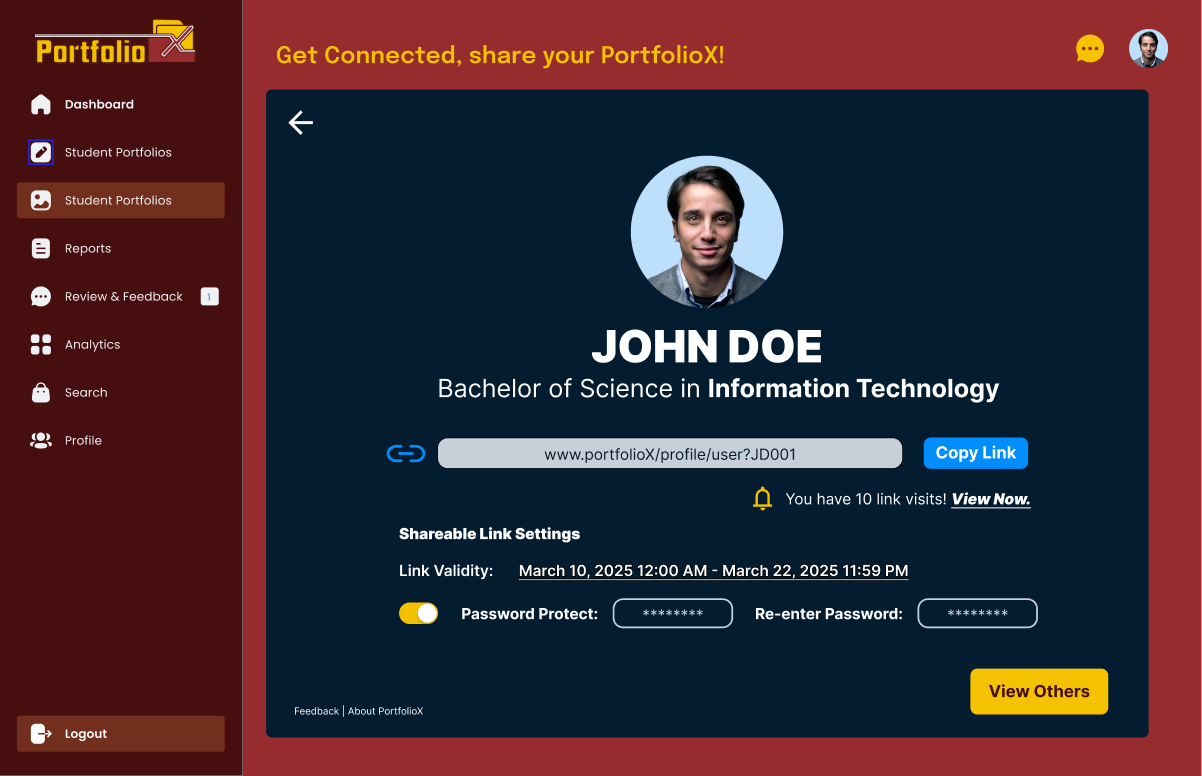


### 

## ***Module 4: Shareable Links***

### ***4.1 Generate Shareable Link***

* + - User Interface Design



* + - Front-end component(s)
      * SharePortfolioModal
        + This component is a modal dialog that allows users to share their portfolio with others. It provides functionality to:

Add people and colleagues (likely by email or username).

Display a shareable link with a "Copy Link" button.

Set a custom date range for link access.

Enable password protection for the shared link.

Close the modal with an "X" button.

Submit the sharing settings with a "Done" button.

* + - * + Component type or format

React functional component (using hooks).

Uses Formik for form handling and validation.

Styled with CSS (potentially using a library like Styled Components or Material UI for better styling).

* + - * + Additional Front-end Components (Potentially):

**DatePicker:** A reusable date picker component (e.g., from a library like react-datepicker) for selecting the start and end dates of the link access range.

**ToggleSwitch:** A reusable toggle switch component for enabling/disabling password protection.

**InputField:** A reusable input field component for adding people and colleagues.

**Button:** A reusable button component for "Copy Link" and "Done."

* + - Back-end component(s)
      * PortfolioShareController
        + This is a REST controller that handles requests related to sharing portfolios. It exposes endpoints for:

Generating a shareable link for a portfolio.

Setting access range and password protection for the link.

Adding people/colleagues to the shared portfolio.

Retrieving shared portfolio information.

* + - * + Component type or format

Spring Boot REST controller (Java class annotated with @RestController).

Uses @RequestMapping to map HTTP requests to methods.

Returns JSON responses.

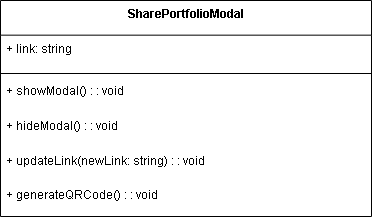
* + - * + Additional Back-end Components:

**PortfolioShareService:** A service layer that encapsulates the business logic for sharing portfolios.

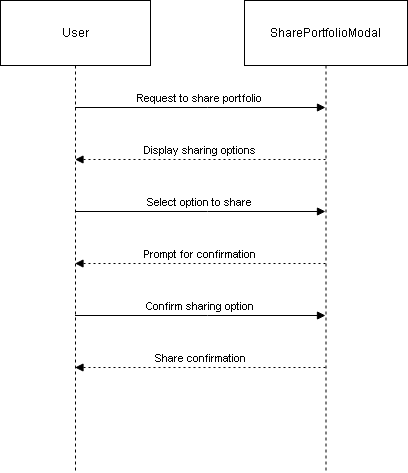
**PortfolioShareRepository:** A repository layer that interacts with the database to store and retrieve portfolio sharing data.

**PortfolioShare Entity:** A JPA entity representing the portfolio sharing information (link, access range, password, etc.).

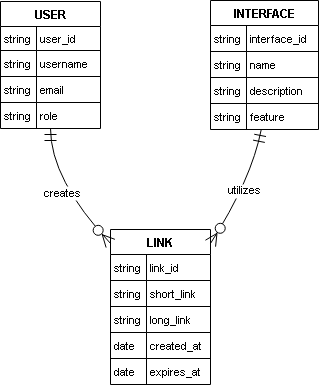
* + - Object-Oriented Components
      * Class Diagram:



* + - * Sequence Diagram:

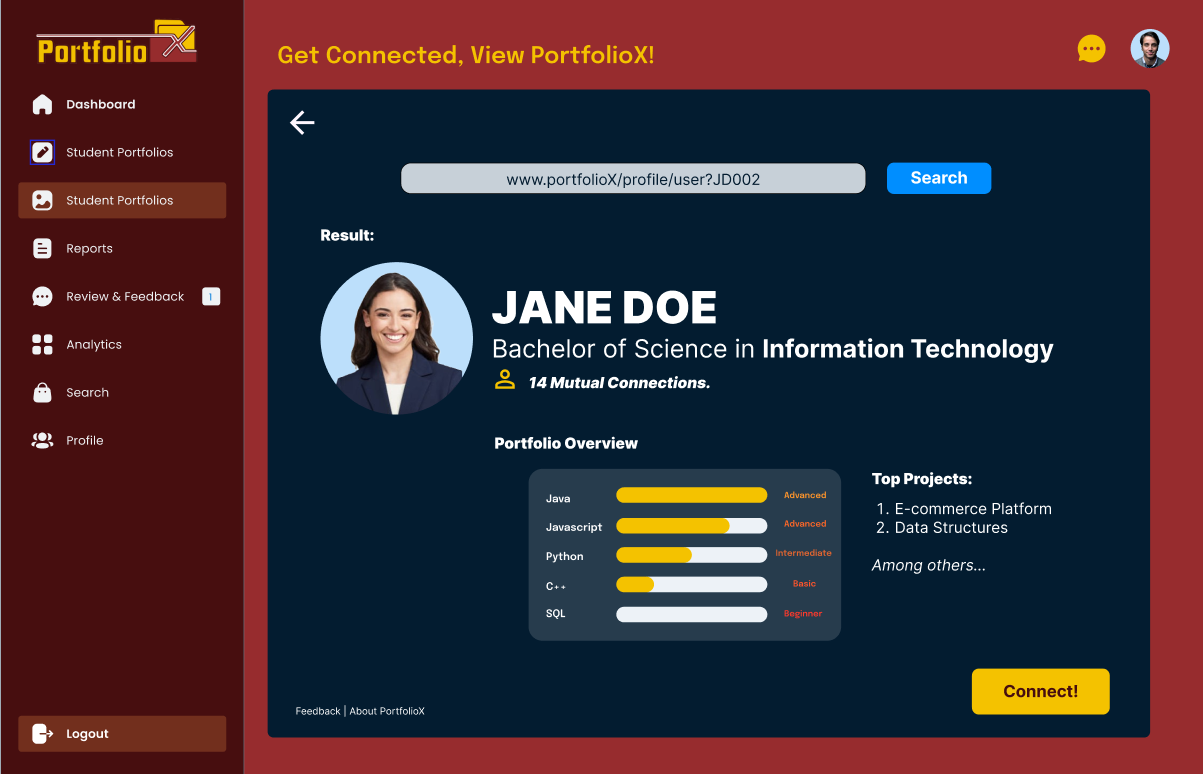


* + - Data Design
      * ERD or schema



### ***4.2 View Portfolio via Shareable Link***

* + - User Interface Design



* + - Front-end component(s)
      * VisitPortfolioModal
        + Description and purpose

This modal component allows a user to visit a portfolio that is protected by a password. It displays:

The user's avatar or profile picture.

The user's name (e.g., "Ricardo Dangal-Dalisay").

A password input field labeled "Link Password."

A "Visit" button to submit the password and access the portfolio.

Links for "Feedback," "About PortfoliX," and "Report User."

A close "X" button.

* + - * + Component type or format

React functional component (using hooks).

Uses Formik for form handling and validation of the password input.

Styled with CSS (potentially using a CSS-in-JS library or a UI component library for consistent styling).

* + - * + Additional Front-end Components (Potentially):

**Avatar:** A reusable component to display the user's avatar.

**TextInput:** A reusable input field component for the password input.

**Button:** A reusable button component for the "Visit" action.

**Link:** A reusable component for the navigation links at the bottom.

* + - Back-end component(s)
      * PortfolioVisitController
        + Description and purpose

This REST controller handles requests to verify the password and grant access to the portfolio. It exposes an endpoint for:

Receiving the portfolio link and password.

Authenticating the password against the stored password for the portfolio.

Redirecting the user to the portfolio page if the password is correct.

Returning an error if the password is incorrect.

* + - * + Component type or format

Spring Boot REST controller (Java class annotated with @RestController).

Uses @RequestMapping to map HTTP requests to methods.

Returns JSON responses or redirects to the portfolio page.

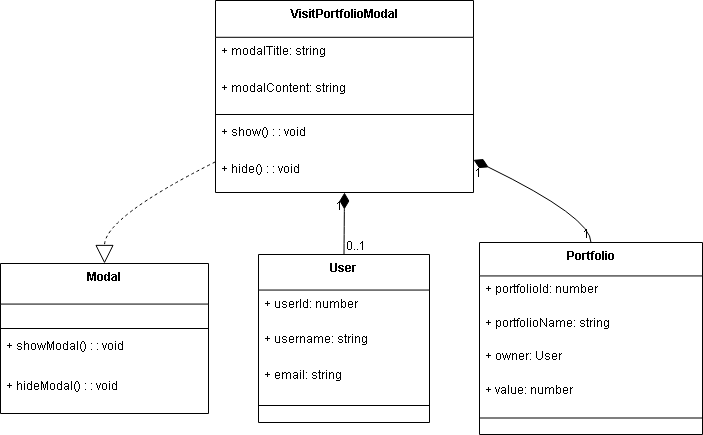
* + - * + Additional Back-end Components:

**PortfolioVisitService:** A service layer that encapsulates the business logic for password verification and portfolio access.

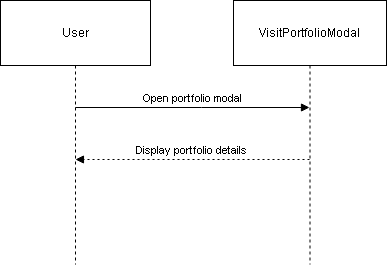
**PortfolioShareRepository:** A repository layer that interacts with the database to retrieve the portfolio sharing information (including the password).

**PortfolioShare Entity:** A JPA entity representing the portfolio sharing information, including the link and password.

* + - Object-Oriented Components
      * Class Diagram



* + - * Sequence Diagram



* + - Data Design
      * ERD or schema

