International Journal of Scientific Research & Engineering Trends
Volume 12, Issue 6, Nov-Dec-2026, ISSN (Online): 2395-566X

Campus Connect: A Student Networking and Messaging Platform

Abhinav Singh [1] Shubham Kumar[2] Vishal[3] Mr. Sachin Chauhan[4]
Dept. Of Computer Science & Engineering
Hmr Institute Of Technology & Management
Delhi, India

Abstract- Students on a college campus share courses, laboratories, clubs and events, yet they often struggle to find peers whose skills and interests match their own. Opportunities to form project teams, find study partners or learn from seniors depend on notice boards, group chats and personal contacts, none of which record skills in a structured way or let a student start a conversation with someone outside their immediate circle. This paper presents Campus Connect, a student networking and messaging platform developed as a minor project in the Department of Computer Science and Engineering at HMR Institute of Technology & Management. The platform lets students create profiles, discover peers, send and accept connection requests, and exchange messages in a single system. The paper describes the motivation and requirements, the system architecture and MongoDB data model, the implementation of the core flows, and a testing approach that uses two independent user accounts to verify connection and messaging behaviour between real participants. It also records the parts of the system that remain as leftovers from earlier design decisions, and it outlines planned extensions, including semantic skill matching with Google Gemini and an AI-assisted team builder for student hackathons. The work is presented with an honest account of what is implemented and what is planned.

Keywords: campus networking, student platform, MongoDB, messaging, connection management, functional testing.

I. INTRODUCTION

A university campus is a dense network of people. Students attend the same lectures, share the same laboratories, compete in the same events and often work toward similar careers. Despite this proximity, most students find it difficult to locate the right peer at the right moment. A student who wants a teammate for a web project may not know who in the department has front-end experience. A first-year student who needs advice on a subject may not know which seniors have taken it. A student who is preparing for a hackathon may struggle to assemble a team with a balanced set of skills. Information about availability, expertise and interest is scattered across social media platforms, messaging groups and personal contacts, and much of it is lost as soon as the conversation moves on.

Campus Connect was conceived to address this gap. The aim is to give each student a structured profile that records their academic background and skills, a discovery mechanism through which they can find others, a connection process that establishes mutual consent, and a messaging channel in which connected students can talk without leaving the platform. By bringing these elements into one system, the project reduces the friction between finding a peer and starting a conversation with them.

The project was undertaken as a minor project for the Bachelor of Technology programme in Computer Science and Engineering, in accordance with the guidelines for course ES-451. The team consisted of three students: Abhinav Singh, Shubham Kumar and Vishal. The work was supervised by Mr. Sachin Chauhan, Assistant Professor, and coordinated by Dr. Gurminder Kaur, who also serves as a co-author of this paper in her role as Project Coordinator.This paper has three purposes.

The first is to describe the problem and the requirements that shaped the system. The second is to explain the design and implementation, including the architecture, the data model and the main flows, illustrated with diagrams. The third is to report on how the system was tested and to reflect honestly on its limitations and on the future extensions that the team has identified. \

The paper concentrates on the work carried out by the team rather than on general background, and it distinguishes clearly between implemented features and planned features.The remainder of the paper is organised as follows. Section 2 reviews the background and related ideas. Section 3 states the problem and the objectives.

Section 4 sets out the requirements. Section 5 describes the system design, and Section 6 covers the implementation. Section 7 presents the testing strategy and its results, Section 8 discusses limitations, Section 9 outlines future work, and Section 10 concludes the paper.

© 2026 IJSRET

II. BACKGROUND AND RELATED IDEAS

Campus networking sits at the intersection of two established categories of software: social networking platforms, which help people build and maintain relationships, and collaboration tools, which help people work together on defined tasks. General-purpose social networks are designed for broad audiences and tend to emphasise content sharing over structured skill discovery. Collaboration tools, by contrast, assume that the team already exists and focus on coordinating its work. Campus Connect occupies the space between these two: it is organised around the people of one institution, and it treats skills and interests as first-class data so that students can find one another for a purpose.

Two design ideas from this space shaped the project. The first is that a connection should be mutual. A request that one student sends to another should be accepted before the two can exchange messages. This protects students from unsolicited contact and gives both sides control over who can reach them. The second is that discovery should be grounded in profile data. A student browsing the platform should be able to see the skills and interests that others have recorded, which makes it possible to judge whether a person is relevant to a particular need.

The team also considered the role of artificial intelligence. Modern language models can read free-form descriptions of skills and projects and identify related competencies that exact keyword matching would miss. The team decided that such semantic matching would be a valuable extension, but that the core platform should first provide a reliable structure for profiles, connections and messages. This decision reflects a principle that guided the whole project: build the foundation correctly before adding intelligence on top of it.

From a software engineering perspective, the project draws on the familiar practice of defining requirements, designing a system around them, implementing it in increments, and testing it against the requirements. The team treated the two-account test not as an afterthought but as a central part of the development process, because the most important behaviours of the platform arise from the interaction between two users rather than from either user alone.

1. Comparison Of Existing Approaches

Students currently use several general tools to find peers and communicate. Table Table 1 compares these approaches with the goals of Campus Connect. The comparison is based on the features each category typically offers and is intended to show the gap that the project addresses, not to rank commercial products.

Table 1: Comparison of existing approaches for finding campus peers

Approach	Strength	Limitation for campus discovery	Campus Connect response
Notice boards and printed notices	Official and visible to all	One-way; no profiles or replies	Profiles and two-way connections
Class and club group chats	Fast, informal communication	Unstructured; skills are not recorded	Structured skill and interest fields
General social networks	Large audience; familiar interface	Not scoped to one institution; mixed purpose	Scoped to campus students only
Professional networks	Good for career profiles	Oriented to employment; less suited to peer study	Academic peer focus
Personal contacts	Trusted introductions	Limited reach; depends on who one already knows	Discovery beyond personal circles

III. PROBLEM STATEMENT AND OBJECTIVES

1. Problem Statement

Students who wish to form project teams, find study partners or seek guidance currently rely on informal means that do not scale and do not preserve information. The problem has two linked parts. The first is a discovery problem: students cannot easily see who is available and what they are able to do. The second is a communication problem: students who do find one another must move to a separate application to talk, which breaks the flow of the interaction and loses context. A solution must address both parts within a single, consistent system.

2. Objectives

The team set the following objectives at the start of the project:

- Provide each student with a profile that records identity, academic background, skills and interests.

© 2026 IJSRET

- Allow students to browse and discover other students on the platform.
- Implement a connection process in which a request must be accepted before messaging is possible.
- Provide a messaging channel between connected students, with a stored conversation history.
- Store all application data in a MongoDB database using a clear and maintainable schema.
- Verify core behaviour through repeatable tests that use two independent user accounts.

Advanced features, including intelligent matching and team formation, were deliberately kept outside the initial scope and recorded as future work. This decision allowed the team to complete, test and document the core system properly.

3. Stakeholders

The platform has several groups of stakeholders. Table Table 2 lists them together with the main interest each group has in the system. Identifying stakeholders early helped the team check that the requirements served the people who would actually use the platform.

Table 2: Stakeholders of the Campus Connect project

Stakeholder	Role	Main interest
Students (users)	Create profiles, find and connect with peers, message connections	Easy discovery; control over contact; reliable messaging
Project team	Design, build and test the platform	A working, documented system within the course timeline
Project supervisor	Guides technical direction and assesses the report	Correct design decisions; honest reporting of scope
Project coordinator	Coordinates the minor project programme	Compliance with ES-451 guidelines and documentation standards
Department faculty	Evaluate the outcome of the minor project	Clear evidence of engineering practice and learning

4. Scope, Assumptions And Constraints

The scope of the first version is limited to the core student flows described in Section 3.2. Several assumptions and constraints shaped the work. These are listed in Table Table 3 so that a reader can judge which conclusions apply and which do not.

Table 3: Scope, assumptions and constraints

Category	Item	Effect on the project
In scope	Profiles, discovery, connections, messaging, persistence in MongoDB	Forms the core deliverable and the basis of testing
Out of scope	Semantic matching, team formation, production hosting	Recorded as future work; not claimed as delivered
Assumption	Students enter their own profile information	Data accuracy depends on students; no external verification
Assumption	Users access the platform from a standard web browser	Interface is designed for common desktop and mobile browsers
Constraint	Academic timeline of one minor project	Incremental delivery; features prioritised by dependency
Constraint	Student team with limited production experience	Emphasis on simple, testable design over scale

IV. REQUIREMENTS

The requirements were gathered through discussion among the team members and with the supervisor, and were refined as the design progressed. They are divided into functional requirements, which describe what the system must do, and non-functional requirements, which describe the qualities it must have.

1. Functional Requirements

Table 4: Functional requirements of Campus Connect

ID	Requirement	Priority
FR-1	A student can create and edit a profile with identity, academic details, skills and interests.	High
FR-2	A student can browse and view the profiles of other students.	High
FR-3	A student can send a connection request to another student.	High

© 2026 IJSRET

ID	Requirement	Priority
FR-4	A recipient can accept or decline a connection request.	High
FR-5	Only connected students can exchange messages.	High
FR-6	Messages are stored and the conversation history can be reloaded.	High
FR-7	Application data is persisted in MongoDB.	High
FR-8	Core flows can be verified by an automated two-account test.	Medium

2. Non-Functional Requirements

- Privacy: a student should see only the profiles and messages that their connections and their own activity make visible to them.
- Consistency: a connection status shown to one student must agree with the status shown to the other.
- Maintainability: the data model and code should be organised so that new features can be added without restructuring the core.
- Usability: the main actions, namely creating a profile, finding a peer, connecting and messaging, should be reachable in a few steps.
- Testability: the core flows should be verifiable by repeatable scripts rather than only by manual checking.

3. Use Case Model

The use case model describes the interactions between students and the platform. Figure Figure 1 presents the use cases, and Table Table 5 describes the main ones in more detail. Each use case corresponds to one or more of the functional requirements in Table Table 4.

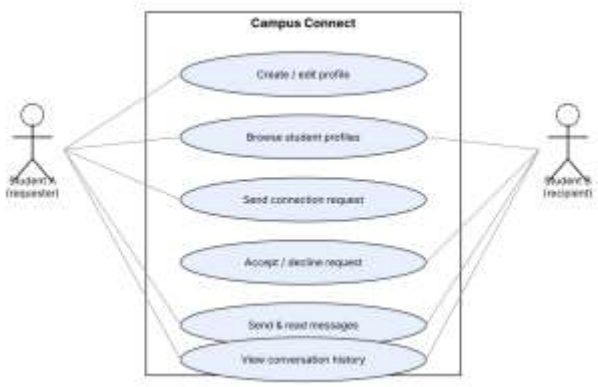

Figure 1: Use case diagram for students on the platform

Table 5: Summary of principal use cases

Use case	Primary actor	Precondition	Main success outcome
UC-1 Create or edit profile	Student	Student is logged in	Profile saved and shown in discovery
UC-2 Browse profiles	Student	At least one other profile exists	Chosen profile details are displayed
UC-3 Send connection request	Student (requester)	Target profile is viewable and no active request exists	Connection stored as pending
UC-4 Respond to request	Student (recipient)	A pending request is addressed to the student	Connection becomes connected or is closed
UC-5 Send and read messages	Connected students	Connection is in the connected state	Message stored and visible to both participants
UC-6 View conversation history	Connected students	A conversation exists	Earlier messages reload in order

4. Risk Assessment

Before development began, the team listed the risks that could threaten the project and the way each would be handled. Table Table 6 summarises the assessment. The table was reviewed during the project and remains a useful record of how the team reasoned about uncertainty.

Table 6: Risk assessment and mitigations

Risk	Likelihood	Impact	Mitigation adopted
Scope grows beyond the timeline	High	High	Features ordered by dependency; advanced features moved to future work
Inconsistent connection state between users	Medium	High	Server holds the authoritative state; two-account test

© 2026 IJSRET

Risk	Likelihood	Impact	Mitigation adopted
Team members working on conflicting changes	Medium	Medium	Single shared repository; regular review meetings
Loss of work or code	Low	High	Version control of the repository
Leftover code from earlier designs	High	Low	Leftovers recorded openly; cleanup listed as planned work
Exposure of personal data in testing	Low	High	Test runs use separate development records, not real student data

5. Development Methodology

The team adopted an incremental development methodology, in which the system is built in small, complete slices and each slice is checked before the next is started. This approach is well suited to a student project with a fixed timeline, because it produces a working system at every stage rather than a large body of code that is only tested at the end. The increments were ordered by dependency: storage and the user model first, then profiles, then connections, and finally messaging, which depends on the connection rule. Each increment was reviewed by the team against the requirements in Table 4 before the next was begun.

Within each increment the team followed a simple cycle. A requirement was discussed and written down in plain language. The design for that requirement was sketched, including any data it would store and any rule it would enforce. The implementation followed, and the behaviour was checked first by hand and then, where possible, by the automated test. Any defect found was fixed and the test was rerun. This cycle of specify, design, build, check and refine kept the team's attention on the behaviour of the system rather than on the amount of code written.

The team also held regular informal reviews, at which members explained what they had built and asked the others to try the flows. These reviews were valuable for catching misunderstandings about the rules, such as whether a rejected request should allow a new request to be sent, which the team resolved by recording the decision in the connection lifecycle described in Section 5.3.

V. SYSTEM DESIGN

1.Architectural Overview

Campus Connect follows a layered organisation in which the student's browser presents the interface, an application server handles requests and applies the platform's rules, and a MongoDB database stores all persistent data. Within the application server, the functionality is divided into three modules that correspond to the main ideas of the system: profile and discovery, connection management, and messaging. Keeping these modules separate makes each one easier to reason about and to test. Figure 2 shows the architecture, including the test scripts, which act as a second client of the application server.

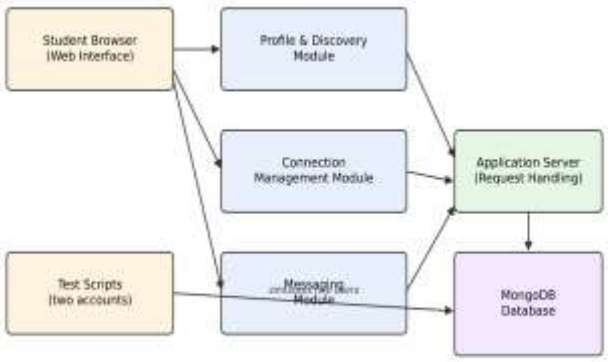

Figure 2: Campus Connect system architecture. The test scripts simulate two separate users against the same application server and database.

The separation between the browser and the server means that the rules governing connections and messages are enforced in one place. A student cannot bypass a rule by changing what the browser displays, because the server checks each request before it changes stored data. This is an important design choice for a platform that handles communication between students.

2. Data Model

The data model is built around four logical entities: users, connections, conversations and messages. Each user document holds the student's identity, academic information and lists of skills and interests. Skills and interests are stored as arrays because a student may record any number of them, and MongoDB's document model handles such arrays naturally. A connection document records the two students involved and the current status of the request. A conversation document records its participants and the time of the most recent message, and each message document refers to the

© 2026 IJSRET

conversation it belongs to. Figure 3 shows the relationships between these collections.

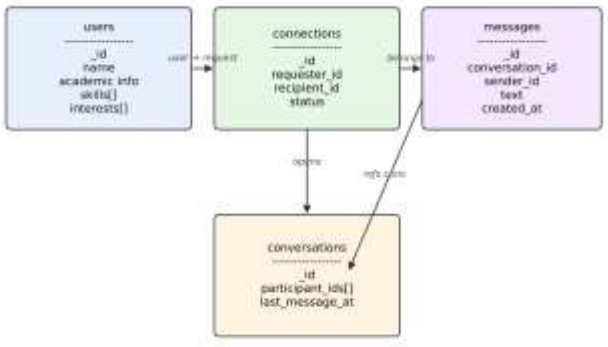

Figure 3: Logical data model. Users are linked to connections and conversations, and messages belong to a conversation.

The use of separate collections for connections, conversations and messages, rather than embedding everything inside the user document, was a deliberate decision. Embedding all messages in a user record would make that record grow without limit and would make it difficult to load a single conversation efficiently. References between collections keep each document a manageable size and allow a conversation to be retrieved directly.

3. Connection Lifecycle

The connection process is the gatekeeper for messaging, so its states were defined carefully before implementation. A connection begins with no relationship between two students. When one student sends a request, the connection enters a pending state. The recipient may then accept the request, which moves the connection to a connected state, or decline it, or the requester may withdraw it, which moves the connection to a rejected or withdrawn state. Only a connected pair can exchange messages. Figure 4 summarises these transitions.

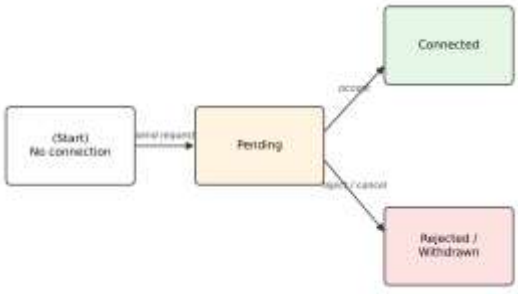

Figure 4: Connection request lifecycle. Messaging is permitted only in the Connected state.

Defining the states explicitly served two purposes. It gave the implementation a precise target, and it provided a basis for the tests. Each transition in Figure 4 corresponds to a behaviour that can be checked, and each illegal transition, such as attempting to message someone who has not accepted a request, is a case that the tests should refuse.

4. Messaging Design

Messaging is available only between students whose connection is in the Connected state. When a student sends a message, the server checks the connection, stores the message against the conversation and returns the saved record to the interface. Storing the message before returning it means that the sender and the recipient see the same persisted version of the conversation, which supports the consistency requirement. Conversation history can be reloaded when a student returns to the platform, so the exchange does not depend on the state of the browser session.

5. Authentication And Its Evolution

The project was initially planned to use Google sign-in for authentication. Over the course of development, the authentication flow was built around the application's own account handling. As a result, the Google sign-in code that remains in the repository is a leftover from the earlier design and is not part of the active user flow. The team has chosen to document this openly. An accurate report of the system as it actually works is more valuable than a description of a feature that is not active, and the leftover code is identified as a cleanup task in the limitations section.

6. Security And Privacy Considerations

A platform that connects students and carries their messages must protect their information and their control over who may contact them. The design addresses this in three ways. First, the connection rule is enforced by the server before any message is stored, so an interface change cannot be used to bypass it. Second, students see only the profiles that are available to them and only the conversations in which they are a participant. Third, the storage of messages is separated from the storage of profiles, so that the handling of conversation data can be reviewed on its own.

The team recognises that a production system would require further measures. These include the protection of credentials and session tokens, the validation of all input received from the browser, controls against automated abuse such as mass connection requests, and a clear policy on how long messages and profile data are retained. The present version is a student prototype and does not claim to meet all of these requirements. The team has recorded them as necessary steps before any wider deployment, and this list forms part of the planned work described in Section 9.

© 2026 IJSRET

Privacy considerations also extend to the data students enter. Profile information should be limited to what is needed for discovery, and students should be free to choose what they reveal. The simple profile design adopted by the team supports this aim, because it asks students for skills, interests and academic information rather than for sensitive personal data.

7. Interaction Sequence For Connections

Figure Figure 5 shows the sequence of messages between the two browsers, the application server and the database when one student sends a connection request and the other accepts it. The sequence makes explicit the duplicate check performed before a new record is created, which prevents two competing requests between the same pair of students.

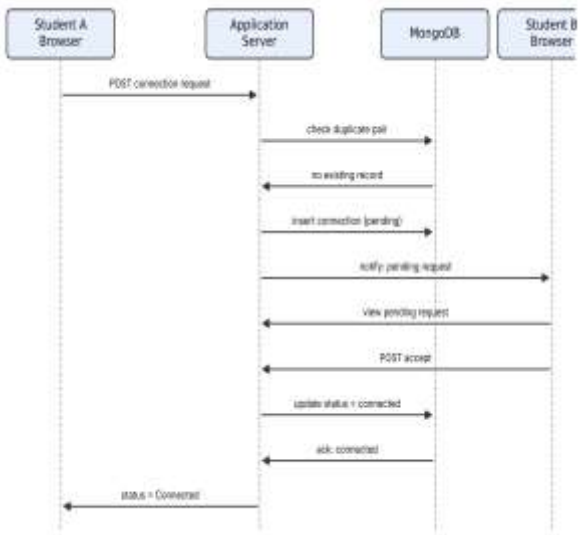

Figure 5: Sequence of a connection request and its acceptance

The diagram also shows that the application server is the only component that writes connection records. This keeps the rule in one place and means that a change to the interface cannot alter the connection state directly.

8. Deployment View

The deployment view in Figure Figure 6 shows where each component ran during development and demonstration. The browser runs on the student computer, the application server runs on a development or demonstration host, MongoDB runs on a separate database host, and the automated test runs from its own host so that it acts as an independent client.

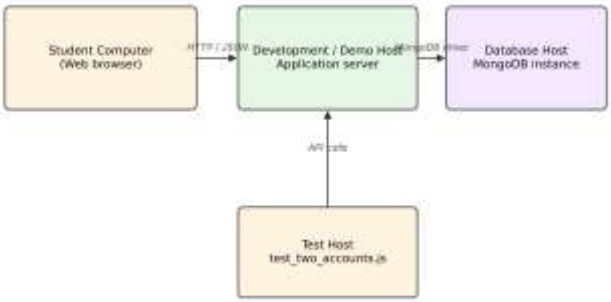

Figure 6: Deployment of Campus Connect components during development and demonstration

9. Interface Design

The interface was planned with simple wireframes before it was built. Figure Figure 7 shows the profile screen sketch. The sketch places the main actions where a student would expect to find them: the connection button appears on a profile that is not yet connected, and the message action appears only once the connection is in the connected state. This arrangement makes the connection rule visible to the user, so a student is not invited to message someone who has not accepted a request.

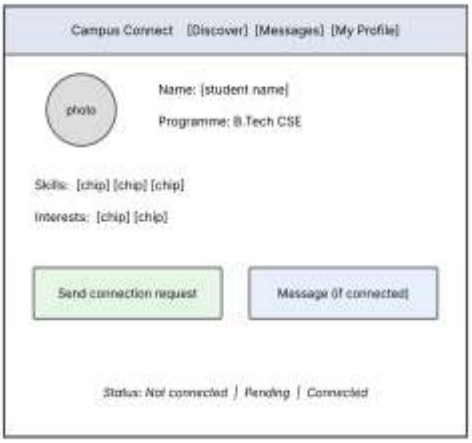

Figure 7: Wireframe of the profile screen (planning sketch)

The navigation was kept to three areas, discovery, messages and the student's own profile, so that the main activities of the platform remain no more than one step away. The team considered a more elaborate layout but decided that a small number of clearly labelled areas was easier to test and to explain.

© 2026 IJSRET

VI. IMPLEMENTATION

The implementation proceeded in increments that followed the requirements in Table 4. The team began with the data model and the storage of users, then implemented profile creation and browsing, then the connection lifecycle, and finally messaging. At each stage, the behaviour was checked against the relevant requirements before the next stage began. This incremental approach kept the system in a working state throughout the project and made it easier to locate the source of any fault.

1. Storage With Mongodb

MongoDB was selected as the database. Its document model suits profile data, where students may have different numbers of skills, interests and project entries, and it avoids forcing every profile into a rigid table layout. The team modelled users, connections, conversations and messages as separate collections with references between them. Indexes were considered for the fields used most often in queries, such as the identifiers of the two participants in a connection, so that looking up the status of a connection remains quick as the number of students grows.

2. Profiles And Discovery

Each student creates a profile that records their name, academic information and a list of skills and interests. Profiles are the basis for discovery. A student can browse the platform and open the profile of any other student, which is the first step toward forming a connection. The profile fields were kept deliberately simple in this version. Simple fields make it more likely that students will fill in their profiles accurately, and they keep the data consistent, which is necessary for any future matching feature.

3. Connections

A student who views another profile can send a connection request. The server records the request as pending and makes it visible to the recipient. When the recipient accepts, the server updates the connection to the connected state. Because the server, rather than the browser, holds the authoritative status, both students see the same state when they next load their connections. The team paid particular attention to preventing duplicate requests between the same pair of students, since duplicate records would make the connection state ambiguous.

4. Messaging

Once two students are connected, either may send messages to the other. The server verifies the connection before accepting each message. Messages are stored with the identifier of the sender, the text and the time of creation. The conversation is ordered by time so that the history reads naturally. The messaging module was implemented after the connection module so that its access rule could rely on the already-tested connection state.

5. Summary Of Implemented And Planned Features

Table 7: Status of Campus Connect features at the time of writing

Feature	Status	Notes
Student profile creation and editing	Implemented	Identity, academic details, skills and interests
Profile browsing and discovery	Implemented	Viewing other students' profiles
Connection requests and acceptance	Implemented	Pending, connected and rejected or withdrawn states
Messaging between connected students	Implemented	Stored conversation history
MongoDB persistence	Implemented	Separate collections with references
Two-account automated test	Implemented	Covers connection and messaging flow
Google sign-in	Leftover	Code remains from an earlier design; not active
Gemini semantic skill matching	Planned	Future work; not implemented
AI hackathon team builder	Planned	Future work; not implemented

6. Development Environment And Tools

The project was developed on standard personal computers available to the team, with the application run locally for testing and demonstration. The source code was kept in a single version-controlled repository, which allowed each member to work on separate features and to merge them with a record of changes. The repository also holds the scripts folder that contains the automated test, so that every member can rerun the same checks on their own machine.

MongoDB was run as the local database during development, which allowed each member to work with an identical data model and to inspect the stored documents directly. Keeping

© 2026 IJSRET

the development database separate from any real student data was an important practice, both for privacy and for repeatability of the tests. Each test run begins from a known state, so that the results do not depend on data left behind by an earlier run.

Coordination among team members was handled through regular meetings and shared notes, with decisions about the data model and the connection rules recorded so that they could be referred to later. The team found that writing these decisions down reduced confusion and prevented the same question from being debated more than once.

7. Activity Flow For Profile Creation

Figure Figure 8 traces the flow a student follows when creating a profile. The flow includes a validation step that checks the required fields before the profile is saved. If validation fails, the student is shown the errors and may correct them and resubmit. Only a valid profile is written to MongoDB and made visible in discovery.

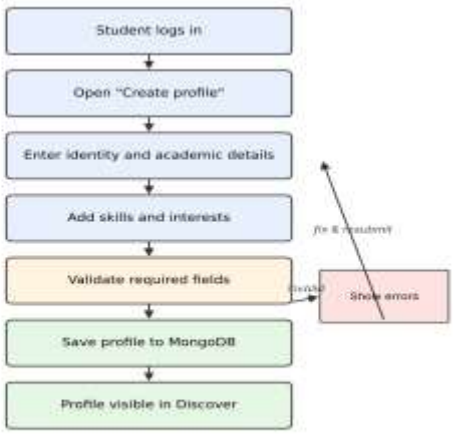

Figure 8: Activity flow for creating a student profile

8. Organisation Of The Code

The code is organised by responsibility so that each part of the system can be located and changed with minimal disturbance to the rest. Table Table 8 describes the main parts. The grouping mirrors the three modules introduced in Section 5.1 and keeps the data access separate from the request handling.

Table 8: Organisation of the Campus Connect project

Part of the project	Responsibility	Relationship to design
Interface files	Presents profiles, connections and conversations	Corresponds to the browser layer in Figure Figure 2
Request handling	Receives requests, applies rules and returns results	Corresponds to the application server layer
Data access	Reads and writes users, connections, conversations and messages	Implements the collections in Figure Figure 3
Connection rules	Checks duplicates and allowed state changes	Implements the lifecycle in Figure Figure 4
Test scripts (scripts folder)	Runs the two-account test	Implements the sequence in Figure Figure 9
Leftover authentication code	Former Google sign-in code, not active	Documented as a leftover in Section 5.5

VII. TESTING AND EVALUATION

Testing was a central part of the project, because a messaging platform fails in visible and damaging ways when its basic flows break. A failed connection check could expose a student to unwanted messages, and a lost message could break a conversation that matters to the people involved. The team therefore adopted a testing strategy that focuses on the interaction between users.

1. The Two-Account Test

The central test is a script, test_two_accounts.js, which logs in as two separate students. The script sends and accepts a connection request, exchanges messages in both directions and then checks that each side has received what the other sent. Figure 9 shows the sequence of interactions that the test performs. Using two accounts in the same run tests behaviour that matters on the platform: the connection between two participants, and the way that a message sent by one is received by the other.

© 2026 IJSRET

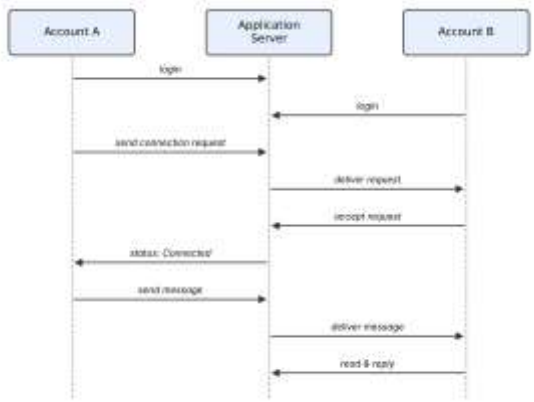

Figure 9: Sequence of the two-account test covering login, connection, acceptance and messaging.

The approach has several advantages. It can be run repeatedly after each change to catch regressions. It does not depend on manual clicking through the interface. It exercises the same database records that real users create, so it checks the real stored state rather than a simulated one. It also exposed cases in which a connection or message appeared correctly for one user but not for the other, which is exactly the kind of inconsistency that a single-user test would miss.

2. Test Cases

=

The test cases are organised around the requirements in Table 4. Each case has an expected result that can be compared against the observed behaviour. Table 9 lists the principal cases.

Table 9: Principal test cases used for the two-account evaluation

Test case	Steps	Expected result
TC-1 Create profile	Log in as Account A; enter details; save	Profile stored and visible to Account A
TC-2 Browse profile	Account A opens Account B's profile	Profile details displayed
TC-3 Send request	Account A sends request to Account B	Connection pending; visible to B
TC-4 Accept request	Account B accepts the request	Connection status is Connected for both
TC-5 Message when connected	Account A sends a message to B	Message stored and received by B
TC-6 Reply	Account B replies to A	Reply stored; conversation shows both messages
TC-7 Message before connection	Attempt message while request is pending	Message refused by the server
TC-8 Reload history	Log out and log in again	Conversation history reloads intact

3. Results And Observations

The two-account test covers the core connection and messaging flow, and the team used it after each significant change. The most valuable outcome of the testing was the confirmation that the server, not the browser, enforces the connection rule. The test case in which a message is attempted before a connection is accepted confirms that the restriction holds regardless of what the interface displays. The team also found that the test helped clarify the states of the connection lifecycle, because writing the expected result for each step required a precise definition of what each state means.

The evaluation is functional rather than statistical. It establishes that the core behaviours work as specified and that the connection and messaging rules are enforced. It does not measure performance under heavy load, nor does it measure usability with a broad group of students. These are important next steps and are discussed in Sections 8 and 9.

4. Procedure For Reproducing The Test

To allow the results to be reproduced, the team documented the procedure for running the two-account test. The procedure begins with starting the local database and the application server. The test script is then run, and it creates the two test accounts, performs the connection and messaging steps shown in Figure 9, and reports whether each step produced the expected result. Because the script uses the same database records as the application, a failure in the script usually points to a genuine failure in the platform, which makes the test useful for diagnosis as well as for confirmation.

If a step fails, the team's practice was to inspect the stored connection and message records before changing the code. This practice distinguishes a fault in the logic from a fault in the test itself, and it has the benefit of showing exactly what the database holds at the point of failure. Once the cause was

© 2026 IJSRET

identified, the fix was made and the entire test was rerun from the beginning, so that a repaired step could not hide a problem in an earlier one.

The test is deliberately simple in structure. Each step is short and named after the behaviour it checks, so that a reader can understand what is being verified without reading the implementation. This readability was chosen so that the test would serve as documentation of the expected behaviour as well as a check on it.

5. Requirements Traceability

A traceability matrix links each requirement to the design element that addresses it and to the test that verifies it. Table Table 10 provides this link. A requirement that has no test in the matrix is a gap in the evaluation, and the matrix makes such gaps visible. The matrix is the main tool the team used to confirm that the evaluation covers the stated requirements.

Table 10: Requirements traceability matrix

Requirement	Design element	Verifying test case
FR-1 Create and edit profile	Profile data model (Figure Figure 3); activity flow (Figure Figure 8)	TC-1 Create profile
FR-2 Browse profiles	Discovery module (Figure Figure 2)	TC-2 Browse profile
FR-3 Send connection request	Connection lifecycle (Figure Figure 4); sequence (Figure Figure 5)	TC-3 Send request
FR-4 Accept or decline request	Connection lifecycle (Figure Figure 4)	TC-4 Accept request
FR-5 Messaging only when connected	Messaging rule in the server (Section 5.4)	TC-5, TC-7
FR-6 Stored and reloadable messages	Conversation and message collections (Figure Figure 3)	TC-6, TC-8
FR-7 Persistence in MongoDB	Data model (Figure Figure 3)	All test cases read stored records
FR-8 Automated two-account test	Test scripts (Figure Figure 9)	Whole two-account run

6. Test Environment And Data

Table Table 11 records the environment in which the testing was performed. Keeping the environment description explicit allows another person to repeat the evaluation under the same conditions, which is the basis of reproducibility in a software project.

Table 11: Test environment and data

Element	Description
Client	Standard web browser on a student computer
Server	Local application server during development and demonstration
Database	Local MongoDB instance used only for development and testing
Test data	Two test accounts created by the script; no real student records
Starting state	Database cleared or reset before each full run
Execution	Test script run from the scripts folder after each significant change

7. Defect Classification

Any defect found during testing was classified so that the team could see which kinds of fault occurred most often. Table Table 12 gives the classification used. The classes help distinguish faults in the logic of the connection rule from faults in the interface or in the test itself, which need different responses.

Table 12: Classification of defects found during testing

Class	Meaning	Typical response
Rule fault	Connection or messaging rule not enforced as specified	Correct the server rule; rerun the full two-account test
Data fault	Stored record does not match what was displayed or submitted	Inspect the stored document; correct the write or the read
Interface fault	Display or navigation problem with correct data	Correct the interface; check the rule was not affected
Test fault	The test script itself is wrong	Correct the script; confirm the expected result against the specification

© 2026 IJSRET

The team recognises that the present report does not include a quantitative defect count. Where such counts are added in a later version, they should be recorded from the actual test runs and reported with the date and environment of each run, so that the figures can be checked.

8. Limitations

The current version of Campus Connect has several limitations that the team recognises and that should be considered when assessing the project.

- Matching is manual. Students find one another by browsing and by the skills and interests they enter. The platform does not rank or recommend profiles.
- Test coverage is focused. The two-account test covers the core flows, but it does not exhaustively test every interface edge case or every error condition.
- The evaluation has been carried out by the development team. The usability of the design has not been tested with the intended audience of students.
- Performance has not been measured. The behaviour of the system with a large number of students and messages is unknown.
- Leftover code remains. The Google sign-in code from an earlier design adds clutter to the codebase and should be removed in a cleanup pass.

These limitations do not undermine the core contribution of the project, which is a working and tested connection and messaging system with a clear data model. They do, however, define the boundaries of what the project has demonstrated, and the paper presents them so that readers can judge the work accurately.

9. Design Trade-Offs

Every design involves choices between competing goals. Table Table 13 records the main trade-offs made in Campus Connect, the option that was chosen, and the reason. Recording these decisions explicitly makes it possible for a later team to revisit them with the same information.

Table 13: Design trade-offs and the reasons for the chosen options

Decision	Option chosen	Alternative considered	Reason
Connection rule	Mutual acceptance before messaging	Open messaging to all students	Protects students from unsolicited contact
Database model	Separate collections with references	Embedding messages in user records	Keeps documents small; allows direct conversation loading
Authority of state	Server holds connection status	Browser holds status	Keeps both users consistent
Matching	Manual browsing of profiles	Automatic AI matching in version one	Core flows first; matching depends on clean profile data
Testing approach	Two-account scripted test	Single-user unit tests only	Exercises the interaction that defines the platform
Authentication	Application account handling	Google sign-in as the active method	Leftover code recorded; active flow kept simple

10. Threats To Validity

A careful report states the conditions that limit how far its conclusions can be trusted. Table Table 14 lists the threats to validity for this project and the steps taken, or still needed, to address each one.

Table 14: Threats to validity and responses

Type of threat	Description	Action taken or needed
Internal validity	Developers tested their own design, which may hide assumptions	Two-account test written from the specification; wider testing still needed
Construct validity	The test checks the flow chosen, not every possible user behaviour	Coverage mapped in the traceability matrix; gaps listed

© 2026 IJSRET

Type of threat	Description	Action taken or needed
External validity	Results come from a small team and a small student group	Usability study with students is planned future work
Reliability	Single test environment may not reflect other machines	Environment documented in Table Table 11
Accuracy of description	Technical description was prepared from the team's design record	Authors to check each section against the final code before submission

VIII. FUTURE WORK

Two extensions were identified during the project and are recorded here as planned directions. Neither is implemented in the present version.

1. Semantic Skill Matching

The first extension is semantic skill matching using Google Gemini. The aim is to read a student's skills and project interests and to suggest peers whose competencies are complementary or closely related, going beyond exact keyword overlap. For example, a student who lists experience with a particular data-processing library might be matched with a peer whose project used a related technique, even if the two never used the same term. The feature depends on the profile data being consistent, which is one reason the simple and structured profile design was chosen in the first version.

2. Ai Hackathon Team Builder

The second extension is an AI-assisted team builder for hackathons. Using the same profile data, the system would propose balanced teams, considering the mix of skills that each team needs and the availability and interests of the students involved. The team builder would extend the matching capability and would be a useful feature for students who want to take part in competitive events but do not know whom to approach.

3. Other Directions

Beyond these extensions, the team identified several improvements that would strengthen the platform. These include a dedicated cleanup pass to remove leftover code, broader automated tests covering error conditions, a usability study with a group of students, and performance testing with realistic data volumes. Each of these would make the platform more robust and would provide stronger evidence for its design.

4. Roadmap Of Development And Future Phases

Figure Figure 10 places the completed development phases alongside the planned extensions. The completed phases followed the dependency order described in Section 4.3. The planned extensions are shown separately to make clear that they are not part of the delivered system. Table Table 15 gives the priority and the prerequisite of each planned item, so that the order of later work can be justified.

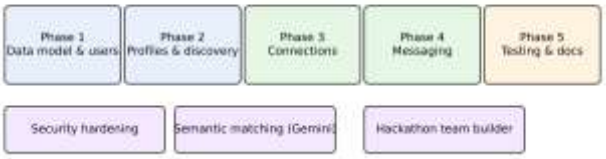

Figure 10: Development phases (completed) and planned extensions (future work)

Table 15: Priority and prerequisites of planned extensions

Planned item	Priority	Prerequisite	Reason for order
Remove leftover authentication code	High	None	Simplifies the codebase before further change
Security hardening (input validation, abuse controls)	High	Stable profile and connection rules	Required before any wider use
Usability study with students	High	Working interface	Gives evidence on the design with real users
Broader automated tests	Medium	Traceability matrix	Extends coverage to error conditions
Performance testing	Medium	Realistic data volume	Measures behaviour as the number of users grows

© 2026 IJSRET

Planned item	Priority	Prerequisite	Reason for order
Semantic skill matching (Gemini)	Medium	Consistent profile data and security review	Depends on clean skills data
AI hackathon team builder	Low	Semantic matching	Builds on the matching capability

IX. CONCLUSION

Campus Connect demonstrates that a focused networking and messaging platform can be built as a student project with a clear data model, a working connection and messaging flow, and a repeatable two-account test. The project addressed a real problem that students face in finding peers and starting conversations, and it did so by bringing profiles, connections and messages into one system. The team worked through requirements, design, implementation and testing, and gained practical experience of building a system that real people would use.

The most important design decisions were to make connections mutual, to keep the authoritative state on the server, to store messages in their own collection, and to verify behaviour through a test that involves two participants. These decisions gave the system a sound foundation. The project also reflects an honest approach to its own scope: it records what is implemented, what is a leftover, and what is planned, rather than overstating the result.

The next stage of the work is to strengthen the matching features, remove the remaining leftover code, and evaluate the platform with a wider group of students. The team is grateful for the guidance of its supervisor, Mr. Sachin Chauhan, and of its coordinator, Dr. Gurminder Kaur, whose support shaped the project throughout.

1 Appendix A: Glossary Of Terms

Table 16: Glossary of terms used in the paper

Term	Meaning in this project
Profile	The record a student creates describing their identity, academic details, skills and interests.
Connection	A mutual relationship between two students, created by a request and an acceptance.
Pending	The state of a connection request that has been sent but not yet answered.
Conversation	The stored sequence of messages exchanged between two connected students.
Collection	A group of related documents in MongoDB, such as users or messages.
Two-account test	The automated test that logs in as two students and checks the connection and messaging flow.
Leftover	Code from an earlier design that remains in the repository but is not part of the active flow.

2.. Appendix B: Index Of Figures And Tables

Figures: Figure 2 system architecture (Section 5.1); Figure 3 logical data model (Section 5.2); Figure 4 connection lifecycle (Section 5.3); Figure 5 connection sequence (Section 5.7); Figure 6 deployment (Section 5.8); Figure 7 profile wireframe (Section 5.9); Figure 1 use cases (Section 4.4); Figure 8 profile activity flow (Section 6.7); Figure 9 two-account sequence (Section 7.1); Figure 10 roadmap (Section 9.4).

Tables: Table 4 functional requirements (Section 4.1); Table 1 comparison of approaches (Section 2.1); Table 2 stakeholders (Section 3.3); Table 3 scope and constraints (Section 3.4); Table 5 use cases (Section 4.4); Table 6 risks (Section 4.5); Table 7 feature status (Section 6.5); Table 8 code organisation (Section 6.8); Table 9 test cases (Section 7.2); Table 10 traceability (Section 7.5); Table 11 test environment (Section 7.6); Table 12 defect classes (Section 7.7); Table 13 design trade-offs (Section 8.1); Table 14 threats to validity (Section 8.2); Table 15 planned extensions (Section 9.4); Table 16 glossary (Appendix A).

3. Appendix C: Logical Data Dictionary

The data dictionary lists the logical fields of each collection as designed for Campus Connect. It describes the intended structure of the stored records and is provided so that the data model can be checked against the implementation.

Table 17: Logical data dictionary of the MongoDB collections

Collection	Field	Type (logical)	Purpose
users	_id	Identifier	Unique reference for each student

© 2026 IJSRET

Collection	Field	Type (logical)	Purpose
users	name	Text	Display name of the student
users	academic info	Structured text	Programme and year of study
users	skills, interests	Array of text	Basis for discovery
connections	requester_id, recipient_id	Identifier	The two students in a connection
connections	status	Text (enumerated)	Pending, connected, rejected or withdrawn
conversations	participant_ids	Array of identifier	The students in a conversation
conversations	last_message_at	Date and time	Orders conversations by activity
messages	conversation_id, sender_id	Identifier	Links a message to its conversation and sender
messages	text, created_at	Text; date and time	Message content and time of creation

4. Appendix D: Steps For Running The Two-Account Test

Table 18: Procedure for the two-account test

Step	Action	Check
1	Start the local MongoDB instance	Database accepts connections
2	Start the application server	Server responds to a health request
3	Run the test script from the scripts folder	Script starts and creates two test accounts
4	Observe the login and profile steps	Both accounts log in
5	Observe the connection request and acceptance	Status shows connected for both accounts
6	Observe the message exchange in both directions	Each account receives the other's message
7	Attempt a message before acceptance (negative case)	Server refuses the message
8	Review the script output and stored records	Output agrees with Table Table 9

REFERENCES

1. MongoDB, Inc. MongoDB Manual. Available at: https://www.mongodb.com/docs/manual/ (accessed for the database design described in Section 5.2).
2. Google. Gemini API Documentation. Available at: https://ai.google.dev/gemini-api/docs (referenced for the planned semantic matching described in Section 9.1).
3. Sommerville, I. Software Engineering, 10th ed. Pearson, 2015 (referenced for requirements and testing practice in Sections 4 and 7).

© 2026 IJSRET