SDD

MU Connect: Mobile Application for MU Students and Alumni

---

1. Introduction

This Software Design Specification (SDS) provides a detailed description of the design and architecture of the *MU Connect* mobile application. The document outlines the technical design required to implement the functional and non-functional requirements defined in the Software Requirements Specification (SRS).

The purpose of this document is to provide a clear understanding of the system's design, its modules, interfaces, data flow, and architecture. It serves as a guideline for the development team, project managers, and Mahindra University (MU) stakeholders, ensuring consistency in implementation and adherence to requirements.

1.1 Purpose

The primary purpose of this document is to describe the software design of the *MU Connect* mobile application, which will act as a centralized platform for students and alumni of Mahindra University. This document bridges the gap between requirements analysis and implementation by providing technical details essential for developers.

This document is intended for the following audience:

- *Developers*: To implement the system based on design specifications.
- *Project Managers*: To monitor development progress and maintain alignment with project objectives.
- *MU Administration*: To verify that the design complies with institutional policies and security guidelines.
- *Testers*: To create test cases based on the design.
- *Maintenance Teams*: For future updates and troubleshooting.

The document structure follows standard SDS guidelines and includes sections like Use Case View, Design Overview, Logical View, Data View, Exception Handling, Configurable Parameters, and Quality of Service.

1.2 Scope

This Software Design Specification applies to the *MU Connect* mobile application, designed for both Android and iOS platforms, targeted for the students and alumni of Mahindra University.

The design outlined in this document affects the following areas of the application:

- User Authentication & Profile Management
- Networking and Communication between Students & Alumni
- Event Management & Registration
- Job Board and Career Resources
- Academic Resources Access
- Notifications & Newsfeed System
- Backend API Integration
- Data Security and Privacy Handling

This document guides all phases of the system development life cycle, including implementation, testing, deployment, and maintenance of *MU Connect*.

1.3 Definitions, Acronyms, and Abbreviations

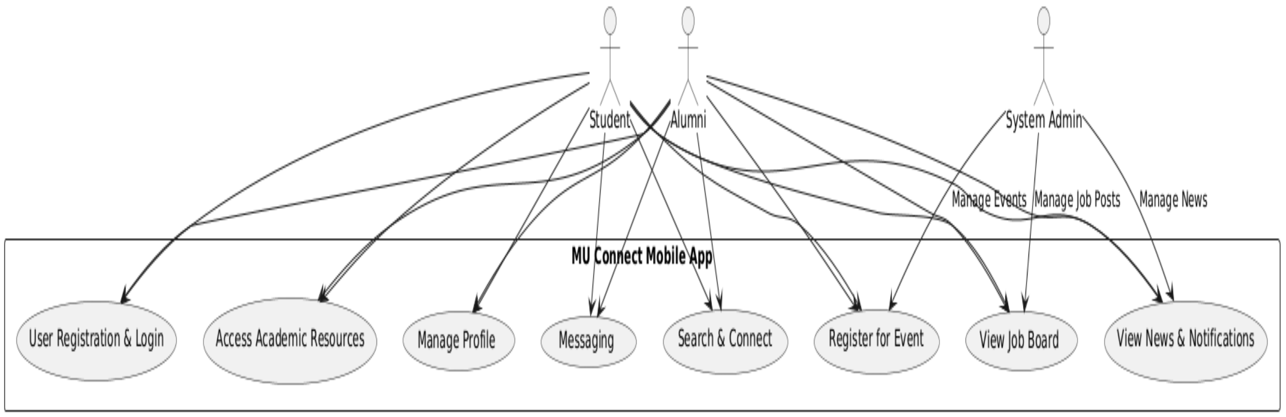
Term / Acronym	**Definition**
MU	Mahindra University
SRS	Software Requirements Specification
SDS	Software Design Specification
API	Application Programming Interface
UI	User Interface
UX	User Experience
FCM	Firebase Cloud Messaging (Used for push notifications)
DB	Database
Android	Mobile Operating System by Google
iOS	Mobile Operating System by Apple

1.4 References

Document Name	**Reference/Source**
Software Requirements Specification (SRS) for MU Connect	Prepared by Team NEXORA
IEEE Software Design Documentation Standards	IEEE Std 1016-2009
Firebase Cloud Messaging Documentation	https://firebase.google.com/docs/cloud-messaging

Document Name	Reference/Source
Android Developer Documentation	https://developer.android.com/
iOS Human Interface Guidelines	https://developer.apple.com/design/human-interface-guidelines/
Mahindra University Branding Guidelines	Internal MU Document

2. Use Case View



Use Case 1: User Registration & Login

Description:

Allows students and alumni to create an account and log in securely using their MU email ID or student ID.

Actors:

- Student
- Alumni

Usage Steps:

1. User opens the MU Connect App.
2. Selects "Register" or "Login".
3. Enters MU email ID / student ID and password.
4. System verifies credentials.
5. Successful login redirects to Home Page.

Use Case 2: Manage Profile

Description:

Allows users to create and update their profile with academic and professional details.

Actors:

- Student
- Alumni

Usage Steps:

1. User navigates to Profile Page.
2. Edits personal, academic, and professional information.
3. Saves changes.
4. System updates the database.

Use Case 3: Search & Connect (Networking)

Description:

Enables students and alumni to search for other users and send connection requests.

Actors:

- Student
- Alumni

Usage Steps:

1. User accesses the Networking Page.
2. Searches users by name, graduation year, or major.
3. Sends connection requests.
4. Connected users can chat directly.

Use Case 4: Messaging (Chat)

Description:

Provides real-time one-to-one or group chat functionality between connected users.

Actors:

- Student
- Alumni

Usage Steps:

1. User opens Chat section.
2. Selects connection or group.
3. Sends and receives messages in real-time.

Use Case 5: Access Academic Resources

Description:

Allows users to view and download academic materials provided by the university.

Actors:

- Student

Usage Steps:

1. User navigates to Academic Resources section.
2. Selects resource category (Library, Course Materials, Calendar).
3. Views or downloads the content.

Use Case 6: View Job Board

Description:

Provides students and alumni access to job postings, internships, and career resources.

Actors:

- Student
- Alumni

Usage Steps:

1. User accesses Job Board section.
2. Views available job postings.
3. Applies for jobs via external or internal links.

Use Case 7: View Notifications

Description:

Displays announcements, and personalized notifications.

Actors:

- Student
- Alumni
- Admin (Manage News)

Usage Steps:

1. User receives push notification for news or updates.
2. User views detailed news on Newsfeed.
3. Admin updates or posts new content from backend.

---

3. Design Overview

This section provides a high-level overview of the software design for *MU Connect*. It outlines the design objectives, system architecture, modular decomposition, dependencies, and external interfaces that the application interacts with.

The system is designed using a modular and layered architecture to ensure maintainability, scalability, and security. The architecture supports both Android and iOS platforms using a common backend with RESTful APIs.

3.1 Design Goals and Constraints

Design Goals:

- Develop a cross-platform mobile application (Android & iOS).

- Ensure secure user authentication and data privacy.
- Provide a scalable backend to handle increasing user data and interactions.
- Deliver intuitive and user-friendly UI/UX for both students and alumni.
- Enable efficient networking and real-time messaging.
- Support modular design for future enhancements and maintainability.

Constraints:

- The application must adhere to Mahindra University's branding guidelines.
- The backend must use REST APIs for communication.
- Data privacy compliance must follow standard encryption protocols.
- Application should load within 3 seconds on stable internet.
- APIs should have a response time of less than 500ms for 95% of requests.
- Push notifications are to be handled using Firebase Cloud Messaging (FCM).
- Development Timeline: Completion within the Software Engineering Lab semester.

Development Tools & Technologies:

- Frontend: Android Studio (Kotlin), Xcode (Swift)
- Backend: Node.js with Express or Django REST API
- Database: MySQL or MongoDB
- Notification Service: Firebase Cloud Messaging (FCM)
- Design Tools: Figma for UI/UX Design
- Version Control: GitHub Repository

3.2 Design Assumptions

- Users will have a stable internet connection for real-time features like messaging.
- Mahindra University will provide access to relevant academic and event databases.
- Third-party services like Firebase will be available and operational.
- Users will access the app using mobile devices (smartphones or tablets) with Android 8+ or iOS 12+.
- Admin backend interface will be minimal for managing events, job posts, and news updates.

3.3 Significant Design Packages

The application is decomposed into the following main modules:

Package/Module	Description

Package/Module	Description
Authentication Module	Handles user registration, login, and password management.
Profile Management Module	Allows users to create, view, and update their profiles.
Networking Module	Manages search, connect, and messaging functionality.
Event Management Module	Displays events and handles event registration.
Academic Resources Module	Provides access to academic materials and resources.
Job Board Module	Displays job postings and career-related resources.
Notification Module	Handles real-time notifications using Firebase.
Admin Module	Allows Admin to manage news, events, and job postings.

3.4 Dependent External Interfaces

The application depends on the following external interfaces for functioning:

External Application	Module Using the Interface	Functionality / Description
University Database API	Academic Resources Module, Event Management Module	Used to fetch data related to academic resources, course materials, events, and schedules.
Firebase Cloud Messaging (FCM)	Notification Module	Used to send push notifications to users about messages, events, or announcements.
External Job Board API (Optional/Future Scope)	Job Board Module	To fetch external job listings and career opportunities if integrated.

3.5 Implemented Application External Interfaces

The application provides the following public interfaces that may be accessed by other university systems or applications:

Interface Name	Module Implementing the Interface	Functionality / Description
User Management API	Authentication Module	Provides user registration, login, and profile management services.
Event Management API	Event Management Module	Allows for event creation, management, and registration functionalities.
News & Announcement API	Admin Module	Provides endpoints for creating and managing university news or announcements.

Absolutely! Here's *Section 4: Logical View* for your *MU Connect* Software Design Specification (SDS), customized as per your SRS.

---

4. Logical View

This section provides a detailed view of the *MU Connect* application's internal design. The system is designed in layered architecture, enabling separation of concerns and modular development. Each module is responsible for specific functionality and interacts with other modules to fulfill the requirements of the application.

4.1 Design Model

The system follows a multi-layered architecture:

Layers of Architecture:

1. Presentation Layer (UI Layer)
2. Application Layer (Controller / Business Logic)
3. Data Access Layer (API Communication)
4. Database Layer (Persistent Storage)

Module-wise Class Design Overview:

1. Authentication Module

Class	**Responsibilities**
LoginManager	Manages user login & authentication.
RegisterManager	Handles user registration process.
SessionHandler	Maintains user session after login.

---

2. Profile Management Module

Class	**Responsibilities**
ProfileManager	Handles CRUD operations on user profile data.
UserProfile	Data class representing user attributes (name, email, academic info, skills).

---

3. Networking & Messaging Module

Class	**Responsibilities**
SearchManager	Implements search functionality for students and alumni.
ConnectionManager	Manages connection requests and acceptance.
ChatManager	Handles real-time messaging and chat data.

---

4. Event Management Module

Class	**Responsibilities**
EventManager	Manages viewing, registration, and storing events data.
Event	Entity class representing event details.

---

5. Academic Resources Module

Class	**Responsibilities**
ResourceManager	Provides access to academic content like course materials, library resources.

Class	Responsibilities
Resource	Entity class representing resources.

---

6. Job Board Module

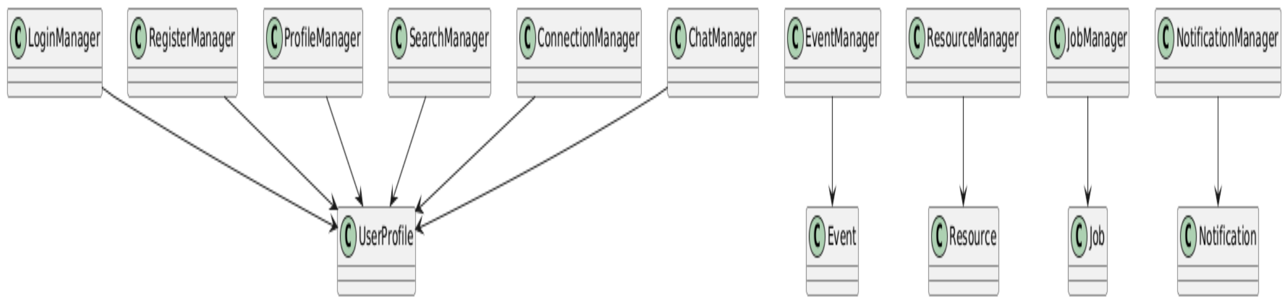
Class	Responsibilities
JobManager	Fetches job postings and career resources.
Job	Entity class representing job details.

---

7. Notification Module

Class	Responsibilities
NotificationManager	Handles push notifications using Firebase Cloud Messaging (FCM).
Notification	Entity class for notifications content.

Class Diagram



4.2 Use Case Realization

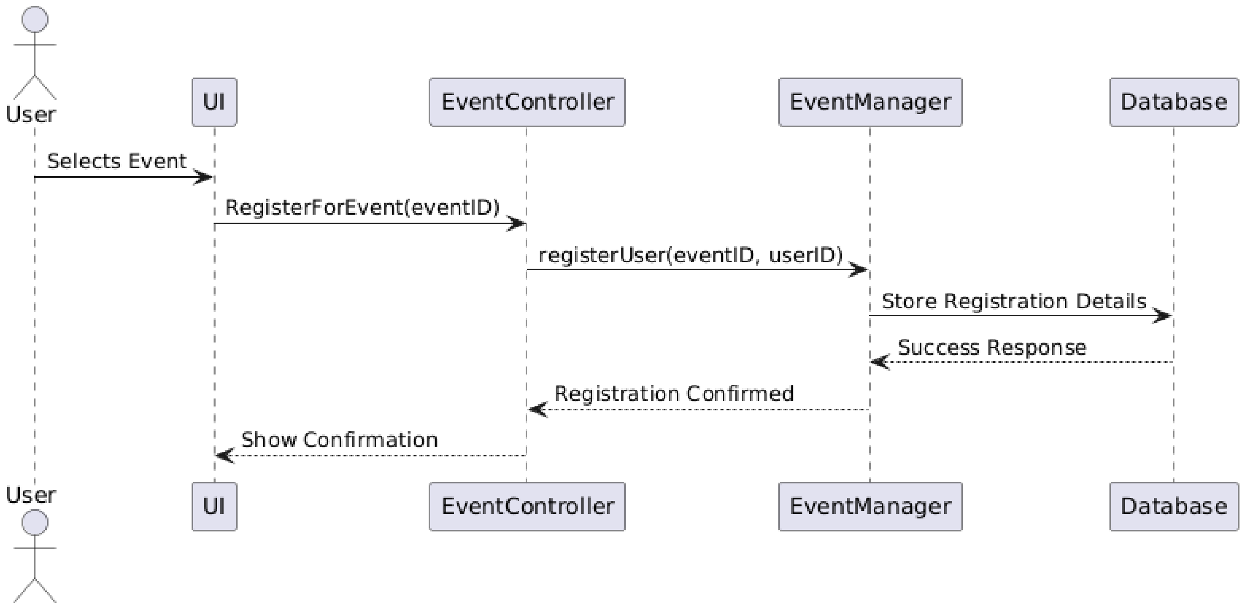
This section explains how the main use cases defined in Section 2 are implemented using the system design.

Example 1: Use Case Realization — Register for Event (UC5)

Sequence of Interaction:

1. User selects event from UI.
2. UI Layer invokes EventController.
3. EventController calls EventManager to fetch or update event data.
4. EventManager communicates with Database/API for registration.
5. Confirmation response sent back to UI for user feedback.

Sequence Diagram



Perfect! Here's *Section 5: Data View* for your *MU Connect* Software Design Specification (SDS), fully customized based on your SRS.

---

5. Data View

This section describes the persistent data storage perspective of the *MU Connect* mobile application. The system stores data related to users, profiles, events, academic resources, jobs, messages, and notifications.

The database follows a relational model using either MySQL or Firebase Realtime Database for storing structured data.

5.1 Domain Model

The Domain Model represents the core entities of the *MU Connect* system and their relationships. The major entities in the domain model are:

Entities:

- User
- Profile
- Connection
- Event
- AcademicResource
- JobPost
- Message
- Notification

5.2 Data Model (Persistent Data View)

The following is an overview of the data model representing the structure of key database tables.

Entity Descriptions:

Entity	Attributes	Description
User	UserID (PK), Email, Password, Role (Student/Alumni/Admin)	Stores authentication details.
Profile	ProfileID (PK), UserID (FK), Name, Branch, GraduationYear, Skills, Bio	Stores personal and academic information of the user.
Connection	ConnectionID (PK), UserID1 (FK), UserID2 (FK), Status	Represents connections between users.
Event	EventID (PK), Title, Description, Date, Location	Stores event details.
EventRegistration	RegistrationID (PK), EventID (FK), UserID (FK)	Stores event registration data.
AcademicResource	ResourceID (PK), Title, Description, FileLink	Stores academic materials.
JobPost	JobID (PK), Title, Company, Description,	Stores job postings and

Entity	Attributes	Description
	Link	details.
Message	MessageID (PK), SenderID (FK), ReceiverID (FK), Content, Timestamp	Stores chat messages between users.
Notification	NotificationID (PK), UserID (FK), Content, Type, Timestamp	Stores notifications sent to users.

5.2.1 Data Dictionary

Field Name	Description	Data Type	Possible Values
UserID	Unique Identifier for User	String	Alphanumeric
Email	User's Email Address	String	Valid Email Format
Role	User Role	String	Student / Alumni / Admin
Name	User's Full Name	String	Text
EventID	Unique Identifier for Event	String	Alphanumeric
EventTitle	Name of the Event	String	Text
EventDate	Date of the Event	Date	DD-MM-YYYY
JobID	Unique Identifier for Job Post	String	Alphanumeric
JobTitle	Job or Internship Title	String	Text
Company	Name of the Company Offering the Job	String	Text
MessageID	Unique Identifier for Message	String	Alphanumeric
SenderID	User ID of Message Sender	String	UserID
ReceiverID	User ID of Message Receiver	String	UserID
NotificationID	Unique Identifier for Notification	String	Alphanumeric
Content	Content of Message / Notification	String	Text

Excellent! Here's *Section 6: Exception Handling* for your *MU Connect* Software Design Specification (SDS), customized as per your SRS document.

---

6. Exception Handling

This section describes the exception handling strategy for the *MU Connect* mobile application. The system is designed to handle common runtime errors, user input errors, and system-level exceptions to ensure robustness, user-friendly error reporting, and secure logging.

Exception handling ensures that errors do not crash the application and provide meaningful feedback to the user while securely logging the technical details for developers.

6.1 Types of Exceptions Handled

Exception Type	Description	Handling Strategy
AuthenticationException	Occurs during user login/registration failures (wrong credentials, invalid email).	Display error message: "Invalid Email or Password". Log attempt for security monitoring.
NetworkException	Occurs when the user has no internet connection or server is unreachable.	Display user-friendly message: "No Internet Connection. Please try again later." Retry mechanism enabled.
DataNotFoundException	Data requested by the user is not available in the database (e.g., no event found).	Show message: "Requested Data Not Available." Log for debugging if frequent.
InputValidationException	Occurs when user inputs invalid data (e.g., empty fields during registration).	Display error next to the invalid input field. Prompt user to correct input.
DatabaseException	Occurs when database read/write operations fail.	Display generic error: "Unable to process request. Please try again." Log detailed error for developer review.
MessageSendFailureException	Error sending message in chat due to network or server issue.	Notify user: "Message not sent. Check connection." Allow retry.
NotificationFailureException	Failure in delivering push notification.	Silent fail — log the error for admin/developer monitoring. No user interruption.

6.2 Follow-Up Actions

Exception Type	Follow-Up Action
Authentication Failures	Lock account after multiple failed attempts (e.g., 5 tries) to prevent brute-force attacks.
Network Errors	Retry mechanism with exponential backoff for critical operations.
Database Errors	Notify technical team if errors persist via admin dashboard monitoring.
Input Errors	Guide users with clear error messages and prevent form submission until corrected.
Unhandled Exceptions	Default error handler will capture and log the error, then show a generic friendly message to users: "Something went wrong. Please try again later."

7. Configurable Parameters

This section lists the configurable parameters used in the *MU Connect* mobile application. These parameters are used for application settings that may require changes without altering the source code.

Configurations are stored in a configuration file or environment variables and can be easily modified by the system administrator or developer without redeploying the entire application.

Some parameters can be updated dynamically (without restarting the application), while others may require a restart.

Configurable Parameters Table

Configuration Parameter Name	Definition and Usage	Dynamic? (Yes/No)
API_BASE_URL	Defines the base URL of the backend API server to fetch data for the application.	No
FCM_SERVER_KEY	Firebase Cloud Messaging key for sending push notifications.	No
MAX_LOGIN_ATTEMPTS	Maximum allowed failed login attempts before account lockout (for security purposes).	Yes
APP_THEME	Controls the theme of the app (Light / Dark Mode).	Yes
EVENT_REMINDER_TIME	Time (in minutes) before an event to send reminder notifications.	Yes
DEFAULT_LANGUAGE	Default language of the app (currently set to English).	Yes
SUPPORT_EMAIL	Email address for user support and feedback. Displayed in the app settings.	Yes
PAGINATION_LIMIT	Number of records to fetch per page in lists (Events, Jobs, Users).	Yes
PROFILE_PICTURE_SIZE_LIMIT	Maximum size (in MB) allowed for profile picture uploads.	No

---

8. Quality of Service

This section outlines the quality attributes of the *MU Connect* mobile application related to availability, security, performance, and monitoring. These aspects ensure that the application is reliable, secure, scalable, and manageable in production.

8.1 Availability

MU Connect is designed to have an availability of 99.9% uptime during operational hours.

Design Features to Support Availability:

- Cloud-based hosting ensures scalability and high availability.
- Load Balancing to distribute user requests evenly across servers.
- Use of Firebase Realtime Database and Cloud Functions for minimal downtime.
- Push Notifications using Firebase Cloud Messaging (FCM) to ensure timely delivery without the need for constant server polling.
- Failover Mechanisms for backend services to minimize downtime in case of server failure.

Factors Affecting Availability:

- Scheduled Maintenance: Minimal downtime required for updates and database migrations — will be planned during non-peak hours.
- Bulk Data Operations: Mass user updates or data loading will be scheduled during low-traffic hours to avoid affecting performance.

8.2 Security and Authorization

The *MU Connect* application ensures secure access to features and user data by implementing the following security measures:

Authentication & Authorization:

- Secure login system using MU Email or Student ID.
- Role-based access control (RBAC) — roles include:
  - Student
  - Alumni
  - Admin

Data Protection:

- User passwords stored in encrypted format.

- All sensitive data transferred using HTTPS protocol.
- Profile visibility controlled by user privacy settings.

Authorization Features:

- Admin users can:
  - Manage Events.
  - Approve or manage Job Posts.
  - Post News and Announcements.

Additional Security Measures:

- Account Lockout Mechanism after multiple failed login attempts.
- Secure token-based authentication for API access.
- Input Validation at both frontend and backend to prevent injection attacks.

8.3 Load and Performance Implications

Based on the SRS, the performance requirements and expected load of *MU Connect* include:

Load Expectations:

- Initial user base: ~500+ users (students and alumni).
- Expected growth: Scalable to 2000+ users.
- Concurrent connections: Expected 100+ concurrent users during peak hours (events, job postings, announcements).

Performance Requirements:

Performance Metric	Expected Value
App Load Time	Less than 3 seconds on stable internet
API Response Time	Less than 500ms for 95% of requests
Push Notification Delivery	Within 5 seconds of trigger
Chat Message Delivery	Real-time (near instant)

Design Considerations for Performance:

- Database indexing for faster query execution.
- Pagination for displaying large datasets (Job posts, Event lists, User lists).
- Efficient caching of static content.

- Asynchronous processing for messaging and notifications.

8.4 Monitoring and Control

Monitoring and control mechanisms will be implemented to ensure that the *MU Connect* system is running smoothly and that any issues can be detected and resolved quickly.

Monitoring Tools:

- Firebase Crashlytics for error tracking and crash reporting.
- Performance Monitoring using Firebase Performance SDK.
- Server Health Monitoring via Cloud Provider (e.g., AWS, GCP).

Controllable Processes:

Process	**Monitoring Parameters**
User Authentication	Failed login attempts, unusual login patterns
Event Management	Number of registrations, API response failures
Messaging System	Message delivery success/failure rates
Push Notifications	Delivery status, FCM errors

Logging:

- Centralized error logging for backend processes.
- User activity logs for auditing purposes.
- Monitoring of API usage to prevent abuse.

Group Name: NEXORA

1) **C. Anirudh :- SE22UCSE059**
2) **N. Durga Prasad :- SE22UCSE176**
3) **P. Madhukar :- SE22UCSE191**
4) **B. Akhil Varma :- SE22UCSE058**
5) **Amarthya Raj :- SE22UCSE023**