Module Guide for Software Engineering

Team 6, Pitch Perfect
Damien Cheung
Jad Haytaoglu
Derek Li
Temituoyo Ugborogho
Emma Wigglesworth

January 17, 2025

1 Revision History

Date	Version	Notes
Dec 23	1.0	Module Decomp
Jan 3	1.1	Added Type Modules
Jan 14	1.2	Revised diagram to acknowledge Record Modules
Jan 17	1.3	Added User Interfaces

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Software Engineering	Explanation of program name
UC	Unlikely Change
RBAC	Role-Based Access Control
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
SSL	Secure Sockets Layers
TLS	Transport Layer Security
JWT	JSON Web Security
API	Application Programming Interface

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: Enhanced Authentication Features.

AC2: Session Management Updates.

AC3: Database Schema Adjustments.

AC4: User Interface (UI) Enhancements.

AC5: Integration of New Scheduling Algorithms.

AC6: Data Security and Privacy Updates.

AC7: Improved Notification System.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Complete Redesign of the Technology Stack.

UC2: Elimination of Authentication Features.

UC3: Replacement of Database Storage.

UC4: Removal of Role-Based Access Control (RBAC).

UC5: Simplification of Scheduling Logic.

UC6: Discontinuation of Encryption Standards.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: User Interface Module

M2: Authentication Module

M3: Team Management Module

M4: Game Management Module

M5: Announcements Module

M6: Standings Module

M7: Scheduling Module

M8: Waiver Module

M9: PlayerT Module

M10: TeamT Module

M11: GameT Module

M12: Notification Module

M13: Backend Module

M14: Scheduling Algorithm Module

M15: GameslotT Module

M16: DivisonT Module

M17: StandingT Module

M18: SeasonT Module

M19: Reschedule Request Module

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2.

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	User Interface Module
	Authentication Module
	Team Management Module
	Game Management Module
	Announcements Module
	Scheduling Module
	Standings Module
	Waiver Module
	PlayerT Module
	TeamT Module
	GameT Module
Software Decision Module	Database Module
	Notification Module
	Scheduling Algorithm Module

Table 1: Module Hierarchy

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *Software Engineering* means the module will be implemented by the Software Engineering software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Behaviour-Hiding Module

7.1.1 User Interface Module (M1)

Secrets :

- The structure of inputs (e.g., username, password, team names, game scores, and

scheduling information).

- User interface components, logic, and states.

Services :

- Rendering views, collecting user input, and displaying data retrieved from the backend.
- Backend interactions to fetch data and send requests.
- Manages user role-based views.

Implemented By: Team 6

Type of Module: Interface.

7.1.2 Authentication Module (M2)

Secrets: The implementation details of password encryption, token generation, and user session management.

Services: Handles the following:

- User authentication, including login, registration, and logout.
- Verification of user credentials and assignment of roles (player, captain, administrator).
- Management of access tokens for secure API interactions.

Implemented By: Team 6

Type of Module: Abstract Object

7.1.3 Team Management Module (M3)

Secrets :

- Team creation for Captains and team joining joining processes and data for players.
- Waiver acceptance states and data.

Services :

- Handles captain input for team creation.
- Manages user-team associations (joining teams), and processing waiver sign-offs.
- Provides interface for captains to manage their teams (e.g., adding/removing players) and roster validation.

Implemented By: Team 6

Type of Module: Abstract Object

7.1.4 Game Management Module (M4)

Secrets :

- The structure of the input data includes game details (game ID, teams, scores, and status).
- Game record verification.
- Game statuses processes (e.g., scheduled, completed, canceled).

Services :

- Enables game score reporting and updating game statuses for appropriate roles.
- Updates store and retrieve game results to trigger standing updates.

Implemented By: Team 6

Type of Module: Abstract Object

7.1.5 Announcements Module (M5)

Secrets: The structure of announcements, including title, message content, timestamps, as well as how announcements are stored, retrieved, and delivered to users, notifying them.

Services :

- Allows admin to create, edit, and delete announcements for the league.
- Enables users to view announcements and ensures announcnement delivery via notifications.

Implemented By: Team 6

Type of Module: Abstract Object

7.1.6 Standings Module (M6)

Secrets: The formulas for calculating rankings, win-loss records, and tie-breaking rules.

Services: Handles the following:

- Tracks and updates league standings based on game results.
- Displays rankings, win-loss records, and statistical summaries for all teams.

- Supports tie-breaking scenarios and rankings by division.

Implemented By: Team 6

Type of Module: Abstract Object

7.1.7 Scheduling Module (M7)

Secrets: Schedule management, including data structures and communication with the scheduling algorithm.

Services :

- Handles game schedule creation and updates.
- Manages scheduling constraints like team preferences.
- Integrates with the Scheduling Algorithm Module for optimization.

Implemented By: Team 6

Type of Module: Abstract Object

7.1.8 Waiver Module (M8)

Secrets: Waiver completion data.

Services :

- Manages waiver access and completion.
- Tracks waiver status for players.
- Integrates with user accounts to validate players.

Implemented By: Team 6

Type of Module: Abstract Object

7.1.9 PlayerT Module (M9)

Secrets: The data structure used to represent a player.

Services: Holds the data of a player.

Implemented By: Team 6

Type of Module: Abstract Data Type

7.1.10 TeamT Module (M10)

Secrets: The data structure used to represent a team.

Services: Holds the data of a team.

Implemented By: Team 6

Type of Module: Abstract Data Type

7.1.11 GameT Module (M11)

Secrets: The data structure used to represent a game.

Services: Holds the data of a game.

Implemented By: Team 6

Type of Module: Abstract Data Type

7.1.12 GameslotT Module (M15)

Secrets: The data structure used to represent a game slot.

Services: Holds information about game time, field, date.

Implemented By: Team 6

Type of Module: Abstract Data Type

7.1.13 DivisionT Module (M16)

Secrets: The data structure used to represent a division.

Services: Holds information about teams and standings within a division.

Implemented By: Team 6

Type of Module: Abstract Data Type

7.1.14 StandingT Module (M17)

Secrets: The data structure used to represent a team's standings.

Services: Tracks wins, losses, ties, and other metrics for ranking for playoffs.

Implemented By: Team 6

Type of Module: Abstract Data Type

7.1.15 SeasonT Module (M18)

Secrets: The data structure used to represent a season.

Services: Holds information about teams, divisions, games, and timeframes for a season.

Implemented By: Team 6

Type of Module: Abstract Data Type

7.1.16 Reschedule Request Module (M19)

Secrets: The data structure used to represent a reschedule request.

Services: Allows for creating and tracking requests to reschedule games.

Implemented By: Team 6

Type of Module: Abstract Data Type

7.2 Software Decision Module

7.2.1 Notification Module (M12)

Secrets: The protocols for sending important league updates via email notifications.

Services: Sends notifications to users for important updates, such as schedule changes, game results, or captain messages.

Implemented By: External Library

7.2.2 Backend Module (M13)

Secrets: The architecture, database schema, and integration logic for backend services.

Services :

- Manages server-side logic and handles API requests and responses.
- Database to storage and data retrieval (e.g., user data, schedules, game results, etc).
- Secure communication between the frontend and backend.

Implemented By: Team 6

7.2.3 Scheduling Algorithm Module (M14)

Secrets: The algorithm used to generate and optimize schedules, including constraints like team preferences, field availability, etc.

Services :

- Generates game schedules based on team inputs and league constraints for admin use.
- Supports conflict resolution for rescheduling (e.g., double bookings, unavailable fields).
- Optimizes schedules to balance fairness and preferences across teams.

Implemented By: Team 6

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
R1	M1, M2, M13
R2	M1, M2, M13
R3	M1, M7, M14
R4	M1, M6, M13
R5	M1, M3, M13
R6	M1, M3, M8, M13
R7	M1, M3, M7, M14, M13
R8	M1, M7, M14
R9	M1, M5, M12
R10	M1, M7, M14
R10	M1, M7, M14
R12	M1, M4, M13

Table 2: Trace Between Requirements and Modules

Note: The specific record modules are not included in traceability but are outlined in MIS.pdf

AC	Modules
AC1	M1, M2, M13
AC2	M1, M2, M13
AC3	M2, M3, M6, M13
AC4	M1, M12
AC5	M7, M14
AC6	M2, M6, M13
AC7	M12, M13

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

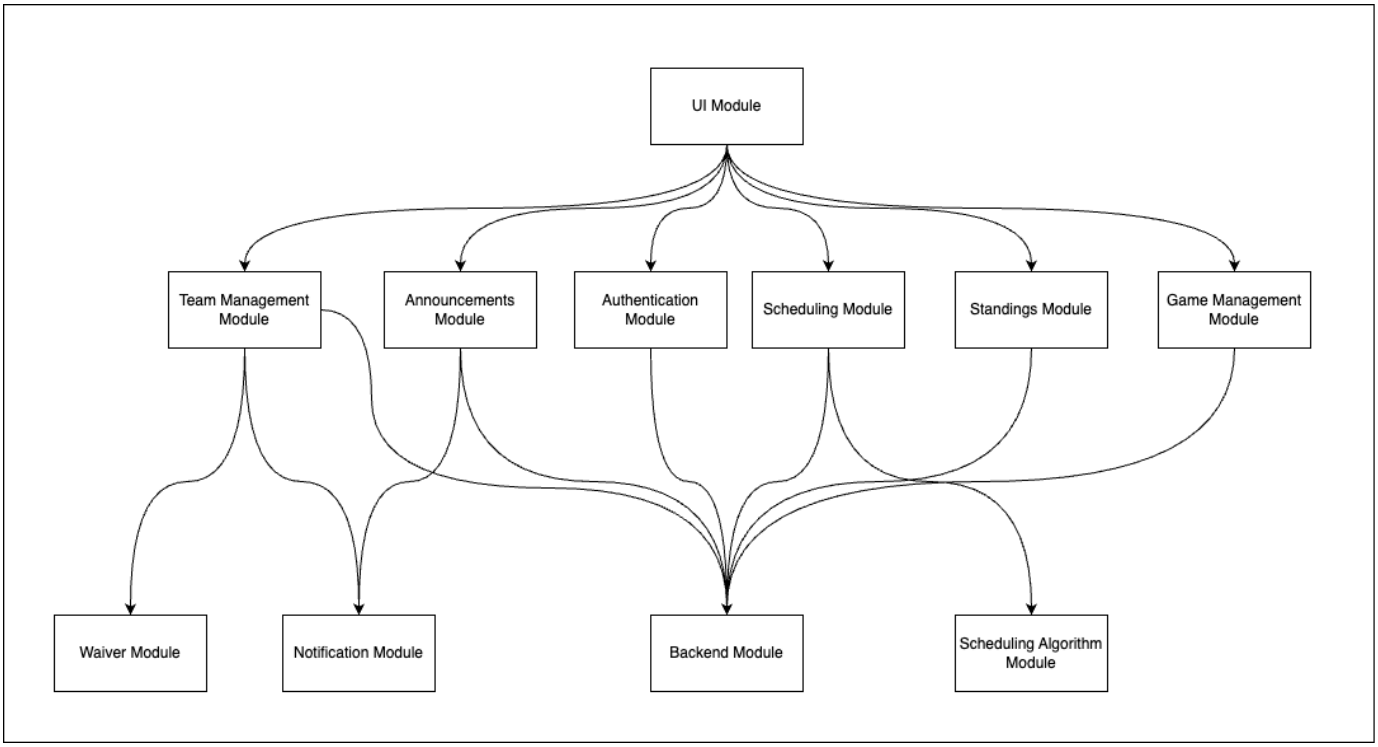

Figure 1: Use hierarchy among modules

Note: Backend Module uses several record modules that represent data entities including Game, Type, Player, Season, Standing, Division, etc. The schemas for these record modules will be further specified in the MIS.pdf

10 User Interfaces

The user interface for the McMaster Graduate Softball Association's platform has been designed to ensure usability, accessibility, and responsiveness. The prototype can be viewed on Figma here. The sections below provide an overview of the key views in the system with associated screenshots.

10.1 Login View

The login view allows users to securely log in to the system using their credentials. The interface is simple and intuitive, ensuring accessibility for all users.

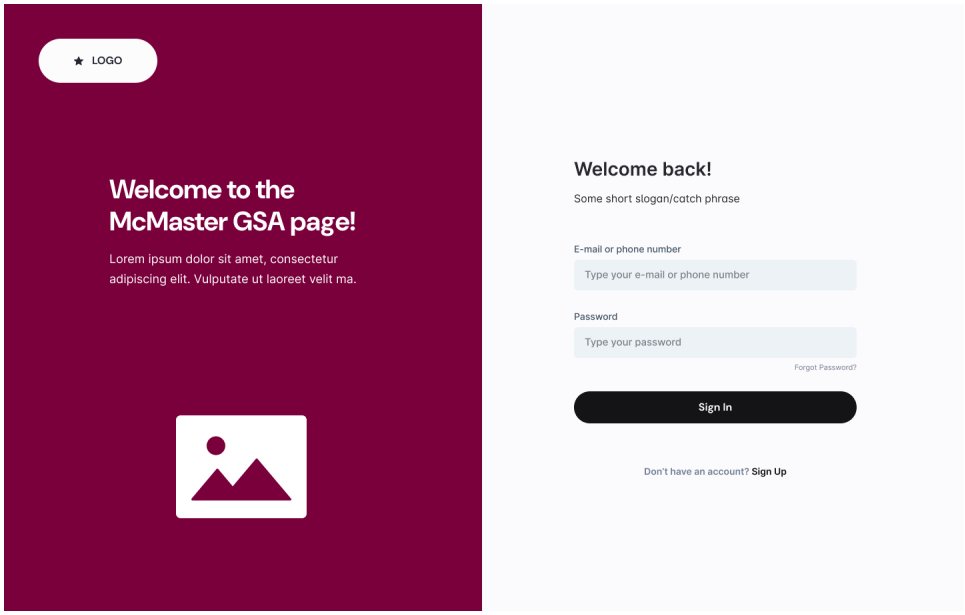

Figure 2: Login View

10.2 Homepage View

The homepage view provides users with a dashboard that displays key information, such as upcoming games, announcements, and navigation options.

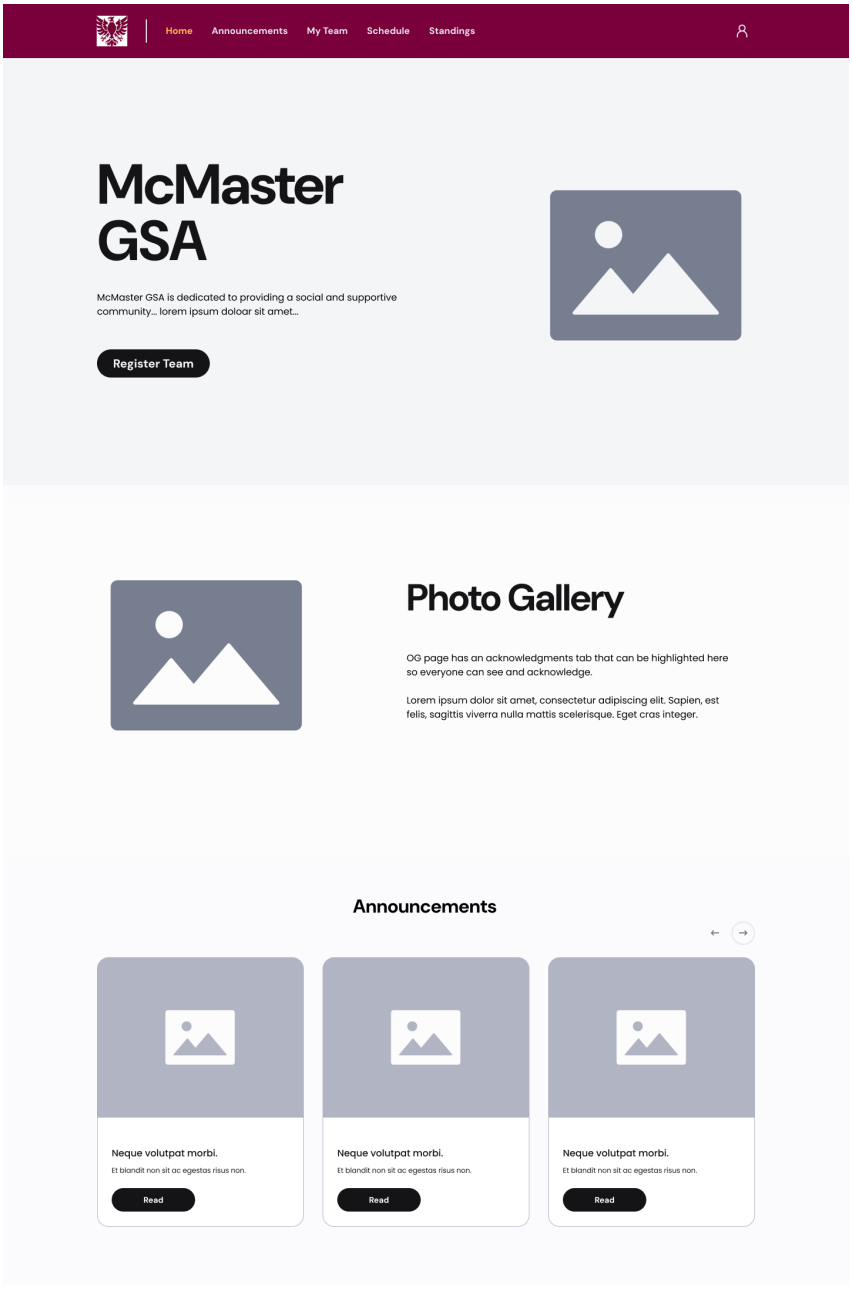

Figure 3: Homepage View

10.3 Team View

The team management view players to view their team information, including upcoming games and current statistics.

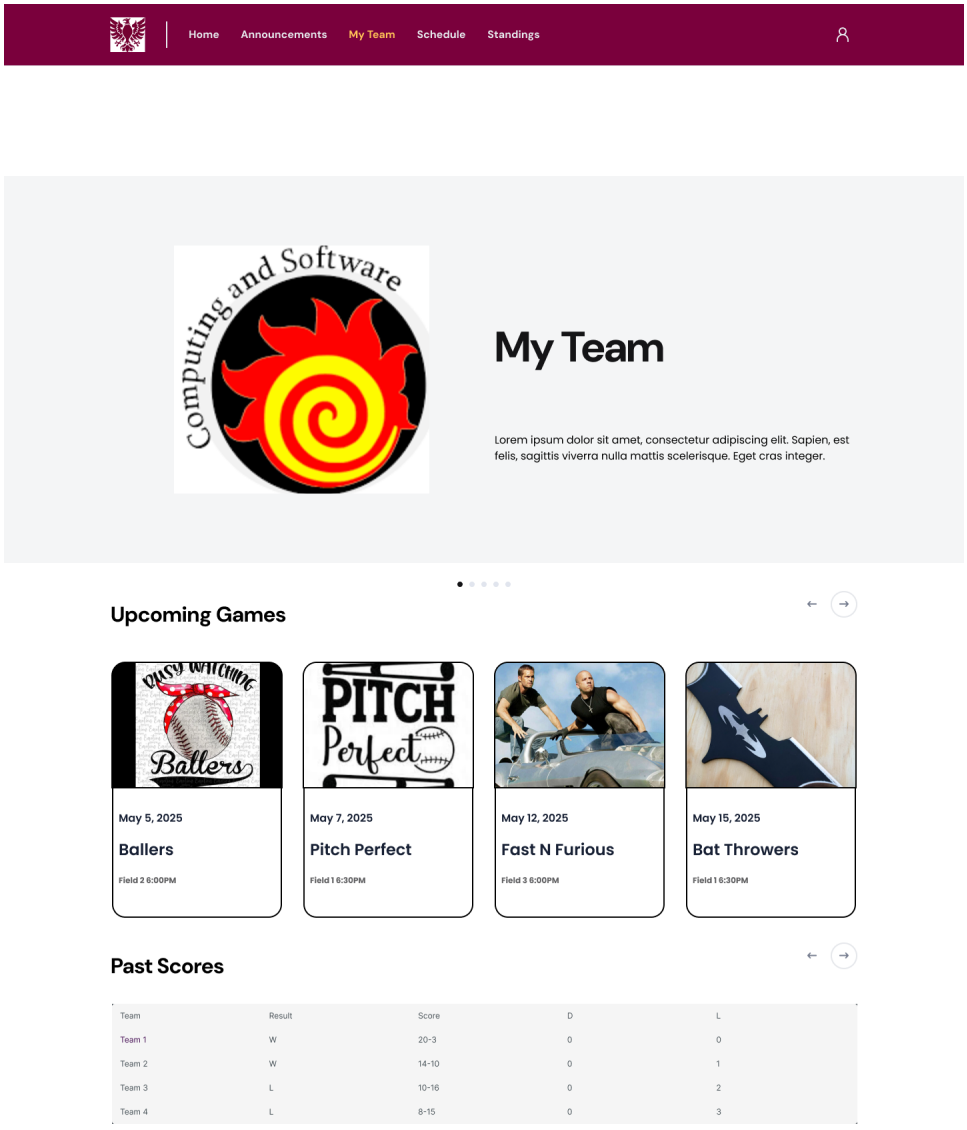

Figure 4: Team Management View

10.4 Scheduling View

The scheduling view allows users to view the season's game schedule.

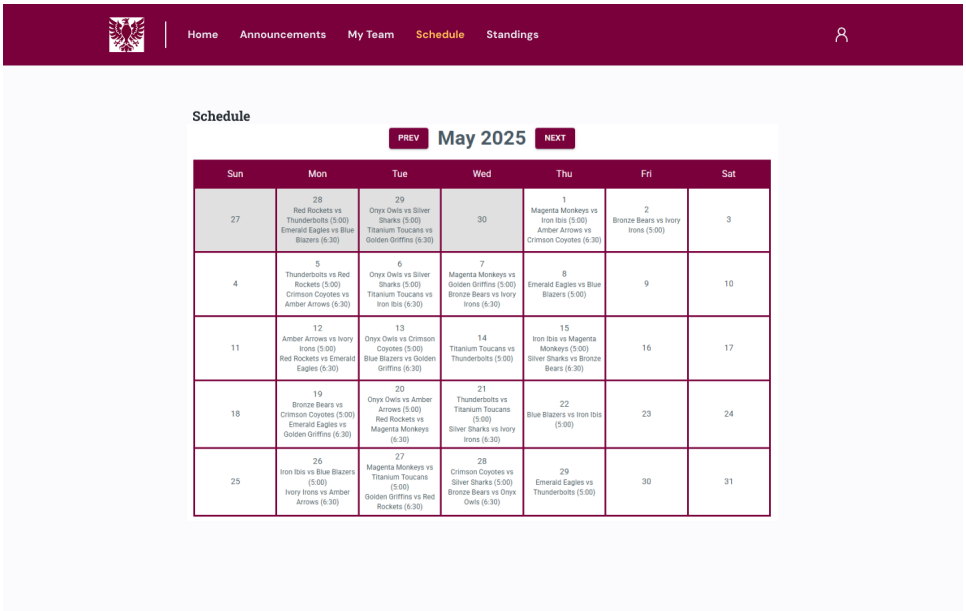

Figure 5: Scheduling View

10.5 Standings View

The standings view displays the league rankings, win-loss records, and statistical summaries for all teams.

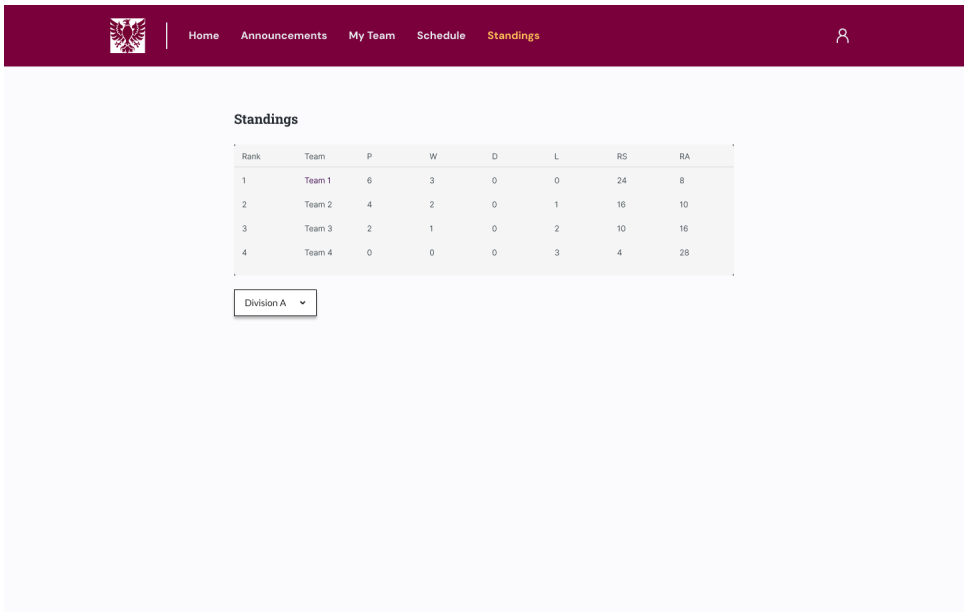

Figure 6: Standings View

10.6 Announcements View

The announcements view is used to view league-wide updates and messages.

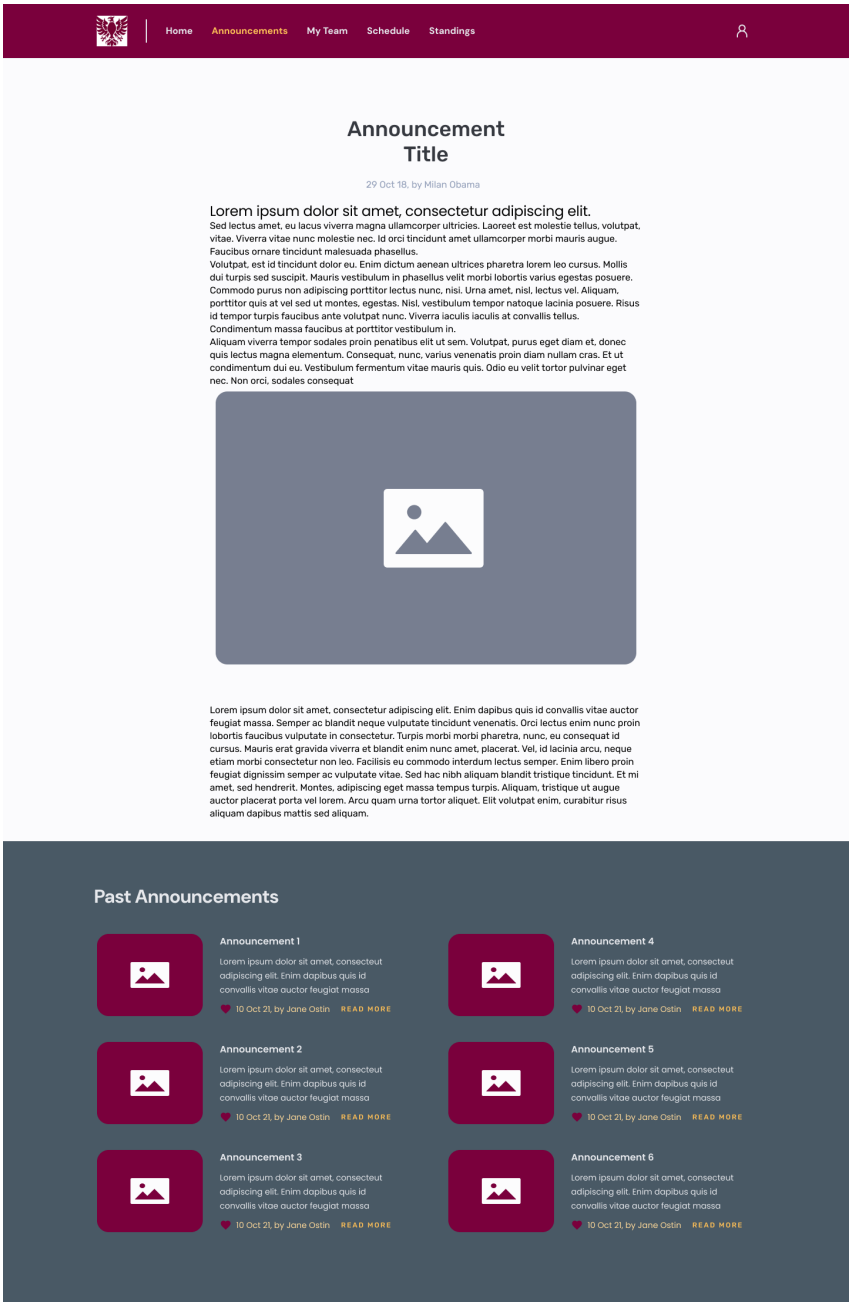

Figure 7: Announcements View

11 Design of Communication Protocols

Overview

The communication protocols for the GSA Softball League platform are designed to ensure secure, reliable, and efficient interaction between system components and users. The platform employs a combination of RESTful APIs, web protocols, and encryption techniques to facilitate communication between the frontend, backend, and database, as well as external services.

Protocols

Frontend to Backend Communication

- **Protocol**: HTTPS (Hypertext Transfer Protocol Secure)
- **Description**:
  - All communication between the frontend and backend is secured using HTTPS, ensuring data integrity and confidentiality.
  - RESTful APIs handle requests such as login, data retrieval, schedule updates, and notifications.
- **Endpoints**:
  - `/api/login`: Handles user authentication.
  - `/api/schedule`: Retrieves game schedules.
  - `/api/report`: Submits match results.

Backend to Database Communication

- **Protocol**: MongoDB Wire Protocol (Over SSL/TLS)
- **Description**:
  - Secure communication between the backend and the MongoDB database is facilitated by the MongoDB Wire Protocol over SSL/TLS.
  - Queries include CRUD operations for users, teams, schedules, and standings.

Backend to External Services

- **Protocol**: HTTPS with Third-Party APIs
- **Description**:

  - External services such as email or SMS notifications (e.g., Twilio, SendGrid) communicate securely using HTTPS.
  - APIs for notification services handle user alerts, such as schedule changes or reminders.

Client to Client Communication

- **Protocol**: WebSockets (Optional)
- **Description**:
  - Real-time updates for users, such as live score tracking or notifications, can be implemented using WebSocket connections.

User Authentication Protocol

- **Protocol**: JWT (JSON Web Tokens) over HTTPS
- **Description**:
  - Upon successful login, the backend issues a JWT to the client.
  - JWTs are used to authenticate subsequent requests securely, reducing the need to reauthenticate for every action.

Error Handling and Fault Tolerance

- **Timeouts**:
  - API requests have a timeout of 30 seconds to avoid hanging requests.
  - The frontend displays appropriate error messages if a timeout occurs.
- **Retries**:
  - Failed requests to external services are retried up to three times with exponential backoff.
- **Error Codes**:
  - Standardized HTTP status codes are used for API responses:
    * `200`: Success
    * `400`: Bad Request
    * `401`: Unauthorized
    * `403`: Forbidden
    * `404`: Not Found
    * `500`: Internal Server Error

Security Measures

- **Data Encryption**: All communication is encrypted using SSL/TLS to prevent data interception.
- **Authentication Tokens**: JWTs are signed and validated to ensure they are tamper-proof.
- **Input Validation**: All user inputs are validated to prevent injection attacks.
- **Access Control**: Role-based access control (RBAC) ensures that only authorized users can perform specific actions.

12 Timeline

Note: Weeks 1 refers to the beginning of the year 2025

Phase 1: Documentation and Initial Setup (Week 1)

1. **Week 1: Finalization of Documentation**
    - **Tasks**:
        - Complete the **Module Interface Specification (MIS)**.
        - Finalize the **Module Guide**.
        - Create the initial sketches for the **User Interface.**
        - Create the **Design of Communication Protocols**.
    - **Responsible Members**:
        - **Damien Cheung:** Module Hierarchy, Module Decomposition, First Traceability Matrix, Use Hierarchy Between Modules, GameT Module, TeamT Module, PlayerT Module, Backend/Database Module.
        - **Temituoyo Ugborogho:** Anticipated and Unlikely Changes, Second Traceability Matrix, Design of Communication Protocols, Timeline, Authentication Module.
        - **Emma Wigglesworth:** User Interface, Scheduling Module, Scheduling Algorithm Module
        - **Derek Li:** Notification Module, Announcement Module, Team Management Module, Waiver Module.
        - **Jad Haytaoglu:** Game Management Module, User Interface Module.
        - **All Members:** Review and finalization of documents.

Phase 2: Core Module Development (Week 2)

1. **Week 2: Core Module Development**

   - **Tasks**:
     - Initialize and develop the **Database**.
     - Develop the **Authentication Module** (login, session management, token generation).
     - Create the **Scheduling Module** (game allocation logic).
     - Implement the **Team Management Module** (team creation, roster management).
     - Conduct preliminary unit testing for each core module.
   - **Responsible Members**:
     - Damien Cheung and Jad Haytaoglu: Database.
     - Temituoyo Ugborogho: Authentication Module.
     - Emma Wigglesworth: Scheduling Module.
     - Derek Li: Team Management Module.

Phase 3: Supporting Module Development and Integration (Week 3)

1. **Week 3: Supporting Module Development and Integration**

   - **Tasks**:
     - Develop the **Notification Module** (alerts for schedule changes, announcements).
     - Implement the **Game Management Module** (game data handling, score reporting).
     - Build the **User Profile Module** (managing user preferences and settings).
     - Integrate all developed modules (core and supporting) into a unified system.
     - Conduct basic system-level integration tests.
   - **Responsible Members**:
     - Derek Li: Notification Module Development.
     - Jad Haytaoglu: Game Management Module and User Profile Module Development.
     - Damien Cheung: Backend and Database Integration.
     - All Members: Collaborative Integration Testing.

Phase 4: System Testing and Final Refinements (Week 4)

1. **Week 4: System Testing and Final Refinements**

   - **Tasks**:
     - Conduct end-to-end system testing for all modules.
     - Validate role-based access control (RBAC) functionality.
     - Validate encryption protocols for secure data transfer (in transit and at rest).
     - Prepare deployment documentation, including user guides and system setup instructions.
     - Conduct a final review of traceability matrices and ensure alignment with anticipated changes.
   - **Responsible Members**:
     - Damien Cheung and Jad Haytaoglu: Final Database and Backend Testing.
     - Damien Cheung and Temituoyo Ugborogho: Final Communication Protocols and Traceability Documentation.
     - Emma Wigglesworth, Temituoyo Ugborogho and Derek Li: Collaborative System Testing.
     - All Members: Deployment Review.

Key Details

- **Documentation**:
  - Week 1 is entirely focused on finalizing all project documentation to ensure clarity and direction for subsequent development phases.
- **Testing and Verification**:
  - Unit testing will be performed for each module during development.
  - Integration testing will be conducted immediately after module integration in **Week 3**.
  - End-to-end testing will occur in **Week 4** to ensure Rev 0 readiness.

Table of Responsibility To Date

Team Member	Responsibility
Emma Wigglesworth	User Interface Scheduling Module Scheduling Algorithm Module
Derek Li	Notification Module Announcement Module Team Management Module Waiver Module
Jad Haytaoglu	Game Management Module User Interface Module
Damien Cheung	Module Hierarchy Module Decomposition First Traceability Matrix Use Hierarchy Between Modules GameT Module TeamT Module PlayerT Module Backend/Database Module
Temituoyo Ugborogho	Anticipated and Unlikely Changes Second Traceability Matrix Design of Communication Protocols Timeline Authentication Module

More info can be found here.

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.