Software Requirements Specification

For Alumni Blog Management System
Version 1.0

Prepared by:
Muhammad Ashir 22K-4504

Sayal Baig 22K-4625

Muhammad Ashar 22K-4471

Revision History

Name	Date	Reason for Changes	Version
Initial Draft	11/26/2024	Created initial SRS draft for the Alumni Blog System	1.0

1. Introduction

1.1 Purpose

The Alumni Blog Management System facilitates the connection between alumni and current students. It includes features for networking, event management, blogging, and referral tracking. The system enables seamless communication and engagement within the alumni community.

1.2 Document Conventions

This document uses bold headings for sections and subsections, and monospaced font for code or technical references. Requirements are uniquely labeled for traceability (e.g., REQ-001).

1.3 Intended Audience and Reading Suggestions

- **Developers**: For understanding the technical requirements.

- **Testers**: For creating test cases and validating functionalities.
- **End Users**: For insight into features and usability.

1.4 Product Scope

The Alumni Blog Management System includes the following features:

- User authentication and role-based access.
- Blog creation and management.
- Event management with a many-to-many user-event relationship.
- Referral page for job opportunities.
- Networking through events and news

1.5 References

1. React documentation: https://reactjs.org
2. Spring Boot documentation: https://spring.io/projects/spring-boot
3. PostgreSQL documentation: https://www.postgresql.org

---

2. Overall Description

2.1 Product Perspective

This system integrates with existing alumni databases and authentication systems, providing a comprehensive platform for alumni engagement.

2.2 Product Functions

- User registration and authentication.
- CRUD operations for blogs and events.
- Networking and communication via news and events.
- Event management with attendee tracking.

2.3 User Classes and Characteristics

- **Alumni**: Create blogs, register for events, and network with peers.
- **Students**: Access job referrals and engage in forums.
- **Admin**: Manage users, blogs, and events.
- **Event:** event description and timings etc.
- **Job:** Add job, job description
- **News:** news about alumni and students.
- **Post**

- **Comment**
- **Category**
- **Donation**
- **Role**
- **Blog**
- **Project**

2.4 Operating Environment

- **Frontend**: React JS.
- **Backend**: Spring Boot.
- **Database**: PostgreSQL.
- **Deployment Environment**: Localhost during development; cloud server in production.

2.5 Design and Implementation Constraints

- Must comply with university privacy policies.
- Data storage in PostgreSQL with role-based access control (RBAC).

2.6 User Documentation

- User manual for navigation.
- Developer documentation for APIs.

2.7 Assumptions and Dependencies

- Assumes internet connectivity for cloud-hosted services.
- Depends on proper configuration of CORS and security settings.

---

3. External Interface Requirements

3.1 User Interfaces

- Login and Registration forms.
- Blog creation page.
- Event management dashboard.

3.2 Hardware Interfaces

- Server hosting requirements: Minimum 8GB RAM, 4-core CPU.

3.3 Software Interfaces

- **Frontend**: Axios for API calls.
- **Backend**: RESTful APIs for data exchange.

3.4 Communications Interfaces

- Protocols: HTTP/HTTPS.

---

4. System Features

4.1 Feature: Event Management

Description and Priority

- High priority: Allows creation, updating, and management of events.

Stimulus/Response Sequences

1. User navigates to the Event page.
2. Creates an event by entering details and selecting attendees.

Functional Requirements

- REQ-001: Allow event creation with the following attributes:
    - Name, location, timing, venue, organizer, description, registration flag.
- REQ-002: Establish a many-to-many relationship between events and users.

---

5. Nonfunctional Requirements

5.1 Performance Requirements

- Good for small scale application

5.2 Security Requirements

- Use BCrypt for password hashing.
- Implement role-based access control.

5.3 Software Quality Attributes

- Maintainability: Modular code structure for easy updates.
- Usability: Intuitive UI with responsive design.

---

6. Glossary

- **RBAC**: Role-Based Access Control.
- **WYSIWYG**: What You See Is What You Get, editor.

This document provides a comprehensive and precise foundation for the Alumni Blog Management System development. Adjust any section as per your project specifics.

SDS

6. System Architecture

This section provides a high-level overview of the system's structure and its partitioning into subsystems or components. It includes a general understanding of how the system is decomposed and how the individual components work together to provide the desired functionality.

6.1 System-Level Architecture

The system architecture decomposes the Alumni Blog Management System into distinct elements with clearly defined roles and relationships. Key areas for consideration include:

- System Decomposition into Elements:
    - Frontend: React-based web interface for user interaction.
    - Backend: Spring Boot RESTful APIs for business logic and data management.
    - Database: PostgreSQL for persistent storage of user, event, and blog data.
- Relationship Between Elements:
    - Frontend communicates with the backend through HTTP REST APIs.
    - Backend interacts with the database to store and retrieve data.
    - External systems like email services for notifications integrate through the backend.
- Interfaces to External Systems:
    - Email notification services (e.g., SMTP).
    - Third-party job referral APIs (if applicable).
- Major Physical Design Issues:
    - Frontend hosted on a client browser or static content delivery network (CDN).
    - Backend deployed on a cloud server or local virtual machine.
    - Database hosted on a managed cloud service or local infrastructure.
- Global Design Strategies:
    - Centralized error handling using exception resolvers in Spring Boot.
    - Secure authentication using JWT tokens.
    - Scalability through containerized deployment (e.g., Docker).

UML Diagrams:

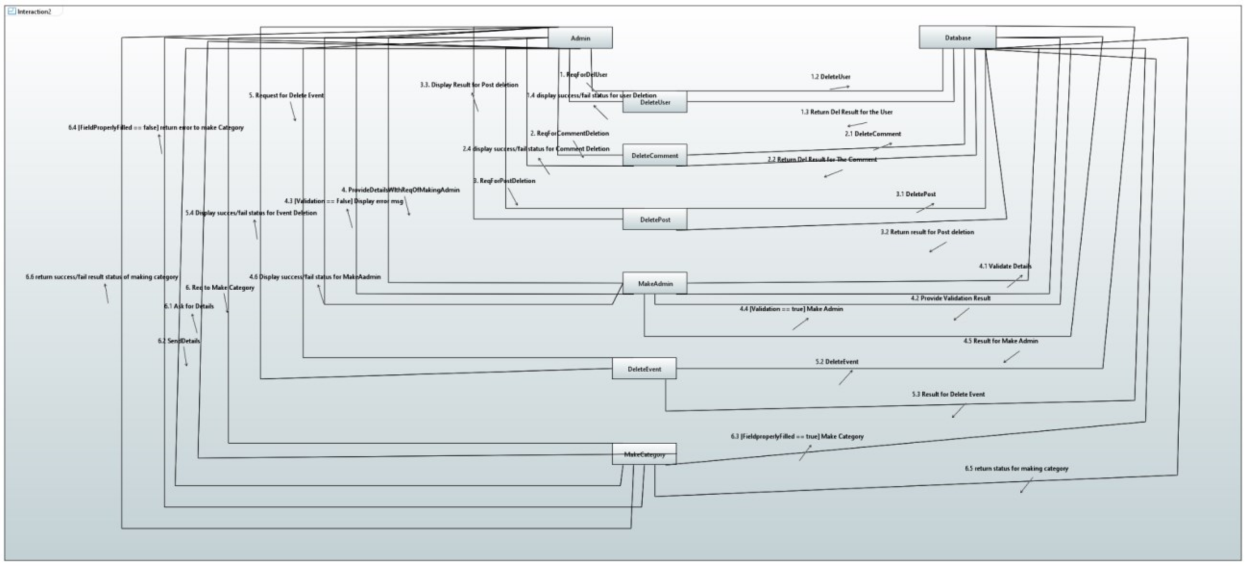
- Deployment Diagram: Represents the physical deployment of system components.
- Component Diagram: Illustrates the relationships between software components.

---

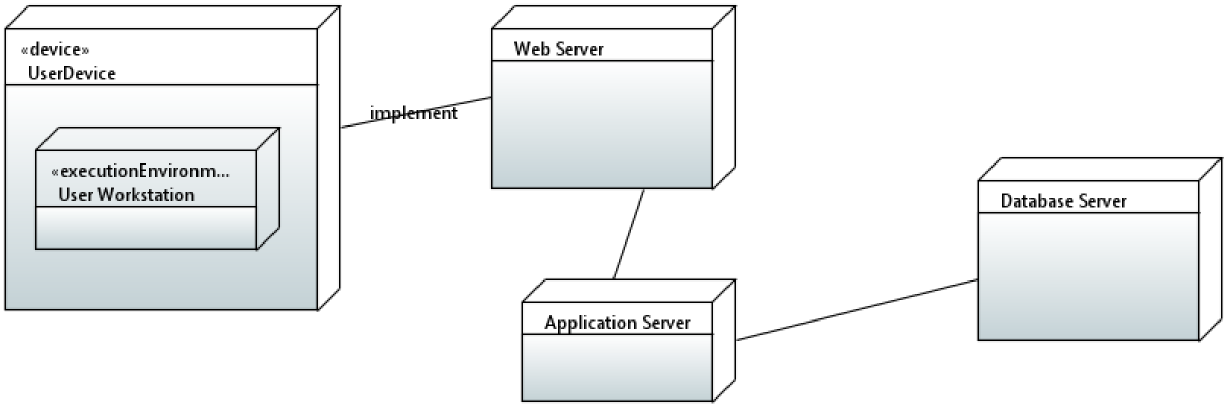
7. Application Design

This section provides detailed application design, including sequence diagrams, state transition diagrams, and activity diagrams to model the application's flow and behavior.

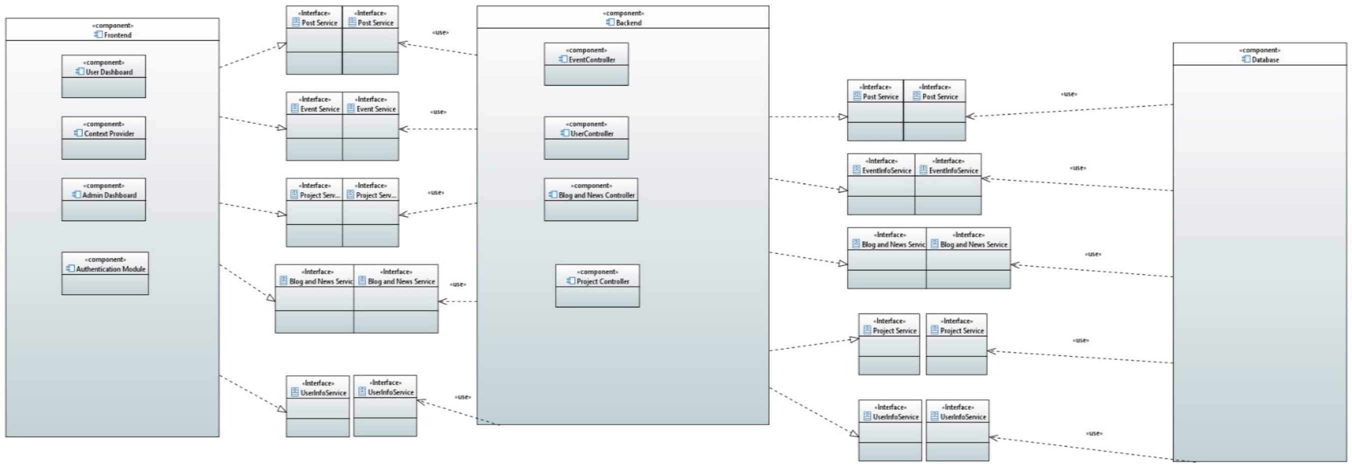
7.1 Communication Diagrams


Interaction2
Admin
Database
1. ReqForDelUser
1.2 DeleteUser
1.3 Return Del Result for the User
1.4 display success/fail status for user Deletion
DeleteUser
2. ReqForCommentDeletion
2.1 DeleteComment
2.2 Return Del Result for The Comment
2.4 display success/fail status for Comment Deletion
DeleteComment
3. ReqForPostDeletion
3.1 DeletePost
3.2 Return result for Post deletion
3.3. Display Result for Post deletion
DeletePost
4. ProvideDetailsWithReqOfMakingAdmin
4.1 Validate Details
4.2 Provide Validation Result
4.3 [Validation == False] Display error msg
4.4 [Validation == true] Make Admin
4.5 Result for Make Admin
4.6 Display success/fail status for MakeAadmin
MakeAdmin
5. Request for Delete Event
5.2 DeleteEvent
5.3 Result for Delete Event
5.4 Display succes/fail status for Event Deletion
DeleteEvent
6. Req to Make Category
6.1 Ask for Details
6.2 SendDetails
6.3 [FieldproperlyFilled == true] Make Category
6.4 [FieldProperlyFilled == false] return error to make Category
6.5 return status for making category
6.6 return success/fail result status of making category
MakeCategory

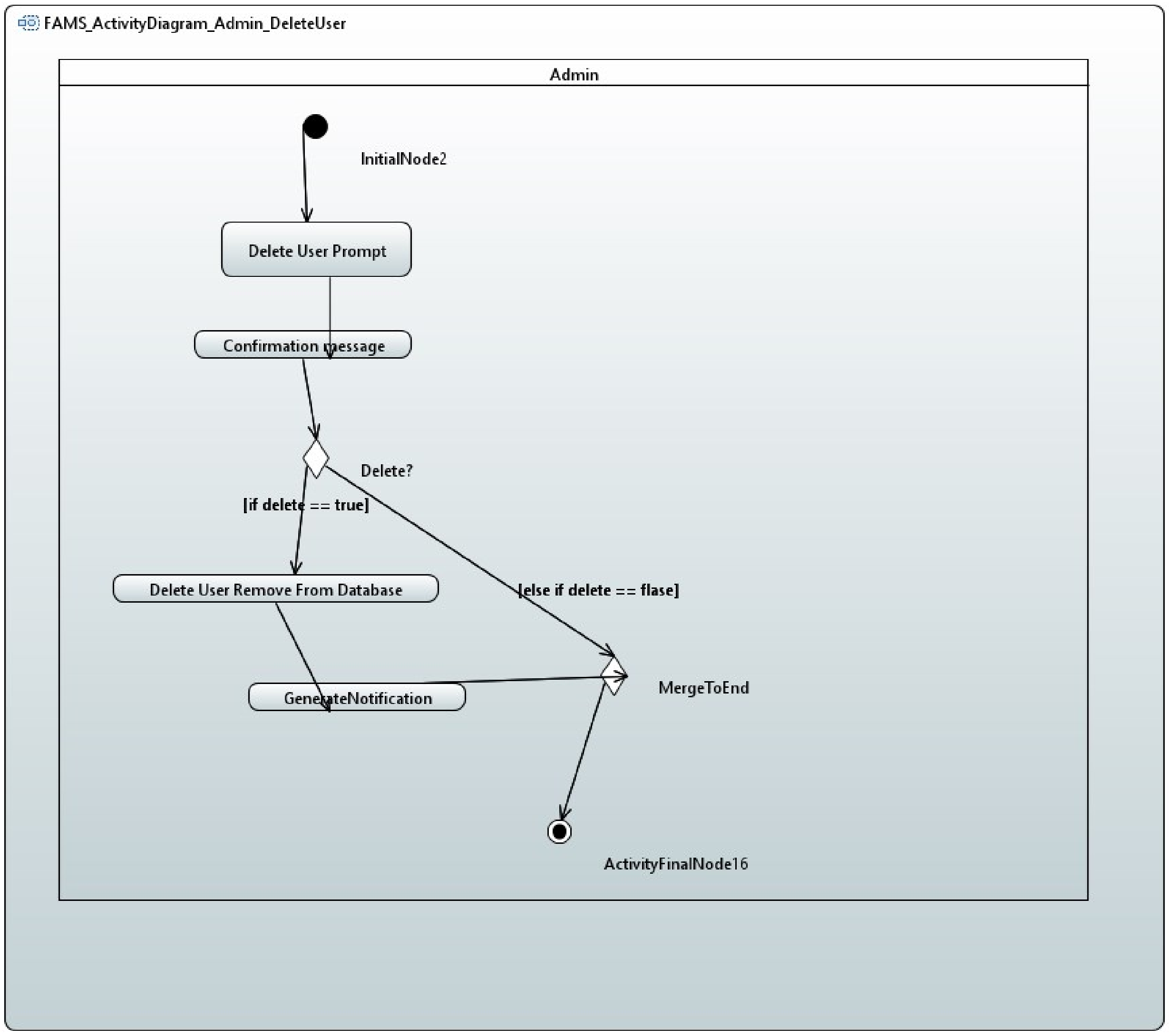
7.2 State Diagrams
Deployment diagram

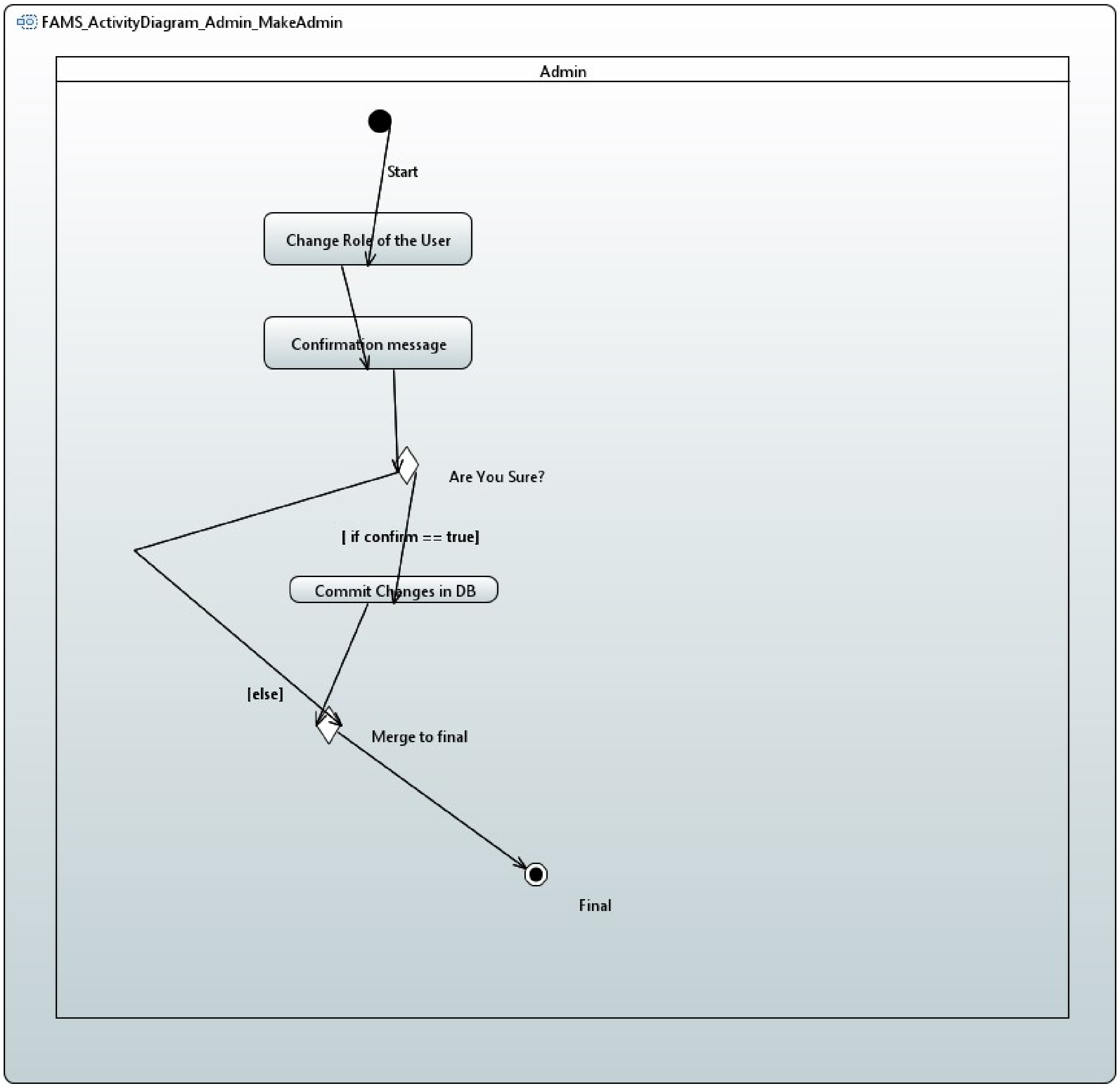


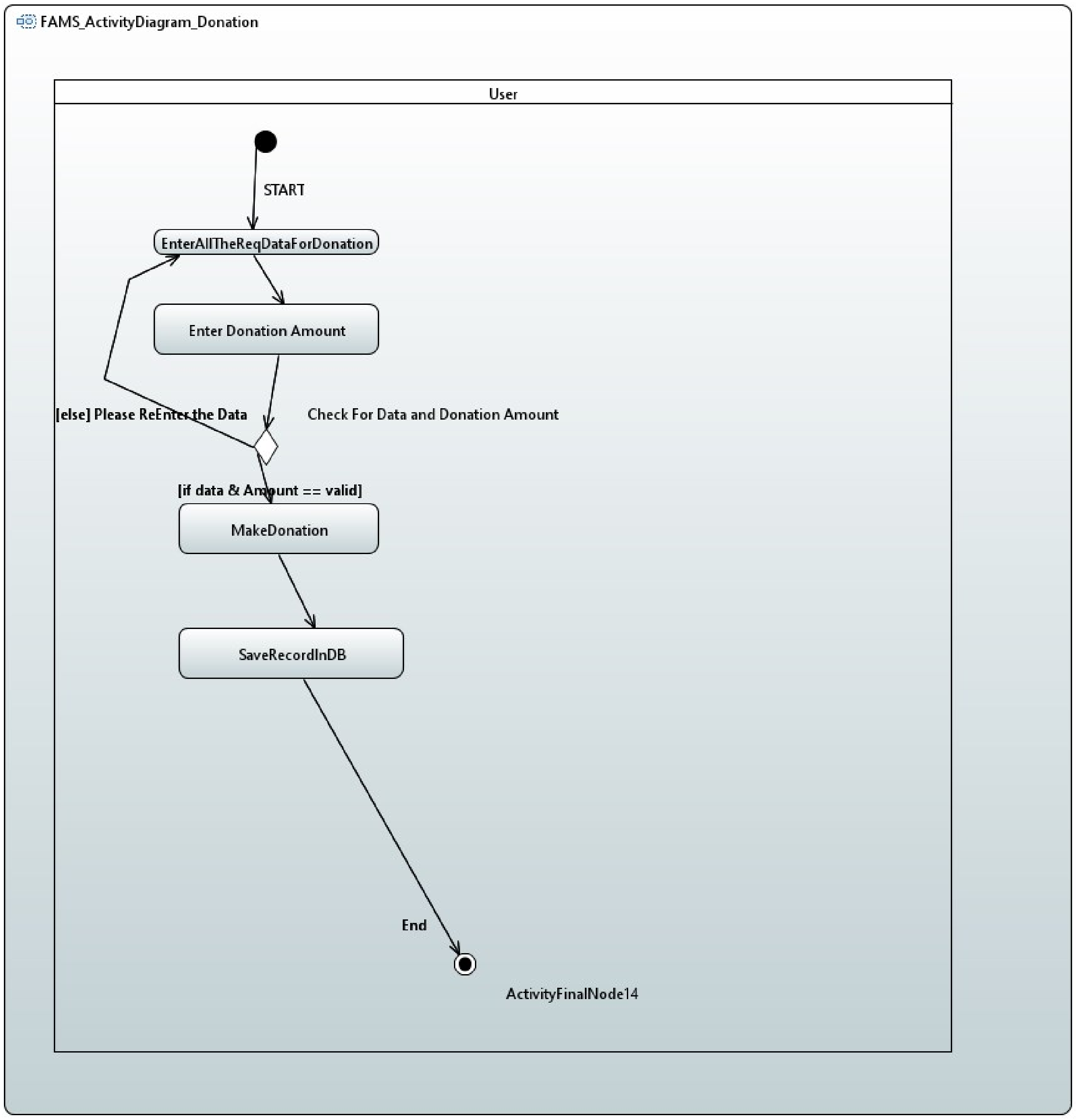
Component Diagram

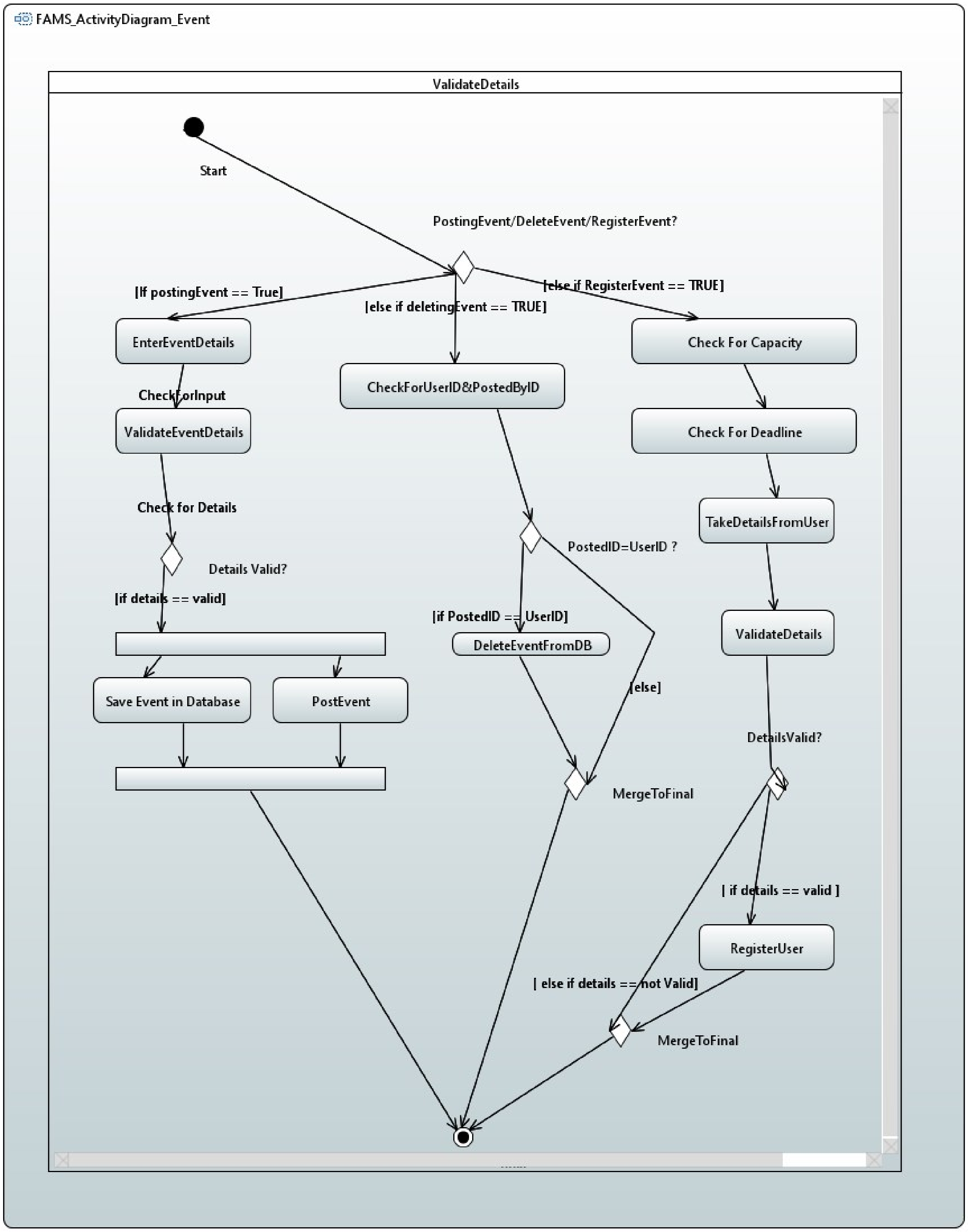


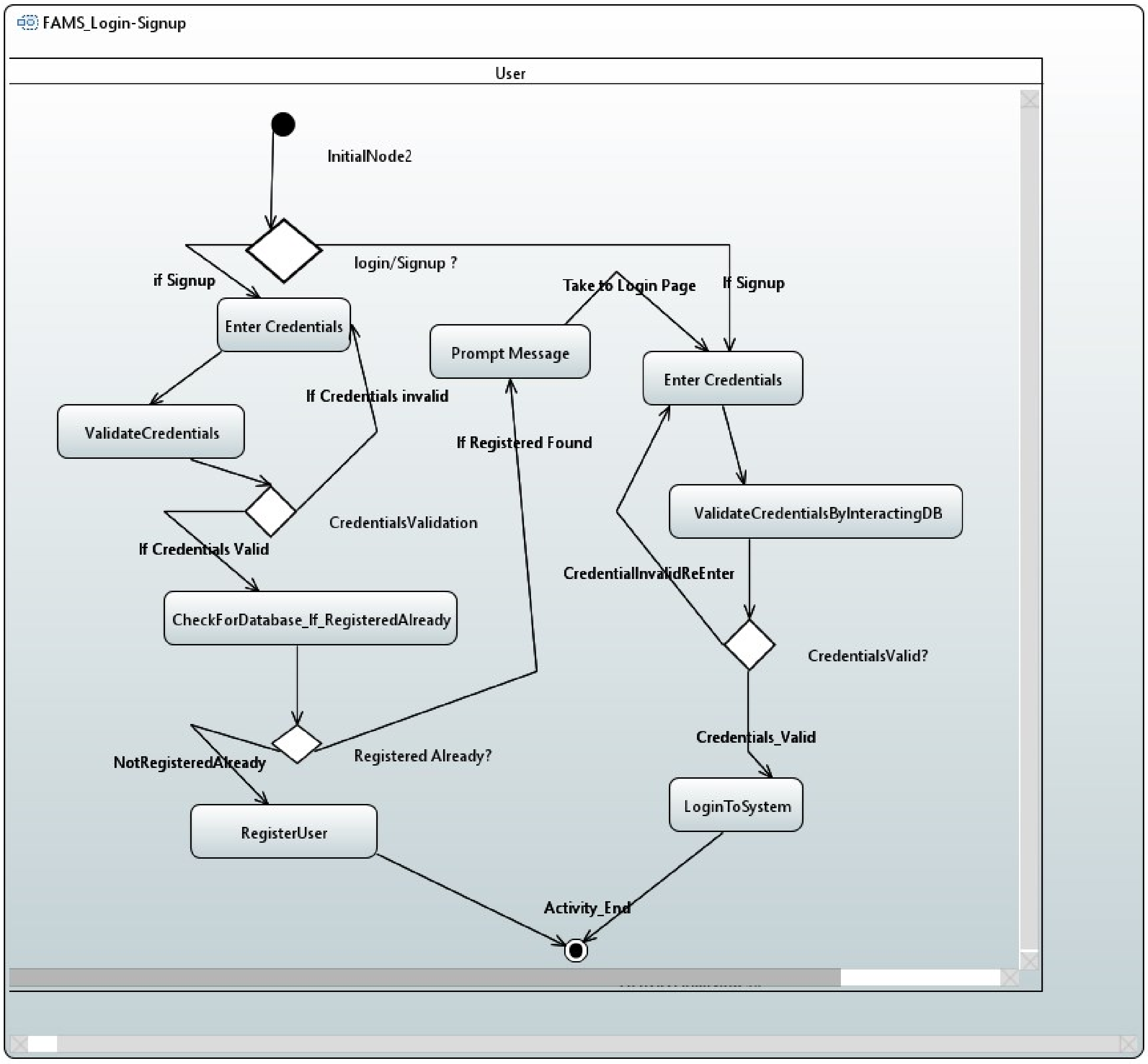
7.3 Activity Diagrams

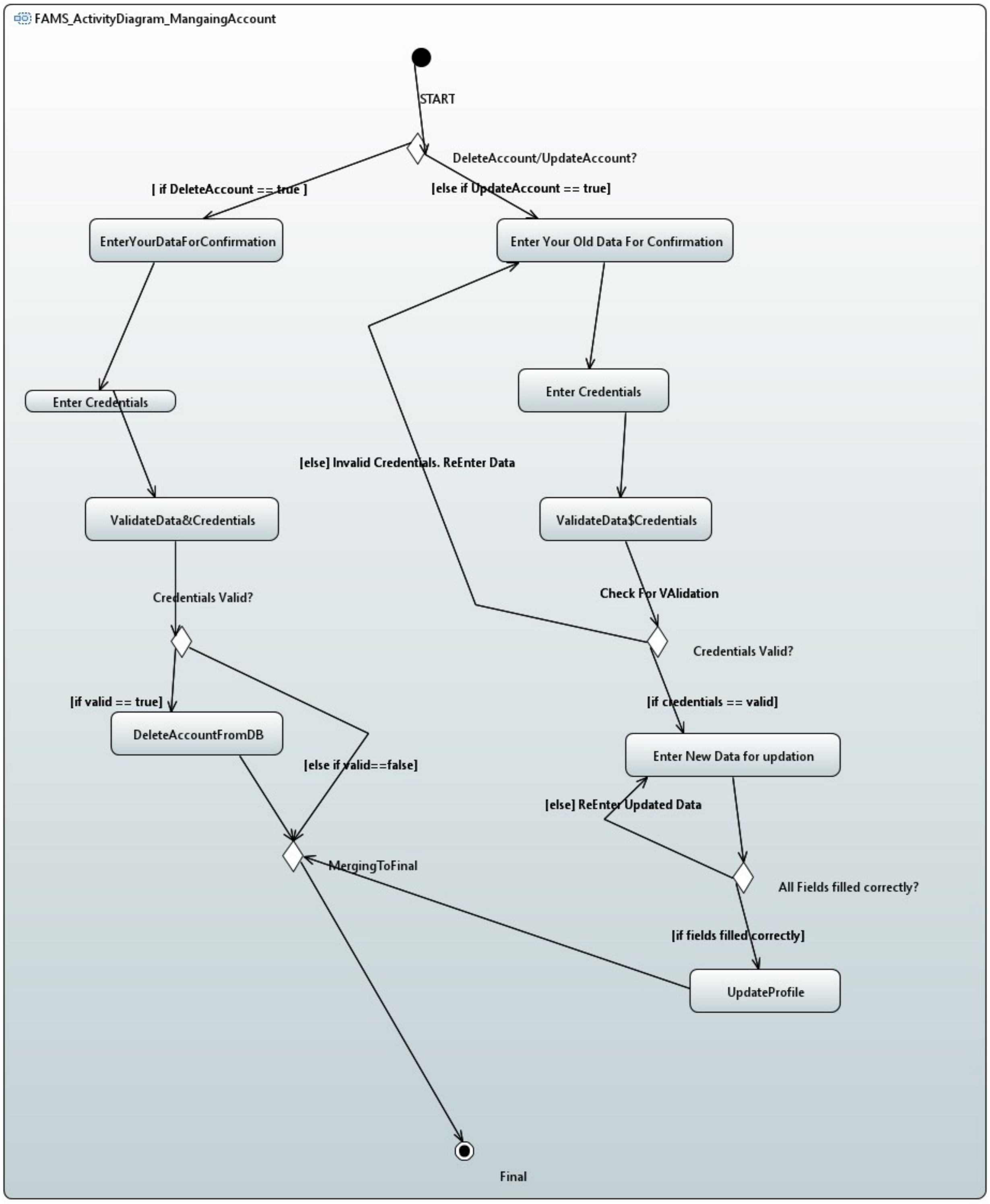

FAMS_ActivityDiagram_Admin_DeleteUser
Admin
InitialNode2
Delete User Prompt
Confirmation message
Delete?
[if delete == true]
Delete User Remove From Database
[else if delete == flase]
GenerateNotification
MergeToEnd
ActivityFinalNode16

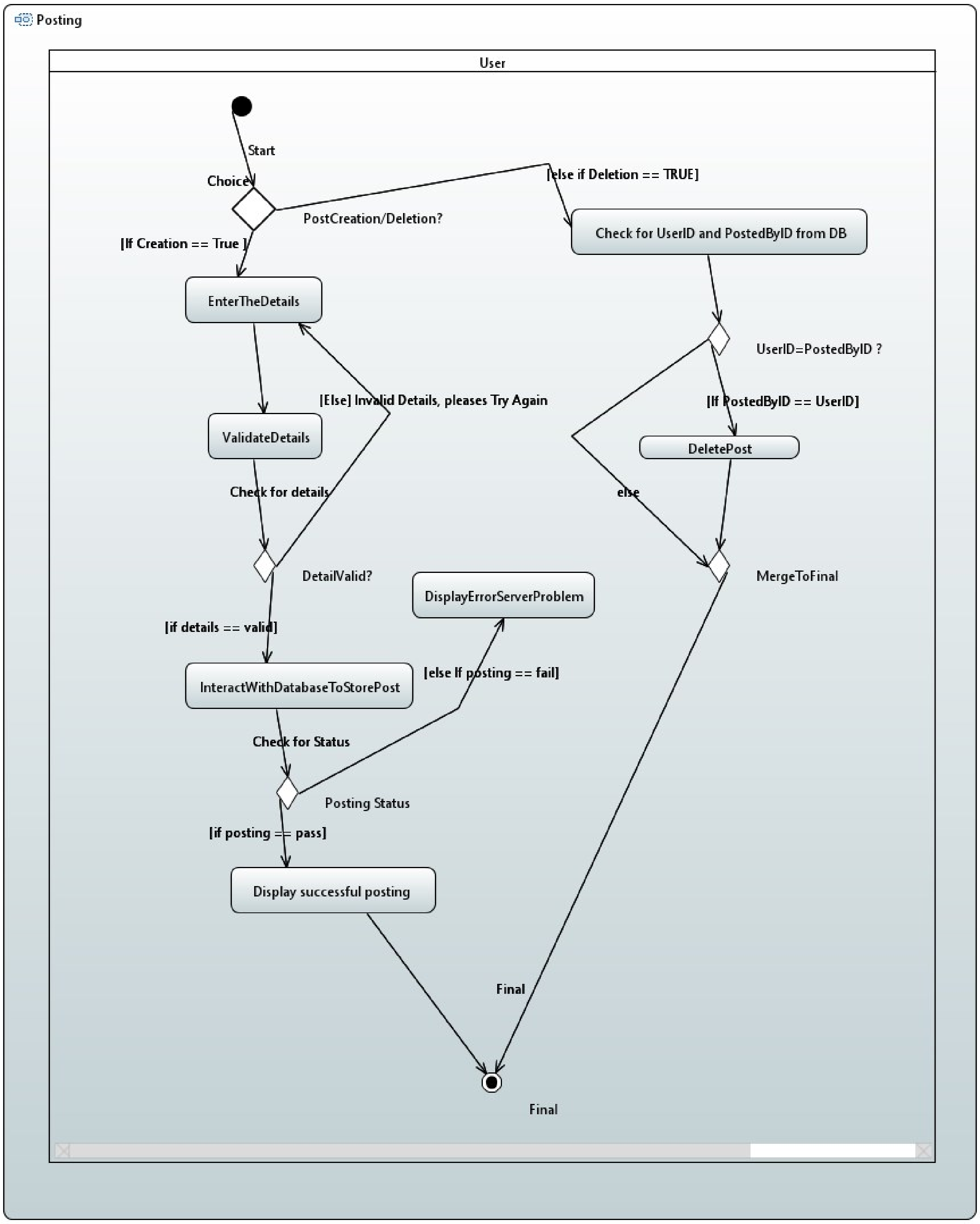

FAMS_ActivityDiagram_Admin_MakeAdmin
Admin
Start
Change Role of the User
Confirmation message
Are You Sure?
[if confirm == true]
Commit Changes in DB
[else]
Merge to final
Final


FAMS_ActivityDiagram_Donation
User
START
EnterAllTheReqDataForDonation
Enter Donation Amount
[else] Please ReEnter the Data
Check For Data and Donation Amount
[if data & Amount == valid]
MakeDonation
SaveRecordInDB
End
ActivityFinalNode14


FAMS_ActivityDiagram_Event
ValidateDetails
Start
PostingEvent/DeleteEvent/RegisterEvent?
[If postingEvent == True]
[else if deletingEvent == TRUE]
[else if RegisterEvent == TRUE]
EnterEventDetails
CheckForInput
ValidateEventDetails
Check for Details
Details Valid?
[if details == valid]
Save Event in Database
PostEvent
CheckForUserID&PostedByID
PostedID=UserID ?
[if PostedID == UserID]
DeleteEventFromDB
[else]
MergeToFinal
Check For Capacity
Check For Deadline
TakeDetailsFromUser
ValidateDetails
DetailsValid?
[if details == valid]
RegisterUser
[else if details == not Valid]
MergeToFinal


FAMS_Login-Signup
User
InitialNode2
login/Signup ?
if Signup
Enter Credentials
ValidateCredentials
If Credentials invalid
CredentialsValidation
If Credentials Valid
CheckForDatabase_If_RegisteredAlready
Registered Already?
NotRegisteredAlready
RegisterUser
Prompt Message
If Registered Found
Take to Login Page
If Signup
Enter Credentials
ValidateCredentialsByInteractingDB
CredentialInvalidReEnter
CredentialsValid?
Credentials_Valid
LoginToSystem
Activity_End


FAMS_ActivityDiagram_MangaingAccount
START
DeleteAccount/UpdateAccount?
[if DeleteAccount == true]
[else if UpdateAccount == true]
EnterYourDataForConfirmation
Enter Your Old Data For Confirmation
Enter Credentials
Enter Credentials
[else] Invalid Credentials. ReEnter Data
ValidateData&Credentials
ValidateData$Credentials
Check For VAlidation
Credentials Valid?
Credentials Valid?
[if valid == true]
[if credentials == valid]
DeleteAccountFromDB
Enter New Data for updation
[else if valid==false]
[else] ReEnter Updated Data
MergingToFinal
All Fields filled correctly?
[if fields filled correctly]
UpdateProfile
Final


Posting
User
Start
Choice
PostCreation/Deletion?
[else if Deletion == TRUE]
[If Creation == True]
Check for UserID and PostedByID from DB
EnterTheDetails
UserID=PostedByID ?
[Else] Invalid Details, pleases Try Again
[If PostedByID == UserID]
ValidateDetails
DeletePost
Check for details
else
DetailValid?
MergeToFinal
DisplayErrorServerProblem
[if details == valid]
[else If posting == fail]
InteractWithDatabaseToStorePost
Check for Status
Posting Status
[if posting == pass]
Display successful posting
Final
Final

Please note that the detailed system diagrams (such as architectural diagrams, data flow diagrams, class diagrams, etc.) are available in a separate folder dedicated to the diagrams. These diagrams complement the information provided in the document and offer a visual representation of the system design

---

8. References

1. IEEE Software Engineering Standards: https://www.ieee.org.
2. React Documentation: https://reactjs.org.
3. Spring Boot Documentation: https://spring.io/projects/spring-boot.
4. PostgreSQL Documentation: https://www.postgresql.org.

---

9. Appendices

This section contains supporting details and artifacts that complement the main body of the document, such as:

- Full database schema.
- API contract definitions (e.g., endpoints, request/response formats).
- Glossary of terms.