Software Engineering Systems Report

By Jaiden Wu

Tempe High School

Contents

1. Identifying and defining

1.1. Define and analyse problem requirements

Problem Context

Currently, volleyball teams face fragmented communication across group chats, emails, and verbal reminders, leading to confusion around training schedules, missed updates, and poor attendance tracking. Coaches and players struggle to coordinate efficiently, especially when dealing with last-minute changes. A centralized platform will resolve these issues by providing a single, reliable hub for all team-related communication, attendance responses and news sharing.

Project Value

This project will significantly improve how volleyball teams coordinate and communicate. Coaches will save time managing attendance and event planning, while players will benefit from quick access to schedules and team updates. The app enhances team cohesion, accountability, and convenience, making it a valuable tool for both players and coaches.

Intended Audience

The software is tailored for volleyball players and coaches operating within school or local club environments. It supports both team management and player engagement by providing tools to track attendance and send messages. Specific client users include Volleyball Coach Ms. Bolton and player Mr. Ngo, who represent the primary needs and feedback channels for development and testing.

Intended Use

The platform will be used to manage team communication, confirm attendance at training sessions and matches and send private messages to team members. It serves as a daily tool for both coaches and players to stay organized, informed, and connected with their teams. The software will be accessible across Android and iPhone devices for mobile convenience, as well as laptops and desktop computers for broader accessibility during planning or coaching sessions.

Functional Requirements

Messaging & Group Communication

The platform will facilitate both team-wide group chats and private one-on-one messaging, enabling seamless communication between players and coaches. Real-time messaging will be supported through, and messages will be stored in a database for future reference.

Team Creation & Joining

Coaches can create teams by specifying a team name and description. Players can search for and join existing teams using a search function. Role-based permissions will restrict team creation and management to coaches only.

Event Scheduling & Attendance Tracking

Coaches can schedule and manage training sessions, matches, and team events. Events can be created as recurring or one-off. Players can RSVP to events within the app to confirm attendance to events. Attendance records will be viewable by coaches, who can monitor team engagement and participation.

User Roles & Group Management

The app will feature clearly defined user roles, including Player and Coach, each with tailored permissions and access. The user will need to login to authenticate and authorise themselves. Users will be encouraged to self-assign to their appropriate team or group.

Search & Navigation

To improve user experience, the app will include a search function that allows users to quickly locate past conversations or teams by keyword. This is especially helpful when trying to find specific information in long chat threads or searching for a specific team to join.

Non-Functional Requirements

Security & Privacy

- Users must provide basic personal information during account creation, including full name and email address, ensuring proper user identification while minimising unnecessary data collection.
- All sensitive data—such as messages, event attendance, and profile information—will be securely stored using encryption and protected against unauthorised access.
- The system will enforce user authentication at regular intervals (e.g., every 24 hours) to prevent persistent unauthorised access.
- Users will be given control over their data, with the ability to update their profile or permanently delete their account and personal records.

Usability & Accessibility

- The platform will offer both light and dark mode themes, catering to user comfort and personal preference.

- The interface will be designed with accessibility in mind, including support for larger font sizes and colour-blind-friendly palettes, ensuring the app is usable for all players, including those with visual impairments.

Performance & Reliability

- The system will be optimised for consistent performance across all major devices, including smartphones (iOS and Android), tablets, and desktops.
- Fast load times and minimal latency will be prioritised, especially in key areas like chat, event loading, and RSVP functionality.
- Client feedback about lag and sluggishness in comparable apps (e.g., TeamApp.) has been addressed through more efficient backend architecture and lightweight frontend design.

Scalability & Maintainability

- The platform will be built with scalability in mind, allowing for multiple teams or clubs to use the system without performance drops.
- A modular design will allow future features like VOD playback, stat tracking, or cross-team communication to be integrated without requiring a full system redesign.
- Clear documentation and maintainable code will ensure that future developers can easily update or expand the app, based on feedback or evolving client needs.

Availability & Support

- The platform will maintain high uptime, ensuring consistent availability, particularly around key usage times such as just before training or matches.

Boundaries

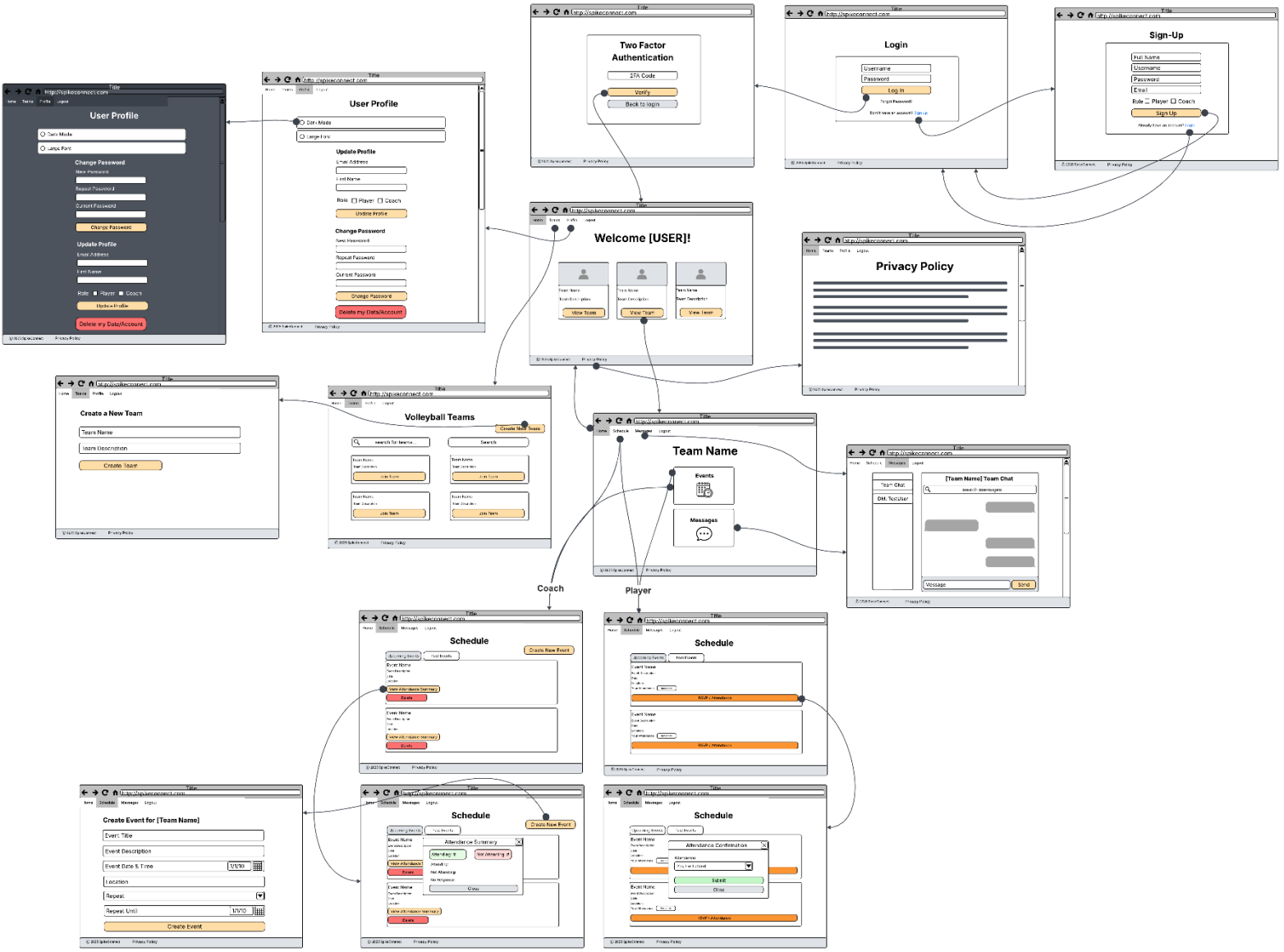
This system is designed to operate across a range of modern mobile and desktop devices and must remain compatible with widely used operating systems including iOS, Android, and Windows. Functionality is dependent on stable internet connectivity, particularly for real-time features such as messaging, attendance updates, and event syncing, meaning limited capability in offline environments.

Security requirements also define system boundaries. All user data must be encrypted in storage , with strict authentication and role-based access control to prevent unauthorised access to sensitive information. Finally, the application must maintain consistent responsiveness and performance across devices with varying hardware specifications, ensuring a smooth and accessible experience for all users.

1.2. Tools to develop ideas and generate solutions

Application of appropriate software development tools

Storyboard (For Closer Look!)



1.3. Implementation method

For the rollout of the Spike Connect, the most suitable implementation strategy is **Pilot Implementation**. This approach involves initially deploying the software to a small, controlled group, in this case, Ms. Bolton's volleyball team, before extending it to a wider user base. A pilot rollout allows for real-world testing of the application's functionality, user interface, and performance within the actual context it is designed for.

Using this method reduces the risk of widespread issues by limiting the initial exposure of the system. Feedback gathered from the coach and players during this phase can be used to identify bugs, enhance usability, and add or refine features. It also allows for monitoring of how the software integrates into the daily routines of a team without disrupting communication across multiple groups.

Furthermore, starting with a single team ensures that technical support and maintenance are manageable during the early stages. Once the platform proves stable and effective in the pilot group, it can be confidently scaled to other teams, clubs, or schools.

The **other implementation methods** are less suitable for the nature and scale of this project:

- **Direct Cutover** would be too risky, as any issues during launch could disrupt communication for the entire team or club, especially if the system has not been fully tested in a live environment.
- **Phased Implementation** could delay essential functions like messaging or attendance tracking, making the app feel incomplete or inconsistent during use.
- **Parallel Implementation** is unnecessary and inefficient in this case since most teams don't rely on complex legacy systems. Maintaining both platforms would increase workload and cause confusion for users.

1.4. Financial feasibility

SWOT Analysis

Strengths	Weaknesses
• Purpose-built for volleyball teams, offering a tailored experience that addresses specific team management needs. • Centralised communication hub combining messaging, event scheduling, and attendance in one platform. • Role-based access ensures clear separation of functionality between players and coaches. • Integrated RSVP and attendance tracking improves team accountability and planning efficiency.	• Limited initial user base may slow feedback loops and reduce perceived usefulness in early stages. • Real-time features depend on stable internet connectivity, limiting use in offline environments. • Maintenance and updates may be difficult if one developer solely manages the platform. • Development constrained by project timelines, possibly limiting feature depth or polish at launch.
Opportunities	**Threats**
• Potential to scale the platform to support other sports teams, school clubs, or extracurricular groups. • Integration with external systems such as school calendars, video review platforms, or student management tools. • Future monetisation through optional premium features like advanced analytics, video tagging, or custom branding. • Collaboration with schools or local clubs to pilot the app and gather ongoing user feedback for iterative development.	• Established competitors (e.g. TeamApp, WhatsApp) already dominate the team communication space. • Privacy concerns from users, parents, or school administrators regarding data handling and messaging. • Bugs, security flaws, or performance issues at launch could harm user trust and long-term adoption. • Changes to third-party APIs, mobile OS updates, or app store policies may disrupt feature rollout or compatibility.

Study	Go or No Go?	Assessment and evidence
Market feasibility	Go	There is a clear niche for a communication platform tailored to school and local volleyball teams. Existing apps like TeamApp are not fully customisable for team-specific needs (e.g. attendance, parental boundaries). Feedback from Ms Bolton and Mr Ngo supports demand.
Cost of development	Go	The project will be developed using free, open-source tools such as Python, Flask, and SQLite. Estimated startup costs include domain registration ($20/year) and cloud hosting ($30/month), totalling approximately $2,500. No developer labour cost is required as the system is built by a student.
Cost of ownership	Go	Ongoing operational costs are minimal, primarily hosting fees (~$30/month) and occasional maintenance. The platform avoids costly third-party licenses, and internal management keeps maintenance low-cos
Income potential	Go	While initially free for schools and clubs, revenue can be generated through optional premium features (e.g. analytics, video tagging), local sponsorships, or subscription-based school plans in future iterations.
Future expansion opportunities	Go	The platform is highly scalable, with potential to expand to other sports teams, clubs, or school groups. Features like VOD review, performance analytics, or integration with broader school systems offer strong potential in the long term.

Opening Day Balance Sheet

Spike Connect
Opening Day Balance Sheet
[AUD $]

Category	Item	Amount (AUD)
Assets		
Current Assets	Cloud Hosting Credit (AWS)	500
	Domain Name (1 year)	30
	App Store Developer Account (Google/Apple)	150
	Total Current Assets	680
Non-Current Assets	Laptop/PC for Development	2000
	Software Licenses	300
	Total Non-Current Assets	2300
	Total Assets	2980
Liabilities		
Current Liabilities	Subscription Fees (e.g., backend services)	200
	Outstanding App Store Fees	500
	Total Current Liabilities	700
Non-Current Liabilities	*(None currently)*	0
	Total Non-Current Liabilities	0
	Total Liabilities	700
Equity	Owner's Capital / Initial Investment	2280
	Retained Earnings (grants, comps, etc.)	0
	Total Equity	2280
	Total Liabilities + Equity	2980

2. Research and planning

2.1. Project management

Software development approach

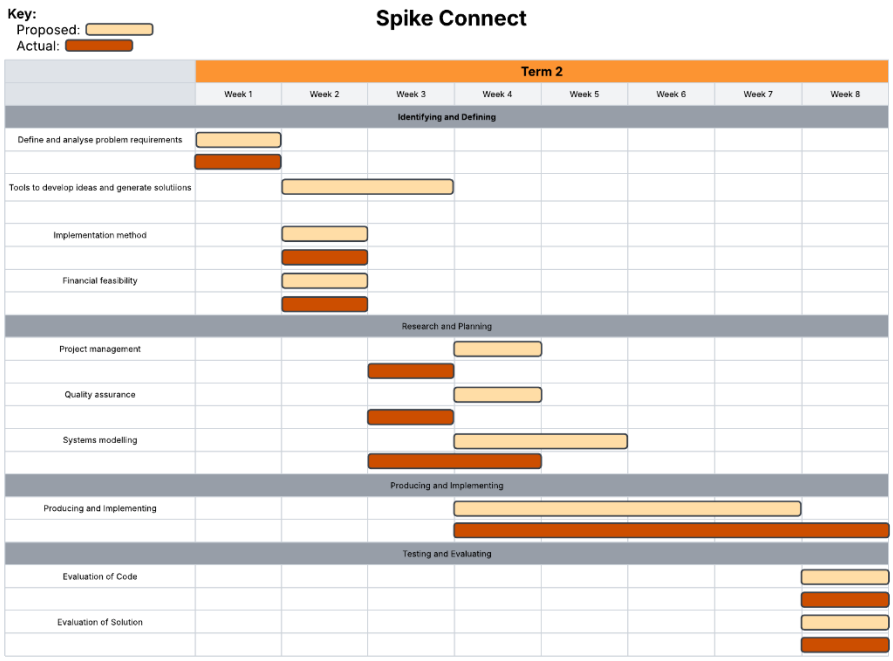
The most suitable software development approach for this project is **Agile**. This method allows for flexibility and iterative improvement, which aligns well with the evolving nature of the platform. Since the app is being developed with direct feedback from real clients (Ms Bolton and Mr Ngo), Agile supports continuous communication, allowing features to be tested, adjusted, or added based on their responses throughout the development cycle.

Why not Waterfall or WAgile?

- **Waterfall** is too rigid, as it locks all requirements early. This doesn't suit the feedback-driven and evolving nature of the app.
- **WAgile** is a blend of both, but still too structured for a project that benefits from rapid prototyping and client testing at each stage.

Scheduling and task allocation

GitHub was used for version control and code collaboration. Separate branches were created for each sprint to manage development stages and prevent code conflicts. GitHub Issues and Pull Requests helped track progress and review changes systematically.



[For Closer Look!]

2.2. Quality assurance

Quality criteria

Quality criteria	Explanation
Secure User Authentication	All users must log in using verified credentials, with securely hashed passwords and session-based re-authentication. This ensures only authorised access to private team data and communication.
Stable Group and Private Messaging	Group and private 1-on-1 messaging must function consistently without noticeable lag or loss of data. This enables seamless and uninterrupted communication between users.
Accurate Attendance Tracking	Players must be able to mark their attendance with a single tap, and coaches must be able to view attendance summaries in real time. This promotes accountability and easy tracking.
Functional Event View with RSVP	Events should be clearly displayed in a schedule page. Players must be easily able to RSVP and confirm attendance to events.
Accessibility Features	The app must support light/dark modes, scalable text sizes, and colour-blind-friendly palettes. This guarantees accessibility for all users, including those with visual impairments.

Compliance and legislative requirements

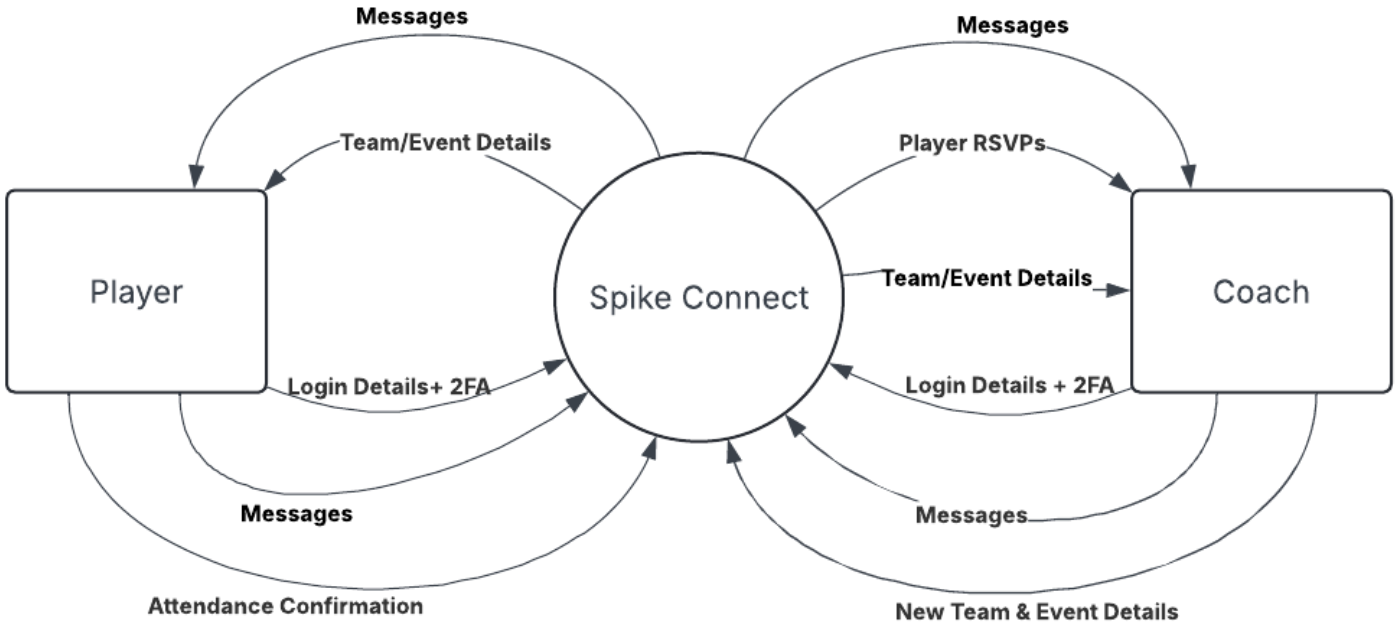
Compliance or legislative issue	Methods for mitigation
NSW Privacy and Personal Information Protection Act (1998)	Collect only essential personal information (e.g., name, email, age). Store data securely using encryption and provide a clear privacy policy outlining data usage.
Federal Privacy Act 1988	Obtain explicit user consent at sign-up. Provide options for users to update or delete their personal data upon request
Web Content Accessibility Guidelines (WCAG)	Ensure the interface meets accessibility standards by using appropriate font sizes, high contrast colour schemes, dark/light modes, and screen reader compatibility.
Data storage and cloud security standards	Host the platform using services compliant with ISO/IEC 27001 or similar standards. Enforce HTTPS encryption, secure authentication, and strict access control policies.

2.3. Systems modelling

Students are to **develop** the given tables and diagrams. Students should consult the Software Engineering Course Specifications guide should they require further detail, exemplars or information. Each subsection below should be completed with Section 1.1. in mind.

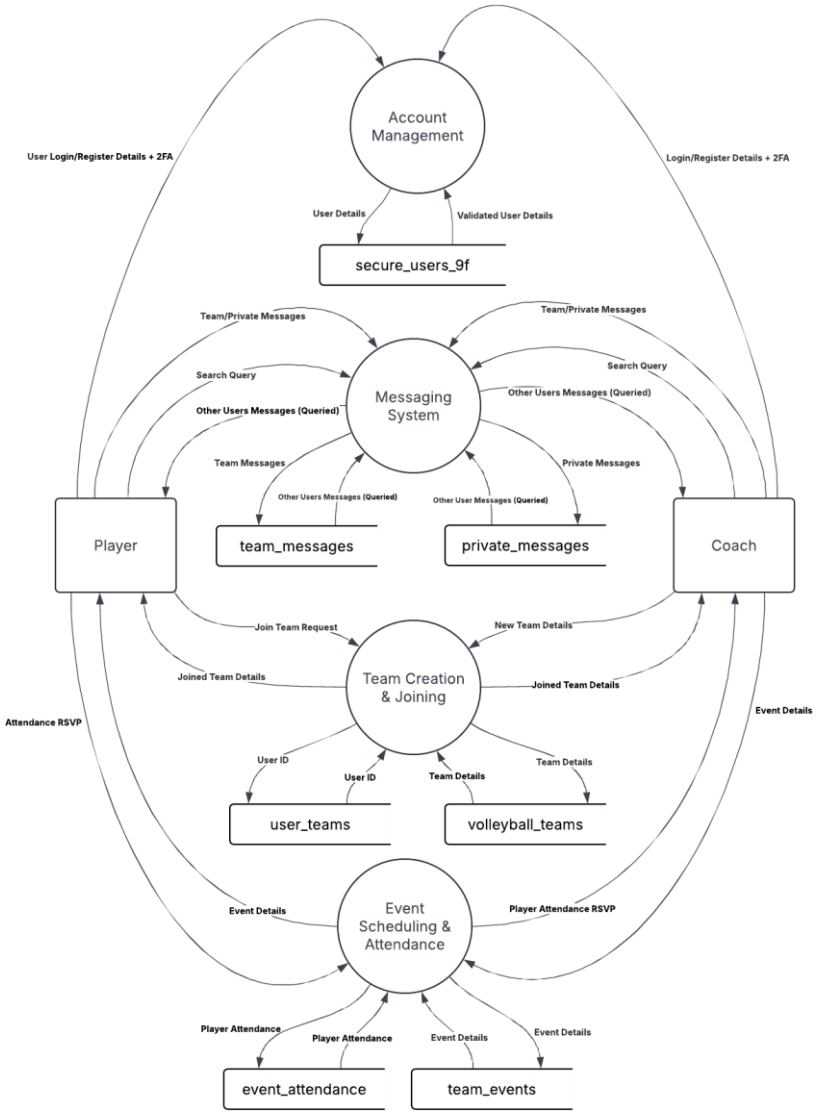
Data flow diagrams (For Closer Look!)

Level 0

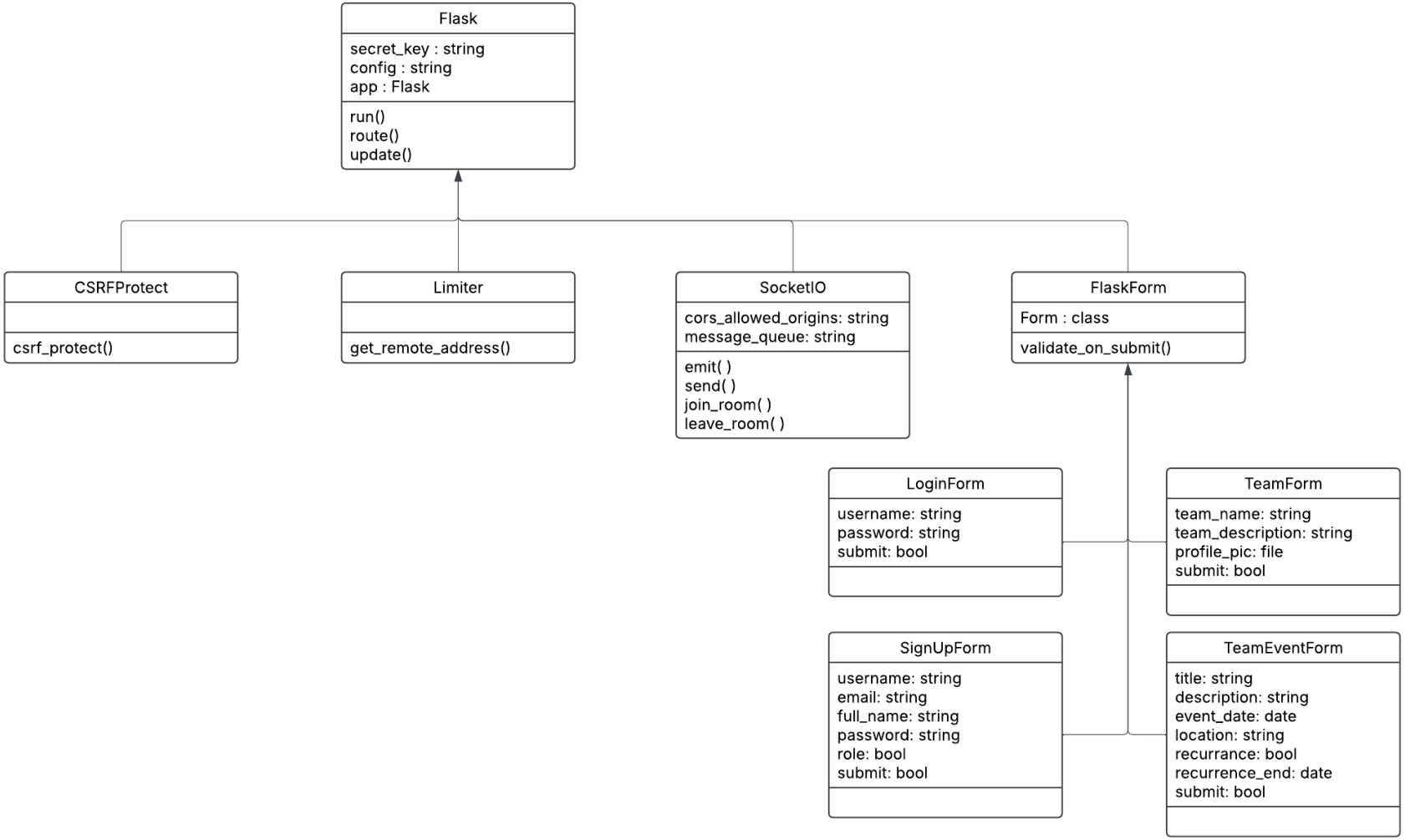


Tempe High School

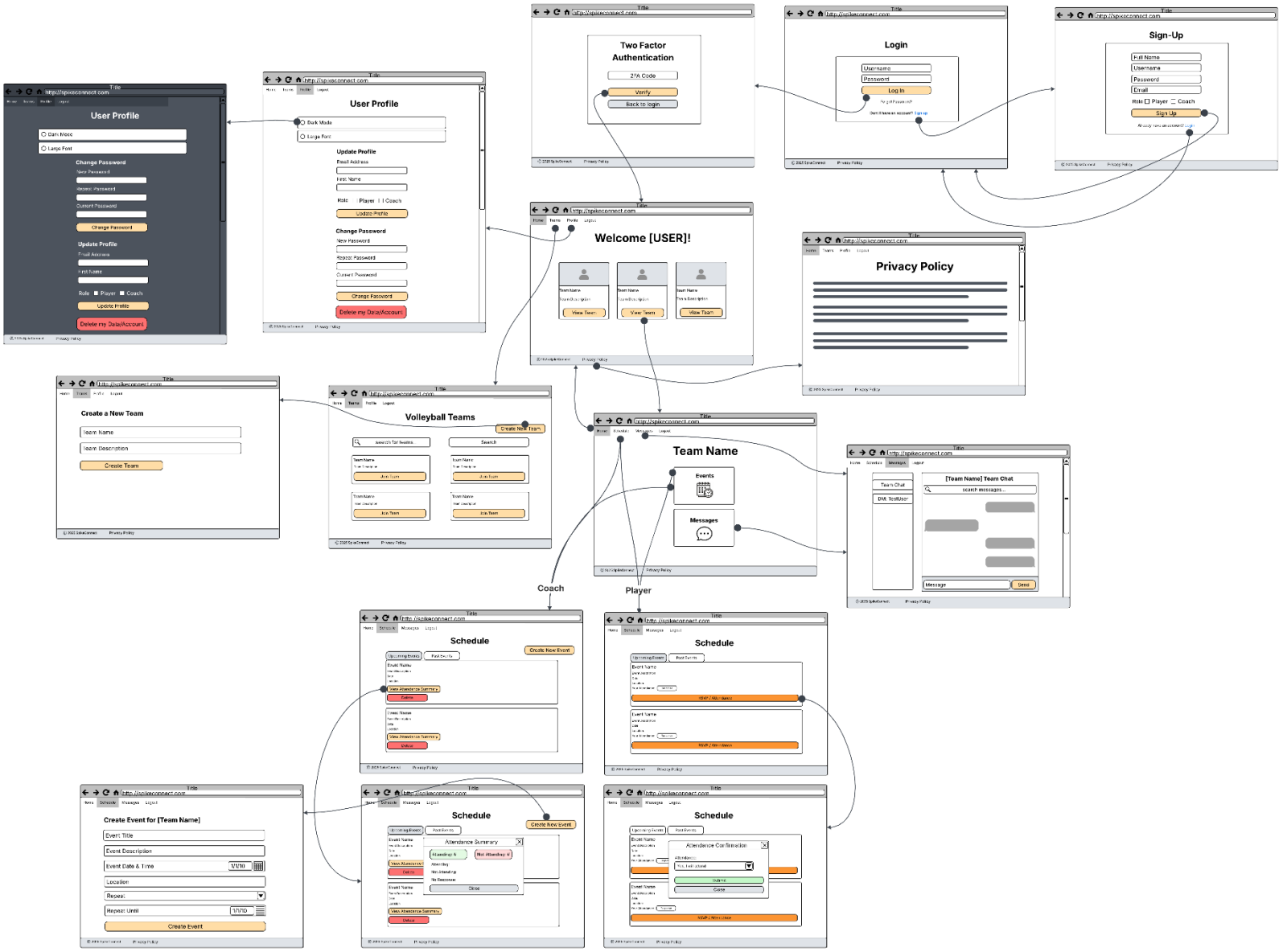
Level 1


Account Management
User Login/Register Details + 2FA
Login/Register Details + 2FA
User Details
Validated User Details
secure_users_9f
Team/Private Messages
Team/Private Messages
Search Query
Search Query
Messaging System
Other Users Messages (Queried)
Other Users Messages (Queried)
Team Messages
Private Messages
Other Users Messages (Queried)
Other User Messages (Queried)
Player
team_messages
private_messages
Coach
Join Team Request
New Team Details
Joined Team Details
Team Creation & Joining
Joined Team Details
Attendance RSVP
Event Details
User ID
User ID
Team Details
Team Details
user_teams
volleyball_teams
Event Details
Event Scheduling & Attendance
Player Attendance RSVP
Player Attendance
Player Attendance
Event Details
Event Details
event_attendance
team_events

Class diagrams (For Closer Look!)



Storyboards (For Closer Look!)

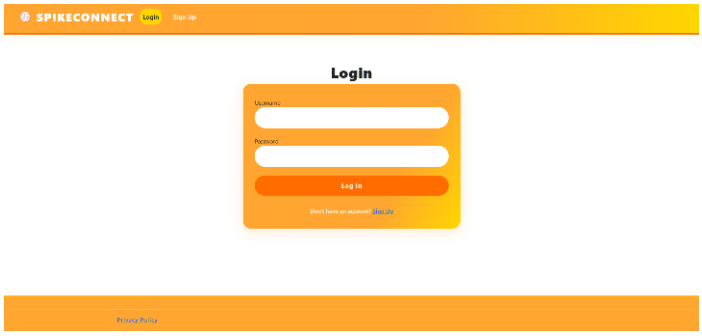


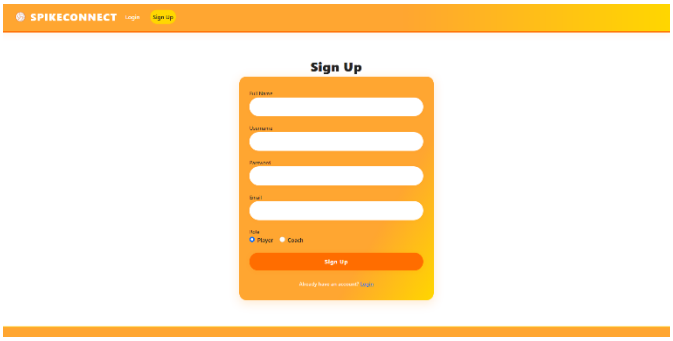
3. Producing and implementing

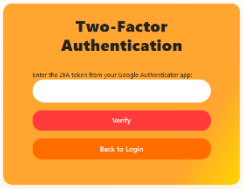
Solution to software problem

Login, Sign-Up, 2FA Page

- These pages handle user authentication and authorisation, allowing users to securely log in or sign up before accessing the PWA's features. This connects directly to **User Roles & Group Management** and **Security & Privacy**.

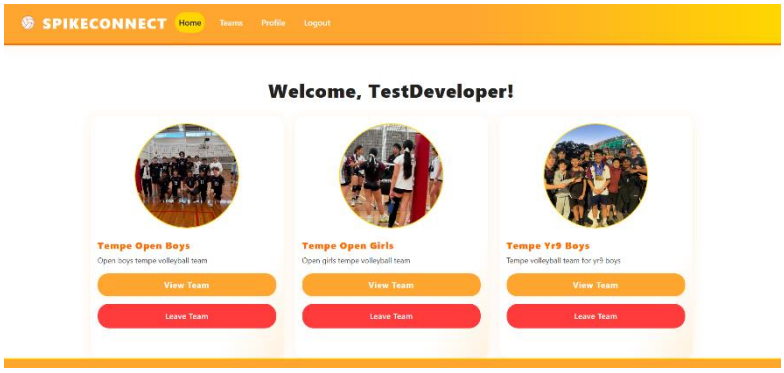


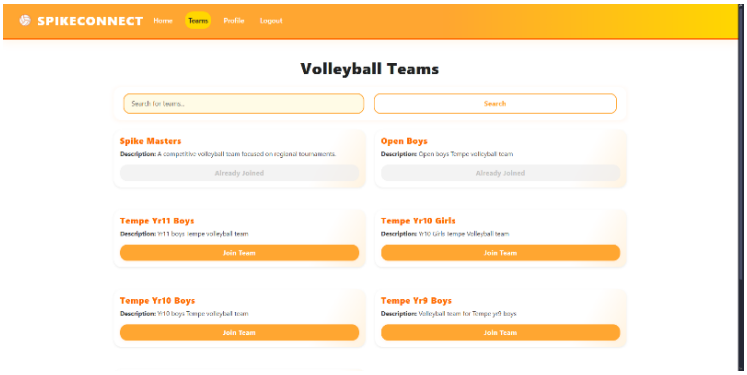




Home & Teams Page

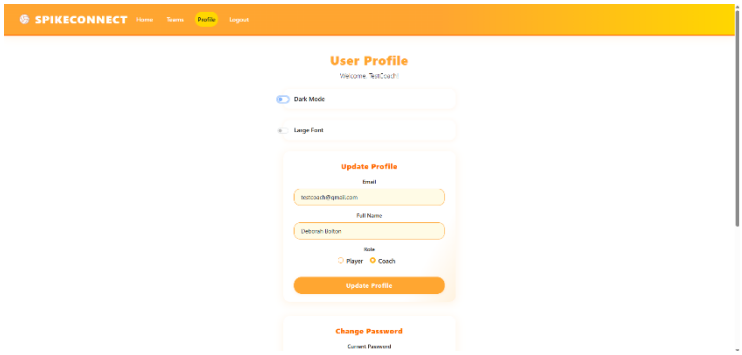
- These pages let users view the teams they belong to and join new teams by searching. Coaches have the additional ability to create teams. This ties into **Team Creation & Joining**, **User Roles & Group Management**, and **Search & Navigation**.

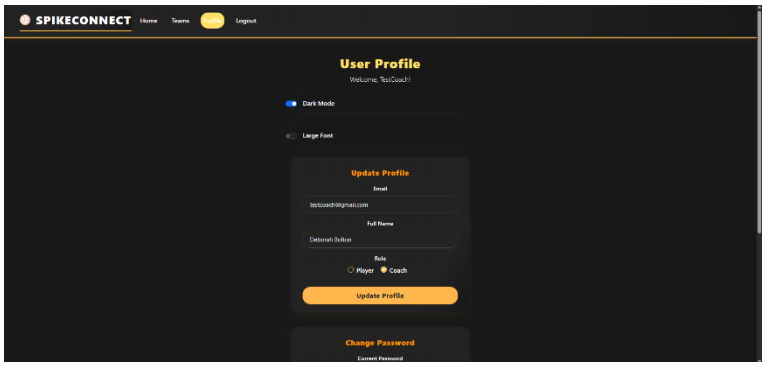


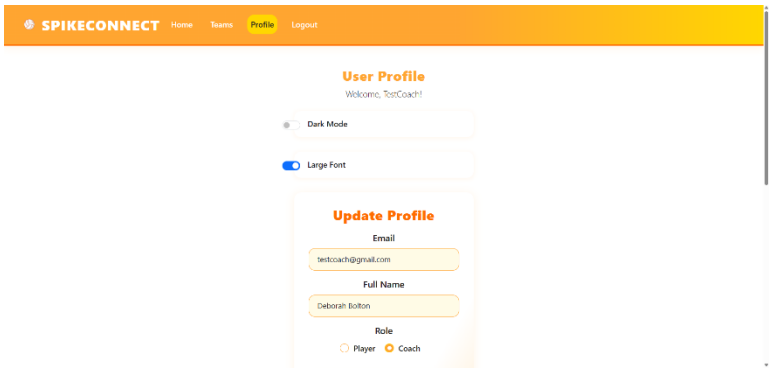


Profile Page

- The profile page enables users to update personal information, change passwords, adjust accessibility settings (e.g., dark mode, font size), and delete their data or account if desired. This supports both **Usability & Accessibility** and **Security & Privacy.**

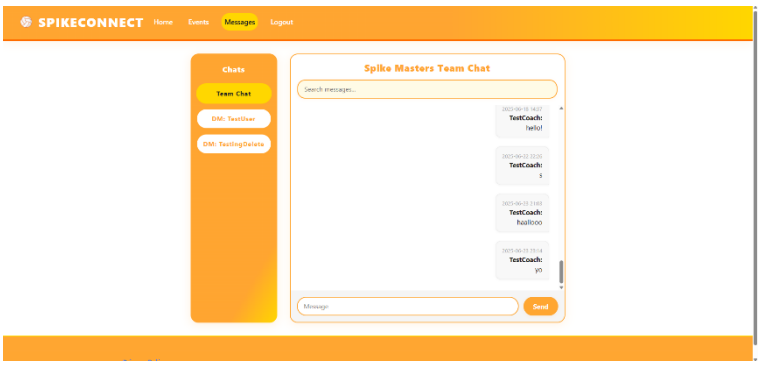






Messaging Page

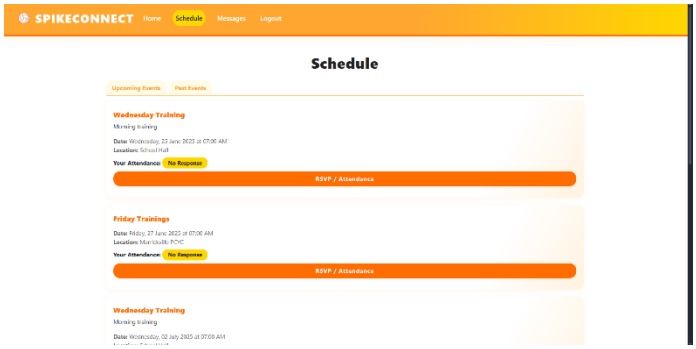
- This page provides access to both team-wide chat and private messaging between users. It supports real-time communication and moderation features, directly addressing **Messaging & Group Communication.**

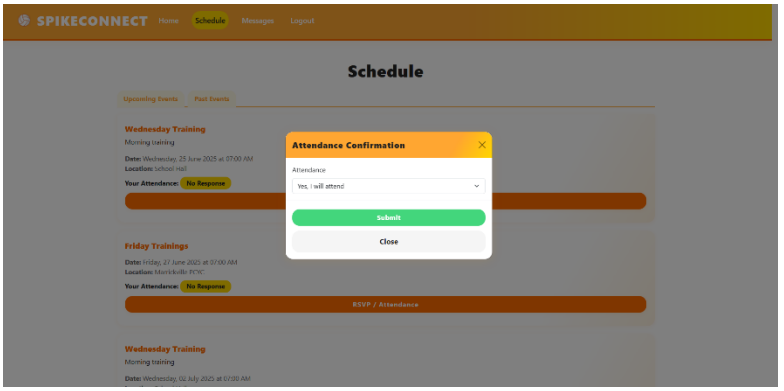


Event Scheduling & Attendance Tracking

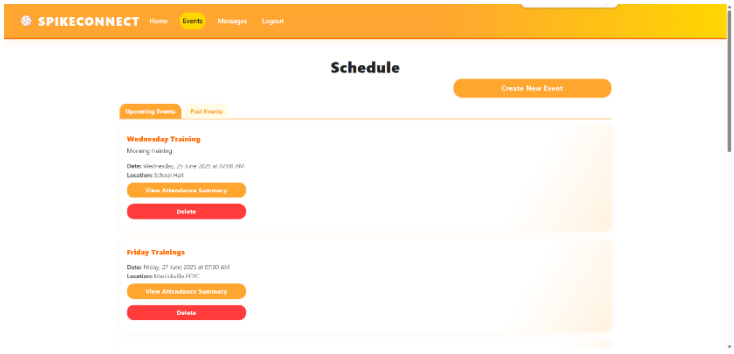
- The event page adapts based on the user's role. Players can view upcoming events and RSVP. Coaches can view attendance summaries, track responses, and create new events, with options for recurring sessions. This page links to **Event Scheduling & Attendance Tracking** and **User Roles & Group Management.**

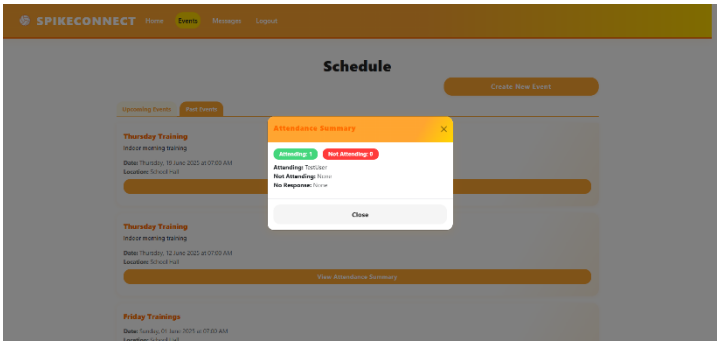
Player's Permission & Access

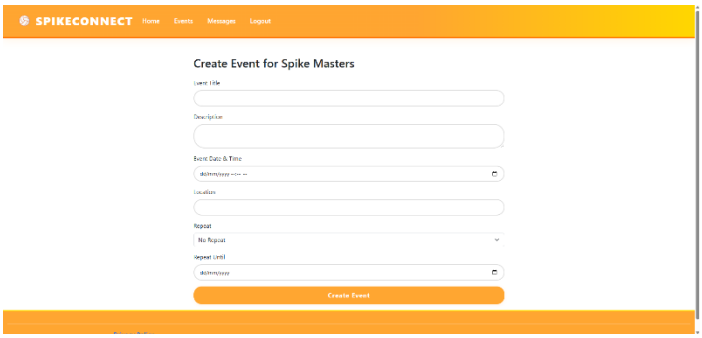




Coach's Permission & Access







Version control

For this project, **Git** through GitHub was used as the version control system. GitHub allowed for efficient tracking of changes, backup of code, and collaboration if needed. Version control was maintained through regular commits, each with clear messages describing updates or fixes. This approach made it easier to manage iterations, roll back to previous versions if issues arose, and maintain an organised development workflow throughout the SDLC.

To support my Agile development approach, I created separate branches for each sprint. This allowed me to focus on specific features without disrupting the main codebase. Once a sprint's tasks were completed and tested, the branch was merged into the main branch. This branching strategy improved project organisation, minimised integration issues, and aligned closely with Agile's iterative workflow.

4. Testing and evaluating

4.1. Evaluation of code

Methodology to test and evaluate code

Unit Testing

Unit testing involves testing the smallest functional units of code (e.g., functions or methods) in isolation to verify that each behaves as expected. I created unit tests for key backend functions such as validate_login(), get_user_role(), and create_event(). These tests helped ensure that individual logic components performed correctly, even when separated from the larger system.

Subsystem Testing

Subsystem testing verifies that multiple related modules or components (e.g., login system, chat module) work together correctly as a group. I tested subsystems like the authentication module (login, registration, role verification) and the messaging system (team chat, private messaging) to ensure that components interacted correctly and passed data between them without errors.

System Testing

System testing involves testing the entire integrated application as a whole to ensure that all modules function together according to the specified requirements. I performed a system test for my PWA, including logging in, joining a team, sending messages and RSVPing to events. This included testing both the player and coach role for their specific functionality and access.

Black Box Testing

In black box testing, the tester does not look at the internal code structure. The focus is on testing inputs and outputs based on functional requirements. I used black box testing, particularly during client meet-ups, to simulate user interactions such as submitting forms or clicking RSVP buttons, without referencing the backend logic, ensuring the interface met user expectations and functional specifications.

White Box Testing

White box testing involves examining the internal logic and code structure of the program. It’s used to check things like logic flow and edge cases. I reviewed functions like event creation and message sanitisation for logic errors, ensuring that each conditional path was tested, especially for edge cases such as invalid inputs or missing fields.

Grey Box Testing

Grey box testing combines both black box and white box approaches. The tester has partial knowledge of the system's internal structure. For example, when testing private messaging, I knew the database schema and internal room naming conventions. This allowed me to test the system both as an end-user and with insight into how the backend handled data.

Quality Assurance

Quality Assurance is a systematic process that ensures a software product meets its defined quality standards and requirements throughout the SDLC. In this project, quality was upheld through consistent testing, bug tracking, version control via GitHub, and regular client feedback. For instance, incorporating client suggestions, such as refining the user interface and improving the team joining process, helped validate that the solution aligned with the initial requirements and user expectations.

Code optimisation

Throughout the development of my project, I applied several code optimisation techniques to improve the performance, maintainability, and clarity of the system.

Dead Code Elimination:
I actively removed unused or unnecessary code such as redundant imports, variables, and functions. This helped reduce the overall codebase size, improved execution speed, and made the project easier to maintain.

Code Movement:
Where applicable, I optimised loops by moving constant expressions outside of them to prevent repeated computation. This reduced redundant processing and improved runtime efficiency, especially in functions dealing with data lookups and rendering.

Strength Reduction:
This method typically replaces expensive operations (e.g., multiplication or division) with simpler ones like addition or bit shifts. However, as my project doesn't rely on arithmetic-heavy operations, this technique was not applicable.

Common Sub-expression Elimination:
This technique aims to compute repeated expressions once and reuse the result. Since my project did not involve complex calculations or repeated expressions within performance-critical areas, this method wasn't necessary for my implementation.

Compile-Time Evaluation (Constant Folding & Propagation):

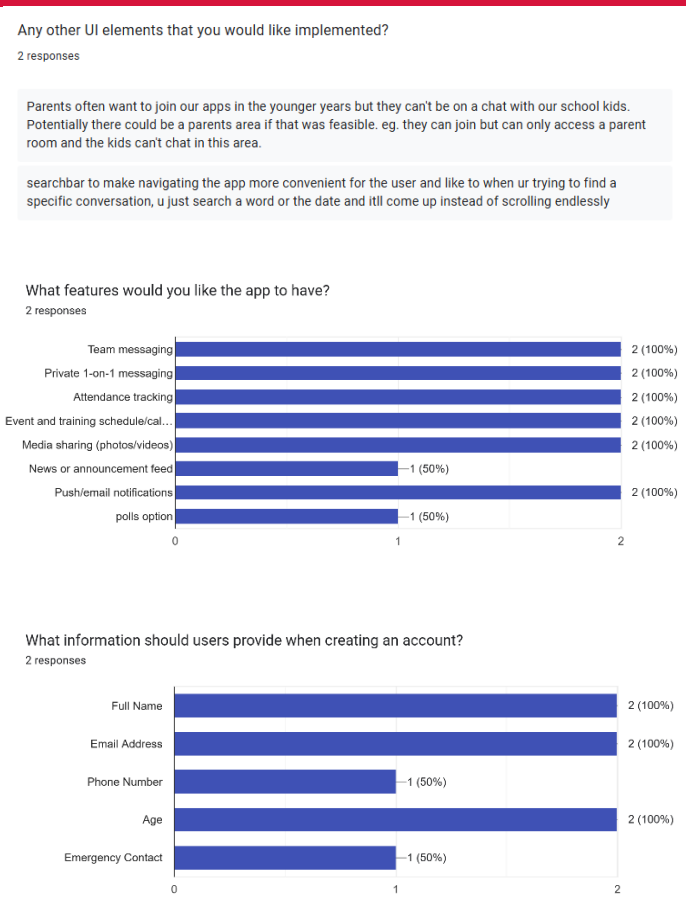
These optimisations are usually handled automatically by the Python interpreter and are not explicitly implemented in my code. My project doesn't require manual use of constant folding or propagation techniques.

Refactoring for Readability and Maintainability:

I extensively refactored my code by modularising key components. Core logic was separated into dedicated modules such as teamHandlers.py, authHandlers.py, and databaseManagement.py. To avoid repetition, I created reusable functions like get_logged_in_user() and require_role() to centralise shared logic. This made the codebase easier to read, evaluate, and extend.

4.2. Evaluation of solution

Analysis of feedback / Client Communication Table

Date	Description	Client Response/Feedback
20/4/25	Gathered requirements from clients through a Google Form.	
06/06/25	Met with client in person and presented my project pitch to them. Also showed the development progress of my app so far.	Hi Jaiden, Great chat today. Thanks for presenting the app you have created 'Spike Connect'. The app presentation was excellent and filled with interesting concept designs that promise an exciting new app that would be relevant and very useful in the volleyball program. I was delighted with the professionalism of the presentation and the opportunity to see how well the app was progressing. Thanks! Deborah Bolton **Additional Points** • Client liked how coaches can see attendance from previous events to keep players accountable. Addresses previous limitation with similar app TeamApp.

Date	Description	Client Response/Feedback
24/6/25	Met with one of the clients in person to present the current progress of the PWA, demonstrating key features and gathering feedback for further refinement.	• The role selection layout during sign-up was unclear, causing confusion about which label matched each option. • Issues were noted with the exclusion of special characters like hashtags (#) when creating a password during sign-up. • Client would like the ability to join and manage multiple teams under one account. • Dark mode and accessibility options (e.g. large fonts, high contrast) were well received. • Client praised app for its simple and intuitive layout.
25/6/25	Met with client to show almost finalised PWA and get final feedback.	This is just an email to confirm that Jaiden and I spent the last period looking over his assessment task on creating a platform for coaches and players. The program is very impressive and easy to navigate. I found it easy to join teams and use the chat function. My only suggestion was to possibly allow for a profile pic to be added for each group. Congratulations to Jaiden on completing this assessment task with such professionalism and skill.

Testing methods

Method	Applicability	Reasoning
Functional testing	Applicable	Used extensively to verify that each feature/function (e.g login, messaging, events) works as intended. Functional tests were run for all major features including event RSVPing, login, messaging, team creation and joining to ensure they met the requirement set out.
User Acceptance testing	Applicable	Conducted with clients to test features in real scenarios. Feedback was gathered during in-person demonstrations of the PWA with the client. This helped identify usability issues like role selection and clarity on functionality like joining multiple teams.
Live data	Applicable	Used when testing real users' accounts and team information during client meet-ups. Real user input from clients during in-person meetings was used to simulate authentic interactions and validate secure data handling within the system.
Simulated data	Applicable	Used in early stages to test feature like attendance and messaging before client testing. Simulated users, events and messages were created to test functionality in isolation before real data was introduced later with the client.
Beta testing	Not Applicable	This method was not fully utilised, as testing was limited to just two clients. However, in future development stages, the PWA could be trialled with an entire volleyball team to identify bugs, usability issues, and gather broader user feedback in a real-world setting.
Volume testing	Not Applicable	Volume testing was not conducted during this phase of the project. For the future, the system could be evaluated under high-load conditions, such as a large number of messages and RSVPs, to assess performance, responsiveness, and stability under typical usage by larger teams or clubs.

Security Assessment

Threat	Countermeasure
Broken Authentication and Session Management	• **Failed Login Policy & Rate Limiting**: I enforced a strict failed login policy with rate limiting and account lockout mechanisms using Flask Limiter to prevent brute-force attacks • **Session Management**: I ensured session tokens were securely managed on the server, properly discarded, and hidden from URLs to prevent session hijacking. • **Two-Factor Authentication (2FA):** I implemented 2FA to add an extra layer of security, requiring additional verification beyond just a password. • **Enforced Strong Passwords:** I mandated the use of strong, complex passwords and restricted credential reuse to protect against database attacks
Cross-Site Scripting and Cross-Site Request Forgery	• **Content Security Policy**: I've implemented CSP to restrict trusted sources for dynamic content, blocking malicious scripts from executing. • **Defensive Data Handling**: I sanitized and validated all user inputs to prevent the processing of harmful data like JavaScript, reducing the risk of XSS. • **Synchronizer Token Pattern**: Through Flask WTForms, I added unique tokens to each request to ensure they are legitimate, protecting against CSRF attacks.
Invalid Forwarding & Redirecting	• **Content Security Policy (CSP):** Restricts the sources from which content and scripts can be loaded, preventing malicious redirects and unauthorized resource loading. • **Input Validation:** Ensures that user-supplied URLs and redirect parameters are properly validated, rejecting or sanitizing any unexpected or untrusted inputs to prevent open redirect exploits.
Race Conditions	• **Session-Based Locking**: Implementing unique session IDs and using them as keys in a locking mechanism ensures that critical operations are executed sequentially, preventing race conditions from being exploited. • • **Asynchronous Input Encryption**: Encrypting form inputs asynchronously by using CSRF Protect helps prevent unauthorized or repeated form submissions, reducing the risk of race condition attacks. • **Rate Limiting**: Restricting the number of requests within a given timeframe prevents attackers from repeatedly triggering race conditions, reducing the likelihood of exploitation

Test data tables (Boundary testing)

Variable	Maximum	Minimum	Default Value	Expected Output	Actual Output	Reason for Inclusion
username	25 characters	4 characters	N/A	Contains only letters, numbers, and certain special characters. Within min and max character range.	Matches expected	Ensures usernames are of a suitable length to avoid confusion, maintain readability, and prevent database or UI layout issues
password	128 characters	8 characters	N/A	Contains an uppercase, lowercase, digit and special character. Within character range.	Matches expected	Minimum password length enforces basic security standards, reducing the risk of easily guessed or weak credentials.
email	120 characters	None	N/A	Valid email address within 120 characters.	Matches expected	Valid email formats ensure reliable user identification.
full_name	100 characters	2 characters	N/A	Within min and max character range.	Matches expected	Prevents UI rendering issues and helps maintain a clean and user-friendly interface.
team_name	50 characters	2 characters	N/A	Only contains letters, numbers, and spaces. Within character range	Matches expected	Prevents UI rendering issues and helps maintain a clean and user-friendly interface.

Analysis of solution against quality success criteria

Quality criteria	Met?	Analysis
Secure User Authentication	Yes	I successfully implemented secure user authentication through login verification, encrypted password storage, and 2FA. I enhanced security further by storing passwords securely and adding a re-authentication requirement using session tracking and QR-based 2FA on initial sign-up only. This aligns with the goal of ensuring only authorised users access sensitive features like team communication and attendance.
Stable Group and Private Messaging	Yes	Group and private messaging were both implemented using Flask-SocketIO, with support for real-time communication. I also handled the naming conventions for private rooms dynamically and ensured past messages were stored and retrieved from the database securely. This provided a smooth messaging experience with minimal delays, fulfilling the requirement for stable communication between team members.
Accurate Attendance Tracking	Yes	The attendance system was effectively implemented, allowing players to RSVP to events and see their own responses. Coaches were able to access attendance summaries showing both totals and individual names. The system was responsive and connected to the event database, fulfilling the need for accurate, real-time attendance tracking and accountability.
Functional Event View with RSVP	Yes	Events were accessible via a dedicated tab, and players could RSVP. Coaches had tools to create recurring events and monitor attendance. The event logic adjusted dynamically based on user roles (player vs coach), making the system both functional and user-friendly. This meets the quality standard for clear, role-based event interaction.
Accessibility Features	Yes	The platform included both light and dark modes, support for larger fonts, and colour-blind-friendly colour palettes. These features were accessible from the user profile page, allowing individuals to tailor the interface to their needs. This ensures inclusivity and aligns with the requirement of making the app accessible to a wide range of users, including those with visual impairments.