SOFTWARE REQUIREMENTS SPECIFICATION

for

BUP ALUMNI

Version 1.0

Prepared by :
MD. Hasibul Islam Mitul (2254901031)
Samia Maliha Lamin (2254901043)
Ishrat Jahan Binti (2254901071)
Anika Tasnim Mrittika (2254901115)

Submitted to :
Dr. Md. Musfique Anwar
Md. Sazzadul Islam Prottasha

October 14, 2025

Contents

Revision History

Revision	Date	Author(s)	Description
1.0	15.09.2025	Mitul,Samia, Binti, Mrittika	Chapter 1,2,3,4,5
2.0			
3.0			
4.0			

Chapter 1

Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) is to describe the functional and non-functional requirements of the BUP Alumni Networking Platform. The system is designed to connect alumni and current students of Bangladesh University of Professionals (BUP) by providing tools for networking, mentorship, event participation, and community services such as blood bank support. This document will serve as a reference for developers, testers ensuring a common understanding of the project objectives and requirements.

1.2 Document Conventions

This document follows the IEEE standard for Software Requirements Specification.

- Sections are numbered hierarchically (such as 1.1, 1.2, 1.3) for clarity.
- Technical terms such as **Node.js**, **MySQL**, and **RESTful API** are written in bold for emphasis.
- Code snippets and file names (e.g., `server.js`, `index.html`) are presented in monospace font for easy identification.

1.3 Intended Audience and Reading Suggestions

This document is intended for:

- **Developers:** To understand the requirements, system architecture, and technical details for implementation.
- **Testers:** To design and execute test cases based on functional and non-functional requirements.
- **Project Supervisors and Instructors:** To evaluate the completeness and quality of the project.
- **End Users (Students and Alumni):** To gain a high-level understanding of the platform's capabilities.

Readers who are primarily interested in the overall goals and features may focus on Section 3, while developers and testers should pay closer attention to Sections 3 and 4 for detailed requirements.

1.4 Project Scope

The BUP Alumni Networking Platform is a web-based system that enhances interaction between alumni and students. The main features include:

- User registration, authentication, and profile management.
- Event scheduling and participation for both alumni and students.
- Mentorship programs to connect alumni mentors with student mentees.
- A blood bank service providing emergency donor information.

The system is built with a modular architecture, ensuring scalability for future enhancements such as an admin dashboard or mobile application. It will provide a secure, responsive, and user-friendly experience across multiple devices.

1.5 References

- IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications.
- Official Node.js Documentation: `https://nodejs.org/en/docs/`
- MySQL Documentation: `https://dev.mysql.com/doc/`
- Tailwind CSS Documentation: `https://tailwindcss.com/docs`

Chapter 2

Overall Description

2.1 User Classes and Characteristics

The platform caters to two main user groups—alumni and students. Each group has unique traits influencing their interaction with the site.

- **Alumni Users:** These are BUP graduates, often professionals or retirees with a solid understanding of technology. They can mentor students, plan events, or use the blood bank, valuing features that strengthen community ties and professional outreach.
- **Student Users:** They are current BUP students, typically younger with basic to moderate tech skills. They might use the platform daily for mentorship, event participation, live chats, or emergency support like needing blood, often on mobile devices, needing simple and accessible interfaces.

2.2 User Needs

The platform addresses the essential requirements of building a connected BUP community, offering tools for interaction and support tailored to each user group.

- Alumni need options to guide students through mentorship, organize and join virtual events, maintain profiles, and assist with blood donations to stay engaged with their university community.
- Students require access to alumni for advice, mentorship opportunities, real-time communication, event registration, and blood bank services for emergencies, aiding their academic and personal development.
- All users need a secure, user-friendly system for managing profiles, communicating, and accessing resources, replicating a real-world alumni network to enhance community involvement.

2.3 Operating Environment

The BUP Alumni platform operates as a web application accessible via browsers on various devices, supported by a client-server setup.

- **Client Side:** Accessible on modern browsers across desktops, tablets, and mobiles. Devices need a reliable internet connection, with JavaScript enabled. Responsive design with Flexbox and media queries ensures compatibility.

- **Server Side:** Runs on Node.js with Express.js on a standard web server, connected to a MySQL database. Configuration is managed via .env files for settings like database credentials.
- **Network and Integrations:** Uses HTTPS for security, with Fetch API for dynamic updates. No specialized hardware is required beyond typical web hosting.

2.4 Constraints

The development and operation of the platform are bound by several limitations to ensure security, scalability, and practicality:

- Technical Constraints: Relies on open-source tools like HTML, CSS, JavaScript, Node.js, Express.js, and MySQL. File uploads are limited. Security features include session-based authentication with bcrypt and httpOnly cookies.
- Regulatory and Resource Constraints: Adheres to basic privacy standards (e.g., hashed passwords) and assumes a single-developer setup with no specified budget for premium hosting or tools.
- Mobile App Limitation: The platform is currently web-only and not developed as a mobile application, restricting native mobile functionality.

2.5 Assumptions

The planning assumes certain conditions that, if incorrect, may necessitate adjustments:

- Users have consistent internet access and basic web navigation skills (e.g., form filling, link clicking).
- Hosting provides adequate storage for files and database growth, with no disruptions from external services.
- Database schemas (e.g., users and events tables) remain stable, with SQL scripts executing correctly.
- Future integrations will be feasible without major changes.

Chapter 3

External Interface Requirements

3.1 User Interfaces

The platform is designed with a simple and user-friendly interface so that both alumni and students can easily interact with the system. The design focuses on clarity, consistency, and responsiveness across devices like desktops, tablets, and mobiles.

- **Homepage:** The homepage includes a navigation bar, a dynamic image slider with photos of the university, and short descriptions about the platform's purpose. From here, users can quickly move to important sections like login, registration, events, or mentorship.
- **Authentication Pages:**
  - **Signup Page:** Registration form with fields for name, email, password, role (alumni/student).
  - **Login Page:** Simple login form with username and password fields.
- **Profile Management:**
  - **Edit Profile Page:** Users can edit their information through a profile editing form. Clear error messages and form validation are included to help users correct mistakes.
- **Event and Mentorship Pages:**
  - **Events Pages:** Lists upcoming events with modal pop-ups for details and registration.
  - **Mentorship Pages:** Provide mentor matching features and descriptions.
- **Community Services:**
  - **BloodBank Pages:** Display donor information and emergency contact details in a structured layout.
  - **Contact Page:** Inquiry form to send messages directly to administrators.
- **Additional Interfaces:**
  - **About Page:** Provides an overview of the BUP Alumni Association.
  - **Statistics Page:** Displays data visualizations of alumni growth and participation.
  - **Alumni Details Page:** Displays data about all the signed up alumni such as name, department, blood group, socials etc.

3.2 Hardware Interfaces

The BUP Alumni platform operates as a web application and does not require direct hardware interfaces for core functionality. It is designed to run on standard web servers capable of hosting Node.js and MySQL, with user interactions occurring via client devices (desktops, tablets, and mobile phones). No specific physical hardware components are integrated, and all interactions are mediated through standard web browsers. Future scalability may consider hardware for enhanced features but this is not currently implemented.

3.3 Software Interfaces

The platform connects with various software components to ensure a unified system:

- **Operating Systems:** Designed to work on different platforms, supporting Windows, macOS, and Linux through browser-based access.
- **Database:** MySQL serves as the relational database management system, storing user credentials, profile details, event schedules, and blood bank data.
- **Libraries and Tools:** Integrates Node.js with Express.js for RESTful API endpoints, Multer for secure file uploads, and bcrypt for password hashing. JavaScript libraries like Tailwind CSS and Font Awesome (via CDN) enhance styling and interactivity.
- **External Components:** No third-party software integrations are currently implemented, though future plans include potential connections for payment processing or social media pending development. Additionally, the platform now uses a third-party API, Formspree, for handling form submissions.

3.4 Communications Interfaces

The system relies on standard and secure methods for communication between the user's browser, the server, and the database.

- **Client and Server:** The web browser communicates with the Node.js server through HTTP/HTTPS. Data is exchanged using RESTful APIs, and the Fetch API in JavaScript allows parts of the page to update without reloading.
- **Authentication:** When users log in, their session is managed through secure cookies. Passwords are hashed with bcrypt before being stored in the database, so sensitive information is never saved in plain text.
- **Database:** The server connects to a MySQL database, which stores user details, events, and blood bank information.
- **External Communication:** For features like password recovery or notifications, the system may send emails through a third-party service such as Nodemailer.

Chapter 4

System Features

4.1 Sign Up

Frontend User Stories

- As a user, I need to choose "I am a Student" or "I am an Alumni" before signup, so that I get the correct signup form.
- As a user, I need to see the signup form with Student ID (mandatory for both Student and Alumni), so that the system can verify my identity.
- As a user, I need to enter my Name, Department, Batch, BUP ID, Email, Phone, Gender, Blood Group, Address, LinkedIn, Facebook, and Password, so that my profile is complete.
- As a user, I need to upload my profile photo (optional), so that others can recognize me.
- As a student, I need to enter my Student ID, so that the system can verify I am an active student.
- As a student, I need to provide my current year/semester, so that my academic status is clear.
- As an alumni, I need to enter my Student ID, so that the system can verify I am a graduate.
- As an alumni, I need to provide my graduation year, so that others can see when I completed my studies.
- As an alumni, I need to indicate my alumni status ("Graduate", "Postgraduate"), so that my education level is clear.
- As an alumni, I need to add my current workplace and designation (optional), so that students and peers know my professional background.
- As a user, I need to see validation errors (required fields missing, invalid email/phone), so that I can correct mistakes.
- As a user, I need to receive OTP after filling the form, so that I can verify my account.
- As a user, I need a resend OTP option if I didn't receive it, so that I can finish signup.
- As a user, I need to see a success message after submitting my signup form, so that I feel assured.

Backend User Stories

- As a system, I need to validate whether the user selected Student or Alumni, so that the correct signup process applies.
- As a system, I need to check that BUP ID (Student/Alumni) and Email are unique, so that duplicate accounts are not created.
- As a system, I need to cross-check the Student ID format, so that fake IDs cannot be used.
- As a system, I need to validate required fields (Name, Department, Batch, Email, Phone, Password), so that incomplete forms are rejected.
- As a system, I need to generate and send OTP in the form, so that only verified users can signup.
- As a system, I need to hash and securely store passwords, so that credentials are protected.
- As a system, I need to store the account type (Student or Alumni), so that the correct access rules apply.
- As a backend system, I need to store student-alumni specific details in the database, so that they can be used later for filtering.
- As a system, I need to send a welcome message after successful signup, so that users are informed.

4.2 Log In

Frontend User Stories

- As a user, I need to choose "Login as Student" or "Login as Alumni", so that the correct login form appears.
- As a user, I need to login with Student ID and password, so that I can access my account.
- As a user, I need to see error messages for wrong credentials, so that I know what went wrong.
- As a user, I need to use the "Remember Me" option, so that I don't need to login every time.
- As a user, I need to reset my password if I forget it, so that I can regain access.
- As a user, I need to receive an email for password reset, so that I can verify my account.
- As a user, I need to enter a new password and confirm it, so that I can safely reset my login.
- As a user (student or alumni), I need to be redirected to my respective dashboard after login, so that I can access role-specific features.

Backend User Stories

- As a system, I need to validate user credentials against the database, so that only authorized users can log in.
- As a system, I need to check the hashed password during login, so that user credentials remain secure.
- As a system, I need to return clear error messages for invalid email/username or password, so that users know why login failed.
- As a system, I need to generate a secure session after successful login, so that user requests can be authenticated.
- As a system, I need to identify the user role (student or alumni) after login, so that the correct dashboard can be displayed.
- As a system, I need to automatically expire inactive sessions, so that user accounts remain secure.
- As a system, I need to allow password reset via Email, so that users can securely recover their account.

4.3 User Profile Management

Frontend User Stories

- As a registered user I need to view all my personal details (name, ID, department, email, phone, social links, etc.) in a structured profile page so that I can have a clear summary of my information in one place.
- As a registered user I need to edit my profile information (phone, address, social media links, etc.) so that I can update incorrect or outdated details.
- As a registered user I need to upload or change my profile picture so that I can personalize my account.
- As a registered user I need a clear success message after saving changes so that I know my updates were applied correctly.
- As a registered user I need clickable social media links in my profile so that I can connect with others directly from the portal.

Backend User Stories

- As a system, I must validate that all mandatory profile fields (e.g., name, email) are correctly filled in, so that incomplete or invalid records are not stored.
- As a system, I must check that the uploaded image follows allowed file types (e.g., .jpg, .png) and does not exceed size limits, so that storage remains secure and efficient.
- As a system, I must store updated profile information in the database and replace previous data, so that the user's latest details are always available.
- As a system, I must securely store uploaded images in a protected directory and link them to the user's ID, so that profile pictures are displayed correctly.
- As a system, I must ensure the user is authenticated before any update is processed, so that unauthorized edits cannot occur.
- As a system, I must generate specific error responses (e.g., "Invalid file format") when validation fails, so that the frontend can present these to the user.

4.4 Events

Frontend User Stories

- As a user, I want to view a list of upcoming events so that I can decide which ones to attend.
- As a user, I want to click on an event to view its details.
- As a user, I want to register for an event so that I can participate.
- As a user, I want to filter or sort events by category so that I can find what interests me.
- As a user, I want to see past events with photos or videos so that I can stay connected even if I missed them.
- As an admin, I want to create, edit, and cancel events so that the information stays up to date.

Backend User Stories

- As a system, I want to fetch event data from the database for display.
- As a system, I want to record event registrations in the database.
- As a system, I want to prevent duplicate registrations by the same user.
- As a system, I want to store feedback after events so future ones can be improved.

4.5 Alumni Details

Frontend User Stories

- As a student, I need to view alumni profiles (name, department, batch, blood group, email, LinkedIn), so that I can connect with seniors.
- As an alumni, I need my profile details to be visible to others, so that students and peers can reach me.
- As a user, I need to search alumni by name, so that I can quickly find someone.
- As a user, I need a filter option for department, so that I can see alumni only from my department.
- As a user, I need a filter option for batch, so that I can find seniors or juniors from a specific year.
- As a user, I need to filter alumni by location/address, so that I can connect with people living nearby.
- As a user, I need to see alumni profile pictures if uploaded, so that I can recognize them.
- As a user, I need alumni details to be clickable, so that I can view their full profile in detail.
- As a student, I need LinkedIn/Facebook links to open directly in a new tab, so that I can easily connect.
- As a user, I need to see only verified alumni profiles, so that I can trust the information.
- As a user, I need to see a "No Alumni Found" message if no alumni match my search/filter, so that I know clearly.
- As a user, I need to bookmark or favorite certain alumni profiles, so that I can quickly access them later.

Backend User Stories

- As a system, I need to fetch all alumni data from the database, so that it can be displayed on the frontend.
- As a system, I need to support search queries, so that the frontend can provide search results.
- As a system, I need to support filter queries (department, batch, location), so that users can refine results.
- As a system, I need to ensure only verified alumni data is returned, so that unapproved users are hidden.
- As a system, I need to sanitize alumni data (hide passwords, sensitive info), so that only safe fields are exposed.
- As a system, I need to handle empty search/filter queries gracefully, so that the frontend can show "No Alumni Found."
- As a system, I need to allow alumni to update their profile information, so that student users see up-to-date data.

- As a system, I need to track which alumni profiles are most viewed or bookmarked, so that popular connections can be analyzed.
- As a system, I need to restrict access to alumni details to only registered users (students or alumni), so that outsiders cannot misuse the data.
- As a system, I need to paginate alumni list results, so that the page remains responsive and fast.

4.6 Mentorship for Alumni

4.6.1 Creating a New Mentorship Offer

Frontend User Stories

- As an alumni user, I want to fill out a form on the dashboard with title, description, expertise areas, availability, and contact preference so that I can create a new mentorship offer.
- As an alumni user, I want to submit the form by clicking "Create Mentorship Offer" so that my offer is added to the system.
- As an alumni user, I want to see a success message after submission so that I know the offer was created successfully.
- As an alumni user, I want to see validation errors (e.g., for missing fields) on the form so that I can correct my input.

Backend User Stories (with multiple cases)

- As a system, I want to validate that all required fields (title, description, expertise areas, availability, contact preference) are provided so that incomplete offers are not created.
- As a system, I want to check that the expertise areas are properly formatted (comma-separated) so that they can be stored and searched correctly.
- As a system, I want to store the new mentorship offer in the database linked to the alumni's user ID so that it can be retrieved for display and applications.
- As a system, I want to return a success response to update the "My Mentorship Offers" table in real-time so that the alumni sees their new offer immediately.
- As a system, I want to handle duplicate offers by checking if a similar offer (e.g., same title and description) already exists for the alumni so that redundancy is avoided.
- As a system, I want to log the creation event for auditing so that administrators can track offer activity.
- As a system, I want to return specific error messages (e.g., "Title is required") if validation fails so that the frontend can display them to the user.
- As a system, I want to ensure the alumni is authenticated and has verified alumni status before allowing offer creation so that only eligible users can proceed.

4.6.2 Viewing and Managing My Mentorship Offers

Frontend User Stories

- As an alumni user, I want to view a table of my mentorship offers on the dashboard, including title, description, expertise areas, availability, contact, created at, and actions (edit/delete) so that I can review my offers.
- As an alumni user, I want to click "Edit" on an offer to pre-fill the form with existing details so that I can update it easily.
- As an alumni user, I want to click "Delete" on an offer and confirm the action so that I can remove unwanted offers.
- As an alumni user, I want the table to refresh automatically after edits or deletions so that I see the updated list.

Backend User Stories (with multiple cases)

- As a system, I want to retrieve all mentorship offers associated with the alumni's user ID from the database so that the "My Mentorship Offers" table can be populated.
- As a system, I want to update an existing offer in the database when edits are submitted so that changes (e.g., updated availability) are persisted.
- As a system, I want to soft-delete an offer (mark as inactive) when the delete action is triggered so that it can be recovered if needed, while removing it from active views.
- As a system, I want to check for pending applications before allowing deletion so that offers with active students are not removed accidentally.
- As a system, I want to return filtered data (e.g., only active offers) to prevent displaying deleted or archived items.
- As a system, I want to log edit and delete actions with timestamps so that changes can be audited.
- As a system, I want to handle concurrency by locking the offer record during edits so that multiple simultaneous updates do not cause data conflicts.
- As a system, I want to return an error if the offer ID does not belong to the current alumni so that unauthorized modifications are prevented.

4.6.3 Managing Mentorship Applications

Frontend User Stories

- As an alumni user, I want to view a table of student applications on the dashboard, including offer title, student name, department, message, status, applied at, and actions (approve/reject) so that I can review incoming requests.
- As an alumni user, I want to click "Approve" on an application to move it to a scheduled status so that I can proceed to session scheduling.
- As an alumni user, I want to click "Reject" on an application and optionally provide a reason so that the student is notified.
- As an alumni user, I want to see updated status (e.g., "pending" to "session$_s cheduled$") $in the table after actions so that I co$

Backend User Stories (with multiple cases)

- As a system, I want to retrieve applications linked to the alumni's offers from the database so that the "Manage Mentorship Applications" table can be displayed.
- As a system, I want to update the application status to "approved" and trigger session scheduling options when approve is clicked so that the process advances.
- As a system, I want to update the status to "rejected" and store any rejection reason when reject is clicked so that the student can be informed.
- As a system, I want to send email notifications to the student upon approval or rejection so that they receive timely updates.
- As a system, I want to check if the application is still pending before allowing actions so that already processed applications cannot be modified.
- As a system, I want to limit actions to the offer owner (alumni) so that only authorized users can approve/reject.
- As a system, I want to log all application actions (approve/reject) with details for compliance and dispute resolution.
- As a system, I want to handle bulk actions if multiple applications exist, but prevent approving more than available slots if limits are set.
- As a system, I want to return an error if the student is ineligible (e.g., wrong department) during approval so that invalid applications are caught.

4.6.4 Scheduling and Viewing Mentorship Sessions

Frontend User Stories

- As an alumni user, I want to schedule a session after approving an application by selecting date, time, and generating a meeting link so that the session is set up.
- As an alumni user, I want to view a table of my mentorship sessions, including offer title, student name, date, time, meeting link, and status so that I can manage upcoming meetings.
- As an alumni user, I want to click "Join Session" on a scheduled session to access the meeting link so that I can participate easily.
- As an alumni user, I want to see status updates (e.g., "scheduled") in the table so that I know the session's state.

Backend User Stories (with multiple cases)

- As a system, I want to create a new session record in the database with date, time, and generated meeting link upon scheduling so that details are stored securely.
- As a system, I want to retrieve all sessions linked to the alumni from the database so that the "My Mentorship Sessions" table can be populated.
- As a system, I want to validate the selected date and time against the alumni's availability so that conflicts are avoided.
- As a system, I want to integrate with a meeting service (e.g., Zoom API) to generate and store a unique meeting link so that sessions can be joined virtually.

- As a system, I want to send calendar invites or notifications to both alumni and student after scheduling so that they are reminded.
- As a system, I want to update the session status (e.g., from "scheduled" to "completed") based on events like session end time.
- As a system, I want to check for double-booking by querying existing sessions for overlapping times so that the alumni isn't overcommitted.
- As a system, I want to log session creation and access attempts for security and analytics.
- As a system, I want to return an error if scheduling fails (e.g., invalid date format or unavailable time) so that the frontend can prompt corrections.
- As a system, I want to ensure the session is only accessible to the approved student and alumni so that privacy is maintained.

4.7 Mentorship for Student

4.7.1 Browsing Available Mentorship Offers

Frontend User Stories

- As a student user, I want to view a list of available mentorship offers on the page, displayed as cards with title, mentor name, department, expertise, and a brief description so that I can browse options relevant to my interests.
- As a student user, I want to scroll through the offers horizontally or vertically so that I can see all available ones easily.
- As a student user, I want to click "View Details" on an offer card to open a modal with full details so that I can learn more before applying.
- As a student user, I want the list to refresh or load more offers if there are many so that I can access all current opportunities.

Backend User Stories (with multiple cases)

- As a system, I want to retrieve all active mentorship offers from the database, including title, mentor details, department, expertise, and description so that the frontend can display them as cards.
- As a system, I want to filter offers based on student eligibility (e.g., matching department or open to all) so that only relevant offers are shown.
- As a system, I want to sort offers by recency or relevance (e.g., based on creation date) so that the most current ones appear first.
- As a system, I want to limit the initial load to a certain number of offers (e.g., 10) and support pagination for additional loads so that performance is optimized.
- As a system, I want to check if the student is authenticated and enrolled before returning offers so that only verified students can browse.
- As a system, I want to log browsing activity for analytics so that administrators can track popular offers.
- As a system, I want to return an empty list message if no offers are available so that the frontend can display "No offers found."
- As a system, I want to handle expired offers by excluding those past their availability date so that outdated ones are not shown.

4.7.2 Viewing Offer Details and Applying for Mentorship

Frontend User Stories

- As a student user, I want to see a modal popup with full offer details (title, description, expertise areas, availability, contact preference, email, LinkedIn) when I click "View Details" so that I can evaluate the offer thoroughly.
- As a student user, I want to enter a message in a field within the modal and click "Apply for Mentorship" so that I can submit my application.
- As a student user, I want to receive a confirmation message after applying so that I know my application was sent successfully.
- As a student user, I want to see validation errors (e.g., for empty message) in the modal so that I can correct my input before submitting.

Backend User Stories (with multiple cases)

- As a system, I want to fetch detailed offer data from the database by offer ID so that the modal can be populated with accurate information.
- As a system, I want to validate the student's application message for length and content (e.g., no spam) so that only meaningful applications are processed.
- As a system, I want to store the application in the database, linking it to the student, offer, and mentor, with initial status "pending" so that it can be reviewed.
- As a system, I want to send a notification (e.g., email) to the mentor about the new application so that they can respond promptly.
- As a system, I want to check if the student has already applied to this offer to prevent duplicates so that multiple submissions are blocked.
- As a system, I want to verify the offer is still active (not expired or full) before accepting the application so that invalid applies are rejected.
- As a system, I want to log the application event with timestamp and details for auditing so that disputes can be resolved.
- As a system, I want to return specific error messages (e.g., "Message is required" or "Offer no longer available") if submission fails so that the frontend can display them.
- As a system, I want to ensure the student meets any prerequisites (e.g., department match) before allowing application so that eligibility is enforced.

4.7.3 Viewing My Applications

Frontend User Stories

- As a student user, I want to click the "My Applications" tab under "My Mentorship Status" to view a table of my submitted applications, including offer title, mentor name, message, status (e.g., pending, $\text{session}_s cheduled), and applied at so that I can track progress.$

- As a student user, I want the table to update in real-time or on refresh if status changes (e.g., from pending to approved) so that I stay informed.
- As a student user, I want to see no actions needed on applications, just status viewing, so that the interface remains simple.
- As a student user, I want to filter or sort the table by status or date if there are many applications so that I can focus on pending ones.

Backend User Stories (with multiple cases)

- As a system, I want to retrieve all applications submitted by the student from the database, including linked offer and mentor details, so that the "My Applications" table can be populated.
- As a system, I want to update application status in the database when the mentor approves/rejects (e.g., to "session_scheduled") so that changes are reflected accurately.
- As a system, I want to sort applications by applied date descending so that the most recent appear first.
- As a system, I want to filter out completed or rejected applications if the student chooses, but default to showing all active ones.
- As a system, I want to check student authentication before returning data so that only the owner's applications are accessible.
- As a system, I want to handle large numbers of applications with pagination so that performance doesn't degrade.
- As a system, I want to log status views for security monitoring so that unusual activity can be detected.
- As a system, I want to return an empty table message if no applications exist so that the frontend can display "No applications yet."

4.7.4 Viewing My Scheduled Sessions

Frontend User Stories

- As a student user, I want to click the "My Sessions" tab under "My Mentorship Status" to view a table of scheduled sessions, including offer title, mentor name, date, time, meeting link, and status (e.g., scheduled) so that I can prepare for meetings.
- As a student user, I want to click "Join Session" on a scheduled session to open the meeting link so that I can participate easily.
- As a student user, I want the table to highlight upcoming sessions (e.g., based on current date) so that I don't miss them.
- As a student user, I want to see status updates (e.g., from scheduled to completed) automatically so that past sessions are archived.

Backend User Stories (with multiple cases)

- As a system, I want to retrieve all scheduled sessions for the student from the database, including date, time, and meeting link, so that the "My Sessions" table can be displayed.
- As a system, I want to generate or retrieve a secure meeting link (e.g., via integration with Zoom) when a session is scheduled so that it's ready for joining.
- As a system, I want to validate the session date and time against the current date (September 12, 2025) to mark upcoming, ongoing, or past sessions.
- As a system, I want to update session status automatically (e.g., to "completed" after the end time) so that the table reflects reality.
- As a system, I want to send reminders (e.g., email) to the student before the session based on the scheduled time so that attendance is improved.
- As a system, I want to check for conflicts (e.g., overlapping sessions) when scheduling and prevent them so that the student isn't double-booked.

- As a system, I want to log join attempts for the meeting link to track participation and security.
- As a system, I want to ensure only approved applications lead to sessions by cross-referencing with application status.
- As a system, I want to return an error if a session is canceled by the mentor so that the frontend can remove it from the table.
- As a system, I want to restrict access to session details to the involved student and mentor so that privacy is maintained.

4.8 Blood Bank

Frontend User Stories

- As a user, I want to view available donors so that I can find help in emergencies.
- As a donor, I want to register as a blood donor so that others can reach me.
- As a user, I want to search or filter donors by location, blood group, and availability so that I can quickly find the nearest match.
- As a donor, I want to set my availability status so that others know when I can help.
- As a user, I want to contact a donor through the system so that I can reach them safely.

Backend User Stories

- As a system, I want to store donor information in the database.
- As a system, I want to fetch and display donors by blood group.
- As a system, I want to allow donors to update or remove their availability.
- As a system, I want to notify donors when someone requests their blood group.

4.9 Contact Us

Frontend User Stories

- As a registered user, I should see an embedded map of the university location on the page so that I can easily find the campus.
- As a registered user I need to view the university's official contact information (address, phone, email, website) so that I can directly reach the institution if required.
- As a registered user, I should be able to give my feedback and submit it, so that I can communicate my thoughts or concerns.
- As a registered user, I should be shown an error message if I attempt to submit the form without writing any feedback, so that I know to provide content before submission.

Backend User Stories

- As a system, I must ensure that the submitted feedback field is not empty, so that blank messages are not processed.
- As a system I need to send the feedback content directly to the admin's email address so that the query reaches the responsible authority.
- As a system, I must send the submitted feedback as an email to the designated admin account, so that it reaches the intended recipient.
- As a system, I must handle and report delivery failures (e.g., mail server not available), so that the user's attempt is not lost silently.
- As a system, I must record each feedback submission in the database, so that administrators can track history of communication.

Chapter 5

Other Nonfunctional Requirements

5.1 Performance Requirements

- The system shall support at least 100 concurrent users without noticeable degradation in performance.
- All pages shall load within 3 seconds on a standard broadband connection.
- The server shall respond to database queries (e.g., profile retrieval, event listings) within 2 seconds under normal load.
- The chat feature shall deliver messages with a latency of less than 1 second in real-time communication.
- The platform shall be optimized for both desktop and mobile devices to ensure consistent responsiveness.

5.2 Safety Requirements

- The system shall provide regular database backups to prevent data loss in case of hardware or software failure.
- Critical errors shall be logged, and the system shall recover gracefully without exposing sensitive information.
- In the event of server downtime, users shall be shown a friendly error page rather than system error details.
- Uploaded files (e.g., profile pictures) shall be scanned or validated to prevent malicious file execution.

5.3 Security Requirements

- All passwords shall be stored in hashed form using bcrypt and never in plain text.
- The system shall use HTTPS for secure data transmission to protect against eavesdropping and man-in-the-middle attacks.
- User sessions shall be managed with secure, HTTP-only cookies and automatically expire after inactivity.
- The system shall implement input validation and prepared statements to prevent SQL injection and cross-site scripting (XSS).
- Only authenticated users shall have access to restricted features such as profile management, event registration, and mentorship applications.

5.4 Software Quality Attributes

- **Reliability:** The system shall maintain 99% uptime excluding scheduled maintenance.
- **Usability:** The interface shall be intuitive and consistent, requiring minimal training for new users.
- **Maintainability:** The codebase shall be modular and well-documented, allowing future developers to make changes easily.
- **Portability:** The application shall run on major browsers (Chrome, Firefox, Edge, Safari) without compatibility issues.
- **Scalability:** The architecture shall support future expansion, such as the addition of an admin panel or mobile app.

5.5 Business Rules

- Only verified alumni and students of BUP may register and use the platform.
- Alumni may create mentorship offers, but only students can apply to them.
- A user must be logged in to access community services such as live chat and blood bank details.
- Donors must provide valid contact details before being listed in the blood bank.
- Event registrations shall be unique per user per event to avoid duplicate bookings.