Verification and Validation Report: Software Engineering

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

March 10, 2025

1 Revision History

Date	Version	Notes
March 1	1.0	Created Document Structure and Outline. Filled out most test cases and appendix reflection.
March 8	1.1	Unit Test report
March 8	1.2	Add more Unit Tests, System Tests, Reflection
March 8	1.3	Add test-requirement traceability

2 Symbols, Abbreviations, and Acronyms

symbol	description
T	Test
UT	Unit Test
RBAC	Role-Based Access Control. Used for defining user permissions
UI	User Interface
JWT	JSON Web Token. A compact way to transmit info securely
GSA	Graduate Students' Association. The association overseeing the league
SRS	Software Requirements Specification
MIS	Module Interface Specification
DD	Design Document
VnV	Verification and Validation
CI	Continuous Integration
API	Application Programming Interface. Defines how software components interact
CRUD	Create, Read, Update, Delete. Basic operations for managing data
DB	Database. A system for storing and managing data

2.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

Parameter	Value	Unit	Description
`MAX_NAVTIME`	60	s	The maximimum time acceptable for testers to navigate to a main view
`MIN_TESTERS`	5	n/a	The minimum number of testers required for a system test
`AVG_TASK_TIME`	3	min	The average task time for a task completion

MAX_GAME_DIFF	2	n/a	The maximum difference in total scheduled games between two teams

Contents

List of Tables

List of Figures

This document ...

3 Functional Requirements Evaluation

3.1 Manual Testing

1. **FR-1**
    - Description: Testing login functionality with wrong credentials.
    - How Test Will Be Performed: Tester will attempt to log in with an invalid `UserID` and `Password`. Verify that all invalid logins are unsuccessful.
    - Requirements Covered: SRS Req #1 (authentication).
    - Inputs: Invalid UserID and Password
    - Expected Output: Failed to login
    - Actual Output: Failed to login
    - Result: Pass
2. **FR-2**
    - Description: Testing login functionality with correct credentials.
    - How Test Will Be Performed: Tester will log in with a valid `UserID` and `Password` for each `Role` (Player, Captain, Commissioner) and verify access to the platform after successful login under the correct account.
    - Requirements Covered: SRS Req #1 (authentication).
    - Inputs: `UserID` and `Password` for valid commissioner, captain, and player accounts.
    - Expected Output: Successful login for valid credentials.
    - Actual Output: Successful login for valid credentials.
    - Result: Pass
3. **FR-3**
    - Description: Attempting to create a duplicate team name

- How Test Will be Performed: Tester will attempt to create a new team suing a duplicate `TeamName`. Verify that the team is not added to the database.
- Requirements Covered: SRS Req #5 (team creation).
- Inputs: Enter a `TeamName` that already exists in the database and a Division.
- Expected Output: Unsuccessful creation of a team with a duplicate
- Actual Output: Unsuccessful creation of a team with a duplicate
- Result: Pass

4. **FR-4**

   - Description: Creating a new team with a unique name.
   - How Test Will Be Performed: Tester will attempt to create a new team using a unique `TeamName` and Division. Verify that the team is successfully added to the database.
   - Requirements Covered: SRS Req #5 (team creation).
   - Inputs: Enter a unique `TeamName` and Division.
   - Expected Output: Successful team creation.
   - Actual Output: Successful team creation.
   - Result: Pass

5. **FR-5**

   - Description: Attempting to create multiple teams under one Captain.
   - How Test Will Be Performed: Tester will attempt to create a second team while logged in as a Captain who already has an existing team. Verify that the new team is not added to the database.
   - Requirements Covered: SRS Req #5 (team creation).
   - Inputs: Enter a new `TeamName` and Division while already associated with an existing team.
   - Expected Output: Unsuccessful team creation. Error message displayed for multiple teams under one `CaptainID`.

- Actual Output: Not Tested Yet.
- Result: Not Tested Yet.

6. **FR-7**

   - Description: Generating a season game schedule based on team availability and preferences.
   - How Test Will Be Performed: Load the database with team availability, divisional assignments, game count requirements, and available slots. Run the schedule generation process. Refer to TPERF-2 for performance testing of the generated schedule.
   - Requirements Covered: SRS Req #7 (Schedule Automation).
   - Inputs:
     - Team Availability: Preferences for game days and times.
     - Division Assignments: Each TeamID is assigned a division.
     - Game Count Requirement: Ensures each team has an equal number of games.
     - Available Slots: The availability of the event space.
   - Expected Output: The system generates an appropriate season game schedule.
   - Actual Output: The system successfully generates a season game schedule.
   - Result: Pass

7. **FR-8**

   - Description: Captains submitting game results, which update the standings.
   - How Test Will Be Performed: Captain submits game scores (ScoreTeamA, ScoreTeamB). Verify that the report is reflected in the database and that standings update accordingly.
   - Requirements Covered: SRS Req #12 (Reporting).
   - Inputs:
     - ScoreTeamA: Score for Team A.
     - ScoreTeamB: Score for Team B.

- Expected Output: Game report is updated, and standings reflect the new results.
- Actual Output: Game report successfully updates, and standings are correctly modified.
- Result: Pass

8. **FR-9**

   - Description: Captains submitting reschedule requests for approval by the Commissioner.
   - How Test Will Be Performed: Captain initiates a reschedule request for an open slot. Verify that the request is sent to the Commissioner for approval.
   - Requirements Covered: SRS Req #8 (Rescheduling).
   - Inputs:
     - ScheduleID: Identifier for the current schedule.
     - GameID: Identifier for the game being rescheduled.
     - SlotNumber: Open slot requested for rescheduling.
   - Expected Output: Request is successfully sent to the Commissioner for review.
   - Actual Output: Not Tested Yet.
   - Result: Not Tested Yet.

9. **FR-10**

   - Description: Commissioner overriding a scheduled game (cancel or reschedule) or approving a reschedule request.
   - How Test Will Be Performed: Commissioner applies an override on an existing game. Verify that the schedule updates correctly (Schedule view, SlotNumber, and Available Slots). Confirm that notifications are sent to all affected team members.
   - Requirements Covered: SRS Req #11 (Game Overrides).
   - Inputs:
     - GameID: Identifier for the game being overridden.

  - Override Type: Cancel or reschedule.
  - New SlotNumber (if rescheduling): The newly assigned game slot.
- Expected Output: The override is reflected in the schedule, and all rostered team members receive notifications.
- Actual Output: Not Tested Yet.
- Result: Not Tested Yet.

10. **FR-11**

- Description: Commissioner posts a league-wide announcement that should be visible across all platform views.
- How Test Will Be Performed: Commissioner submits an announcement. Verify that the announcement is displayed to all users across the platform.
- Requirements Covered: SRS Req #9, #10 (Announcements).
- Inputs:
  - Announcement Title: The subject of the announcement.
  - Announcement Body: The message content.
  - Visibility Scope: League-wide.
- Expected Output: Announcement is displayed across all platform views and is accessible to all users.
- Actual Output: Announcement successfully visible to all users.
- Result: Pass

3.2 System Testing

1. **FR-1**

- **Description:** Test login functionality with valid credentials.
- **Test Method:** Automated test using `playerControllers.test.js` to create a player account and log in with valid credentials for each `Role` (Player, Captain, Commissioner).
- **Expected Result:** Successful login for each role and access to platform functionalities.

- **Result:** Pass

2. **FR-2**

   - Description: Testing login functionality with correct credentials.
   - How Test Will Be Performed: A backend script in 'playerControllers.test.js' will create a Player account with a `UserID` and `Password`. The tester will log in with a valid `UserID` and `Password` for each `Role` (Player, Captain, Commissioner) and verify access to the platform after successful login under the correct account. Test data will be created in the database and cleaned up after the test.
   - Requirements Covered: SRS Req #1 (authentication).
   - Result: Pass

3. **FR-3**

   - **Description:** Create a new team with a unique name.
   - **Test Method:** Automated test using `teamControllers.test.js` to input a unique `TeamName` and Division, and create a team.
   - **Expected Result:** The team should be successfully added to the database.
   - **Result:** Pass

4. **FR-4**

   - **Description:** Create a new team with an existing name.
   - **Test Method:** Automated test using `teamControllers.test.js` to input an exisiting `TeamName` and Division, and create a team.
   - **Expected Result:** The team should be successfully added to the database.
   - **Result:** Fail

5. **FR-5**

   - **Description:** Test system behavior when attempting to create multiple teams under a single Captain.

- **Test Method:** Automated test using `teamControllers.test.js` to create multiple teams under one Captain and verify error handling.
- **Expected Result:** The second team should not be created, and an error message should appear.
- **Result:** Fail

4 Nonfunctional Requirements Evaluation

4.1 Usability

4.1.1 Usability Survey Results

As part of our usability evaluation, we conducted a survey with participants who tested the platform for the first time. Testers were asked to complete key tasks such as signing up, logging in, creating a team, requesting reschedules, and navigating the interface. After completing these tasks, they provided feedback via a post-task survey.

The full survey can be accessed at: Google Forms Usability Survey.

Table 2 summarizes the responses of `MIN_TESTERS` participants (Response IDs 0–4) to our usability questionnaire. Each row corresponds to one survey question.

Table 2: Usability Survey Questions and Responses (Updated Data)

Question	**Resp. 0**	**Resp. 1**	**Resp. 2**	**Resp. 3**	**Resp. 4**
How easy was it to learn how to use the platform?	5	4	5	5	5

Continued on next page

Table 2 (continued)

Question	**Resp. 0**	**Resp. 1**	**Resp. 2**	**Resp. 3**	**Resp. 4**
How confident do you feel using the platform without assistance after your initial experience?	5	5	5	5	5
How intuitive do you find the navigation of the platform?	5	5	5	5	5
How long did it take you to navigate all the views? (Approx. in minutes)	3	5	2	2	3
How easy was it to complete basic tasks (login, create team, request reschedule)?	4	4	5	5	5
Did you encounter any errors or issues while using the platform? If so, please describe them.	*(none)*	Invalid email accepted. Misspelled @gmail.com and it still allowed it.	*(none)*	NA	*(none)*

Continued on next page

Table 2 (continued)

Question	Resp. 0	Resp. 1	Resp. 2	Resp. 3	Resp. 4
How helpful were the in-app help resources in learning how to use the platform?	5	5	3	5	5
Please share any suggestions for improving the in-app help resources.	4	*(none)*	i didn't need them	NA	*(none)*
Did the platform's interface appear modern, intuitive, and visually consistent across all views?	the use of yellow randomly appears only in a couple pages	4	5	5	5
Did you observe any remarkable inconsistencies in the visual interface when navigating (login, team mgmt, schedule)?	the use of yellow randomly appears only in a couple pages	*(none)*	*(none)*	NA	maybe some fonts

Continued on next page

Table 2 (continued)

Question	**Resp. 0**	**Resp. 1**	**Resp. 2**	**Resp. 3**	**Resp. 4**
How accessible do you find the platform in terms of visual design (color contrast, text size, font)?	4	5	5	5	5
Do you have any suggestions for improving the accessibility of the platform?	yellow might be hard for some people to see	*(none)*	maybe bigger fonts in some places	NA	*(none)*
Based on the Canadian formatting standards provided, do you believe the platform adheres to these conventions?	Yes	Yes	Yes	Yes	Yes
If you noticed any deviations from Canadian formatting standards, please describe them.	*(none)*	*(none)*	*(none)*	NA	i didn't notice

Continued on next page

Table 2 (continued)

Question	Resp. 0	Resp. 1	Resp. 2	Resp. 3	Resp. 4
Would you like to provide any additional comments on your experience?	*(none)*	I like overall that the colours adhere to McMaster	super easy to use	NA	i like it a lot

The usability survey results indicate that the platform is highly intuitive and easy for new users to learn and navigate. The responses were overwhelmingly positive, with all participants successfully completing tasks such as signing up, logging in, and exploring key sections of the site. Navigation times were within the expected range, and testers generally found the interface modern, visually consistent, and user-friendly.

One minor concern was that a participant rated the in-app help resources lower than others. However, this response likely reflects the fact that the help resources were unnecessary rather than inadequate, which is ultimately a positive outcome. Another issue noted was an invalid email being accepted during sign-up. While this can be corrected in the edit profile feature, the team will consider implementing stricter validation on the registration page to prevent similar cases.

Overall, the results confirm that the platform meets usability expectations and is on track for successful deployment, with only minor refinements needed for edge cases.

4.1.2 Manual Testing

1. **TUH-1**

    - Description: Evaluate ease of navigation for new users by measuring time taken to access key platform views.
    - How Test Will Be Performed: Conduct an observational usability study where participants navigate through login, announcements, standings, and schedule pages. Record navigation times and collect feedback through a post-task survey.

- Requirements Covered: SRS Req #11.1 (Usability - Navigation).
- Inputs:
  - Navigation Tasks: Tester navigates through login, announcements, standings, and schedule pages.
  - Participants: MIN_TESTERS with no prior experience with the platform.
- Expected Output: Participants successfully navigate to each view within MAX_NAVTIME without prior experience or external assistance.
- Actual Output: Participants completed navigation within the expected time.

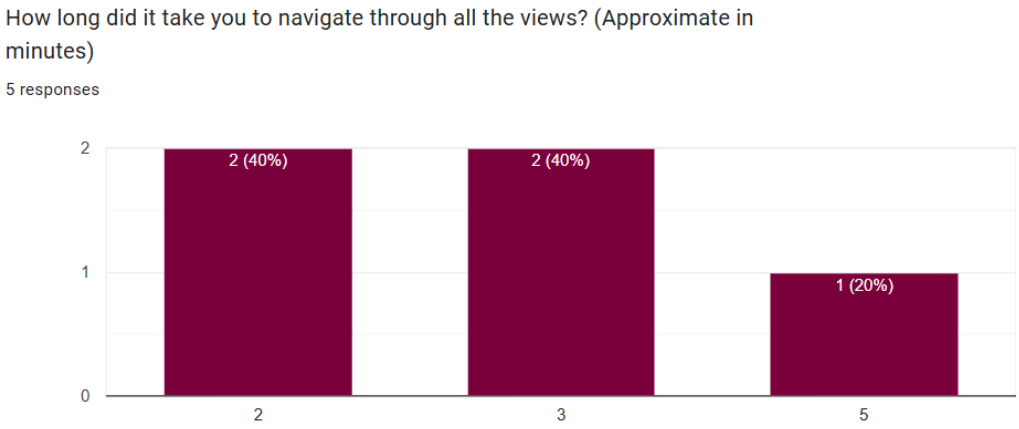

Figure 1: Survey results showing participants' average navigation times.

- Result: Pass

2. **TUH-2**

- Description: Verify the platform displays date, time, and measurements according to Canadian localization standards.
- How Test Will Be Performed: Verify date, time, and measurement units across the platform views. Ask usability testers to confirm that information formats are clear and match Canadian standards. Feedback survey included in Appendix.

- Requirements Covered: SRS Req #11.2 (Localization - Canadian Standards).
- Inputs:
  - Date, Time, and Measurement Fields: Tester reviews the date, time, and metric system information across the platform.
  - Participants: Usability testers verify format clarity and standard compliance.
- Expected Output: Platform displays date and time in Canadian format, uses the metric system, and all text is in Canadian English.
- Actual Output: Not Tested Yet.
- Result: Not Tested Yet.

3. **TUH-3**

- Description: Evaluate the effectiveness of in-app learning support by measuring how quickly new users can complete tasks using the platform's guidance tools.
- How Test Will Be Performed: Conduct an observational usability study with MIN_TESTERS participants who have no prior experience with the platform. Record the time taken for each task completion. Following the tasks, participants will complete a survey (in Appendix) to provide feedback on task difficulty and provided guidance.
- Requirements Covered: SRS Req #11.3-11.4 (Usability - Task Completion and Learning Support).
- Inputs:
  - Tasks: Tester performs login, creates a team, and requests a reschedule.
  - Participants: MIN_TESTERS with no prior experience with the platform.
- Expected Output: Tester completes each task within an average of AVG_TASK_TIME minutes, utilizing in-app guidance (navigation instructions, tooltips, help documentation) to easily understand and complete tasks.

- Actual Output: Tester completed each task within the expected time and used in-app help effectively.

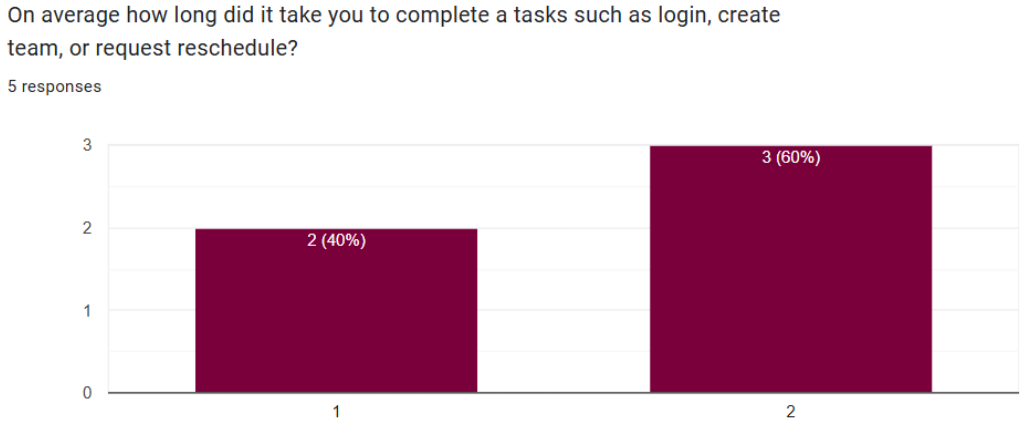

Figure 2: Survey results showing participants' average task completion times.

- Result: Pass

4.2 Performance

4.2.1 Manual Testing

1. **TPERF-1**

   - Description: Verify the accuracy of the league standings calculations.
   - How Test Will Be Performed: Create test cases with predefined standings. Use an automated script to perform calculations and check results against expected outcomes, ensuring all calculations are accurate.
   - Requirements Covered: SRS Req #12.1, 12.3 (Performance - Standings Calculation Accuracy).
   - Inputs:
     - Sample Data: Predefined standings data for the league.
     - Tester: Automated script performing the calculation checks.

- Expected Output: The standings are calculated with 100
- Actual Output: The standings were calculated with 100
- Result: Pass

2. **TPERF-2**

   - Description: Verify the accuracy and balance of the league schedule, ensuring that teams play within their division, with balanced game counts, and no conflicts.
   - How Test Will Be Performed: The platform is prepared with a sample dataset containing multiple teams and generates a schedule. The tester manually verifies the schedule, checking that each team is assigned games within the schedule, ensuring a balanced number of games, verifying divisional match-ups, and logging any conflicts.
   - Requirements Covered: SRS Req #12.4 (Performance - Scheduling Accuracy and Optimization).
   - Inputs:
     - Sample Data: Teams and their preferences for scheduling.
     - Tester: Manually verifying the schedule.
   - Expected Output:
     - The displayed schedule shows that all teams have a balanced number of scheduled games, with no team having a game count differing by more than MAX_GAME_DIFF games from any other team.
     - The displayed schedule shows that all teams play 100
     - The displayed schedule is conflict-free, considering time and location.
     - The displayed schedule is optimized based on matching team preferences.
   - Actual Output: Not Tested Yet.
   - Result: Not Tested Yet.

3. **TPERF-3**

- Description: Verify that new login credentials are properly added and stored in the database.
- How Test Will Be Performed: The tester manually adds a pair of new credentials and checks whether the information is stored correctly in the database.
- Requirements Covered: SRS Req #12.2 (Performance - Credential Storage).
- Inputs:
  - New Login Credentials: A pair of new user credentials to be added to the system.
  - Tester: Manually verifying the database storage.
- Expected Output: The credentials are stored in the database in the correct format.
- Actual Output: The credentials were stored in the database in the correct format.
- Result: Pass

4. **TPERF-4**

- Description: Verify that the platform handles invalid form submissions by highlighting errors and preventing submission until corrections are made.
- How Test Will Be Performed: Testers will intentionally leave required fields blank, enter invalid data formats, and submit the form. Testers will verify that the fields with errors are highlighted visually and that informative error messages appear next to the problematic fields, indicating the nature of the errors. Testers will confirm that the information in the form is not submitted into the database until the errors are corrected and the form is resubmitted, showing a success message upon submission.
- Requirements Covered: SRS Req #12.4 (Performance - Form Validation and Error Handling).
- Inputs:
  - Invalid Data: Required fields left blank, incorrect data formats (e.g., invalid email).

  - Tester: Manually interacting with the form and submitting.
- Expected Output:
  - Fields with errors are highlighted visually.
  - Informative error messages appear next to the problematic fields.
  - Form submission is prevented until all errors are corrected.
  - A success message appears upon a successful resubmission after error correction.
- Actual Output: Not Tested Yet.
- Result: Not Tested Yet.

4.3 Operational and Environmental

4.3.1 Manual Testing

1. **TOPE-1**

   - Description: Verify that the platform is fully accessible and functional across desktop, tablet, and smartphone devices.
   - How Test Will Be Performed: Perform manual testing of the platform features, responsiveness, and navigation on desktop, tablet, and smartphone. Ensure all features are accessible and function correctly on each device type.
   - Requirements Covered: SRS Req #13.1 (Platform Compatibility - Multi-Device Accessibility).
   - Inputs:
     - Device Types: Desktop, tablet, and smartphone.
     - Tester: Manually interacting with the platform across all device types.
   - Expected Output: The platform is fully accessible and functional on all device types, with no loss of features or navigation issues.
   - Actual Output: Not Tested Yet.
   - Result: Not Tested Yet.

2. **TOPE-2**

- Description: Verify that the platform is compatible and displays correctly across major browsers (Chrome, Firefox, Safari, and Edge).
- How Test Will Be Performed: Testers manually access all features and views of the platform in each of the browsers (Chrome, Firefox, Safari, and Edge). Ensure that the platform displays correctly and functions as expected in each browser.
- Requirements Covered: SRS Req #13.2-13.3 (Browser Compatibility).
- Inputs:
  - Browser Types: Chrome, Firefox, Safari, and Edge.
  - Tester: Manually interacting with the platform across each browser.
- Expected Output: The platform displays correctly across all major browsers, with no visual or functional issues.
- Actual Output: Not Tested Yet.
- Result: Not Tested Yet.

4.4 Maintainability and Support

4.4.1 Manual Testing

1. **TMS-1**

   - Description: Verify that users can contact support via email without any external assistance.
   - How Test Will Be Performed: A minimum number of testers (MIN_TESTERS) will be asked to reach support through email. Testers will document any difficulties encountered during the process.
   - Requirements Covered: SRS Req #14.1, 14.2 (Support - Contact Mechanism).
   - Inputs:
     - Support Email: The email address to contact support.
     - Tester: Manually sending an email to the support team.

- Expected Output: The support team receives the help request from the tester via email, with no external help required.
- Actual Output: Not Tested Yet.
- Result: Not Tested Yet.

4.5 Security

4.5.1 Manual Testing

1. **TSEC-1**

   - Description: Verify that users with different roles (Player, Captain, Commissioner) have the appropriate level of access to platform features.
   - How Test Will Be Performed: Manually log in with a dummy user for each role (Administrator, Team Manager, Player, and Captain). Attempt to access various features and verify that the access levels match the defined role-based access control requirements.
   - Requirements Covered: SRS Req #15.1, 15.2 (Security - Role-Based Access Control).
   - Inputs:
     - User Roles: Administrator, Team Manager, Player, and Captain.
     - Tester: Manually logging in with each role and testing access to platform features.
   - Expected Output:
     - The Player role has restricted access, with only basic features (e.g., viewing schedule, joining teams).
     - The Captain role has access to additional team management features.
     - The Commissioner has access to all administrative features.
   - Actual Output: Not Tested Yet.
   - Result: Not Tested Yet.

4.6 Cultural

4.6.1 Manual Testing

1. **TCU-1**

    - Description: Verify that all text is displayed in Canadian English and the time is shown in the correct Hamilton, Ontario, Canada time zone (EST).
    - How Test Will Be Performed: Manually inspect the platform's content, ensuring that all text is displayed in Canadian English and that date/time formats reflect the Hamilton, Ontario time zone (EST).
    - Requirements Covered: SRS Req #16.1 (Content - Localization and Time Zone).
    - Inputs:
        - Platform Content: Text, date, and time information.
        - Tester: Manually inspecting the platform for correctness in language and time zone.
    - Expected Output: All text is displayed in Canadian English, and time is correctly displayed in EST (Hamilton, Ontario).
    - Actual Output: The platform displayed all content in Canadian English, with the correct time zone.
    - Result: Pass

4.7 Compliance

4.7.1 Manual Testing

1. **TC-1**

    - Description: Verify that a user can delete their account and associated personal data, and receive confirmation of deletion.
    - How Test Will Be Performed: Manually delete a dummy user account and verify that all associated personal data (e.g., email, phone number) is removed from the database and platform. Log any unexpected time elapsed to ensure the deletion process is secure and compliant with expected time frames.

- Requirements Covered: SRS Req #17.1 (Functional - Data Deletion and Compliance).
- Inputs:
  - User Account: Dummy account with personal data (email, phone number).
  - Tester: Initiating the account deletion request.
- Expected Output: All personal data associated with the account is removed from the database and platform, and the user receives confirmation of successful deletion.
- Actual Output: Not Tested Yet.
- Result: Not Tested Yet.

2. **TC-2**

- Description: Verify that the platform adheres to W3C web standards, including readable fonts, accessible colors, and clear navigation.
- How Test Will Be Performed: Manually inspect the platform's interface to ensure compliance with W3C web standards, focusing on font readability, color contrast, and navigation clarity.
- Requirements Covered: SRS Req #17.2 (Functional - Web Standards Compliance).
- Inputs:
  - Platform Interface: The visual elements of the platform, including text, colors, and navigation.
  - Tester: Manually inspecting the platform against W3C standards.
- Expected Output: The platform meets W3C standards for font readability, accessible color contrasts, and clear navigation.
- Actual Output: The platform adheres to the W3C web standards.
- Result: Pass

5 Comparison to Existing Implementation

Existing Implementation: GSA Softball Website

- **User Experience:** Our platform provides a more intuitive and streamlined navigation experience, with easy-to-find features like schedules, team management, and announcements.
- **Design and Aesthetics:** Our site boasts a modern, responsive design that adapts seamlessly across devices, offering a clean and visually appealing interface.
- **Functionality:** We provide advanced features such as live score updates, team rosters, and user-specific dashboards, which are absent or less efficient on the existing site.
- **Accessibility:** We prioritize accessibility, ensuring that the site is usable for individuals with disabilities, with features like screen reader compatibility and improved color contrast.
- **Team Page View:** Unlike the old website, our website allows users to view their team information on a "my team" page, which allows for easy access to useful information.
- **Game Scores:** Unlike the old website, the user had to manually fill out a game form and send it to the commissioner. Our new implementation allows captains to fill out the scores quickly and submit in less than a couple seconds.

6 Unit Testing

6.1 Current Backend Unit Testing

Our current backend unit testing aligns closely with our VnV Plan for unit testing. We opted to start by unit testing the backend models in order to ensure that the foundational components of our platform are reliable and follow our requirements so that we can build more complex features with confidence.

Some tests were added since the VnVPlan was created and some tests were removed and are to be tested in integration testing.

Our backend unit testing is conducted using Jest and executed via npm test in the backend directory. The test files are located in the backend/test folder. These unit tests focus on verifying the fundamental functionality of our backend models, ensuring they behave as expected when interacting with MongoDB.

We began with backend testing because any issues caused by front-end components would be reflected in the backend models. By starting with the backend, we can ensure that the core functionality of our platform is reliable and that we can build more complex features with confidence.

Tests for Season, Reschedule Request, Announcements are WIP and some of their tests are documented below, but were not part of the VnVPlan.

6.1.1 Player Model

1. **test-player-creation**
   Path: backend/test/models/playerTests.test.js
   Description: Verify that the player model can be created with valid input and default fields are set correctly.
   Result: Pass

2. **test-player-uniqueness**
   Path: backend/test/models/playerTests.test.js
   Description: Verify that the player model enforces uniqueness for the email field.
   Result: Pass

6.1.2 Team Model

1. **test-team-creation**
   Path: backend/test/models/teamTests.test.js
   Description: Verify that the team model can be created with valid input, and default fields are set correctly.
   Result: Pass

2. **test-team-invalid-fields**
   Path: backend/test/models/teamTests.test.js
   Description: Verify that attempting to create a team with missing or invalid fields fails validation.
   Result: Pass

Note: Team uniqueness (VnVPlan test-team-uniqueness) will be verified as part of integration testing rather than unit testing.

6.1.3 Game Model

1. **test-game-creation**
   Path: backend/test/models/gameTests.test.js
   Description: Verify that a Game can be created with valid input and default fields are set correctly (scores defaults to 0).
   Result: Pass

6.1.4 Gameslot Model

1. **test-gameslot-creation**
   Path: backend/test/models/gameslotTests.test.js
   Description: Verify that a game slot can be created with valid input, ensuring all required fields are properly stored.
   Result: Pass

2. **test-gameslot-uniqueness**
   Path: backend/test/models/gameslotTests.test.js
   Description: Verify that a game slot enforces uniqueness based on date, time, and field to prevent duplicate entries.
   Result: Pass

6.1.5 Schedule Model

1. **test-schedule-creation**
   Path: backend/test/models/scheduleTests.test.js
   Description: Verify that a schedule can be created with the required fields, including `seasonId`, `gameSlots`, and `games`.
   Result: Pass

2. **test-schedule-timestamps**
   Path: backend/test/models/scheduleTests.test.js
   Description: Verify that a schedule model includes automatically generated timestamps (`createdAt` and `updatedAt`).
   Result: Pass

6.1.6 Reschedule Request Model

1. **test-reschedule-request-missing-required-fields**
   Path: backend/test/models/rescheduleRequestTests.test.js
   Description: Verify that missing required fields such as `game`, `requestingTeam`, `recipientTeam`, or `requestedGameslot` result in validation errors.
   Result: Pass

2. **test-reschedule-request-default-status**
   Path: backend/test/models/rescheduleRequestTests.test.js
   Description: Verify that the `status` field defaults to `Pending` when no status is provided.
   Result: Pass

3. **test-reschedule-request-valid-status**
   Path: backend/test/models/rescheduleRequestTests.test.js
   Description: Verify that the `status` field accepts valid values, including `Accepted`, `Pending`, and `Declined`.
   Result: Pass

4. **test-reschedule-request-invalid-status**
   Path: backend/test/models/rescheduleRequestTests.test.js
   Description: Verify that invalid values for the `status` field, such as `InvalidStatus`, throw a validation error.
   Result: Pass

5. **test-reschedule-request-timestamps**
   Path: backend/test/models/rescheduleRequestTests.test.js
   Description: Verify that the `createdAt` and `updatedAt` fields are automatically set by the Mongoose schema when a reschedule request is created.
   Result: Pass

6.1.7 Announcement Model

1. **test-announcement-missing-required-fields**
   Path: backend/test/models/announcementTests.test.js
   Description: Verify that missing required fields such as `title` or `content` result in validation errors.
   Result: Pass

2. **test-announcement-content-validation**
   Path: backend/test/models/announcementTests.test.js
   Description: Verify that the `content` field is required and cannot be empty.
   Result: Pass

3. **test-announcement-timestamps**
   Path: backend/test/models/announcementTests.test.js
   Description: Verify that the `createdAt` field is automatically set when an announcement is created.
   Result: Pass

6.1.8 Season Model

1. **test-season-unique-name**
   Path: backend/test/models/seasonTests.test.js
   Description: Verify that the `name` field is unique, ensuring no two seasons can have the same name.
   Result: Pass

2. **test-season-default-allowedDivisions**
   Path: backend/test/models/seasonTests.test.js
   Description: Verify that the `allowedDivisions` field defaults to `4` when not provided.
   Result: Pass

3. **test-season-missing-required-fields**
   Path: backend/test/models/seasonTests.test.js
   Description: Verify that missing required fields such as `name`, `startDate`, or `endDate` result in validation errors.
   Result: Pass

4. **test-season-status-enum**
   Path: backend/test/models/seasonTests.test.js
   Description: Verify that the `status` field accepts only valid values (`upcoming`, `ongoing`, `archived`) and rejects any invalid status.
   Result: Pass

5. **test-season-registeredTeams-unique**
   Path: backend/test/models/seasonTests.test.js

Description: Verify that the `registeredTeams` array enforces uniqueness for each team, ensuring no duplicate teams are registered for a season.
Result: Pass

6. **test-season-timestamps**
   Path: backend/test/models/seasonTests.test.js
   Description: Verify that `createdAt` and `updatedAt` timestamps are automatically set when a season is created.
   Result: Pass

Note: More in depth schedule testing is required and will be done in integration or manually testing.

6.2 Current Frontend Unit Testing

At present, unit testing for the frontend remains incomplete. It was not included in the VnVPlan but our team realized that having basic unit tests for the Frontend interactions is critical. This will be a priority for moving forward.

Our team is in the process of creating frontend unit tests for basic test displays, interactions, and components. We are using Jest, Babel, and React Testing Library to conduct these tests. The test files are located in the src/frontend/test directory.

6.2.1 Login Page

1. **test-login-form**
   Path: src/frontend/test/login.test.js
   Description: Verify that the login form renders correctly and accepts user input.
   Result: Not Tested Yet

2. **test-login-page-render**
   Path: src/frontend/test/login.test.js
   Description: Verify that the login page is rendering correctly by checking for text and input fields.
   Result: Not Tested Yet

7 Changes Due to Testing

During testing, some test cases were scrapped due to changes in the platform's implementation. Initially, we had the following test case for player team requests:

FR-6

- Description: Player requesting to join a team.
- How Test Will Be Performed: Tester will attempt to join a team while logged in as a Player without an existing team. Verify that the associated Captain receives the request (Refer to FR-7).
- Requirements Covered: SRS Req #6 (joining teams).
- Inputs: Player requests to join `TeamName`.
- Expected Output: Player's request is sent to the team's Captain.
- Actual Output: Not Tested Yet.
- Result: Not Tested Yet.

However, after further meetings with our supervisor, we revised the implementation to allow only team captains to invite players. As a result, the functionality to request joining a team was removed to prevent spamming of join requests, and this test case was no longer valid.

While performing other test cases, we identified improvements that prompted changes to the platform and test cases. For example, the following test case was modified:

FR-8

- Description: Captains submitting game results, which update the standings.
- How Test Will Be Performed: Captain submits game result and game scores (Win/Lose/Tie, ScoreTeamA, ScoreTeamB). Verify that the report is reflected in the database and that standings update accordingly.

- Requirements Covered: SRS Req #12 (Reporting).
- Inputs:
  - ScoreTeamA: Score for Team A.
  - ScoreTeamB: Score for Team B.
- Expected Output: Game report is updated, and standings reflect the new results.
- Actual Output: Game report successfully updates, and standings are correctly modified.
- Result: Pass

Upon review, we decided to simplify the process by removing the need for captains to manually submit the result (Win/Lose/Tie). Instead, we implemented backend logic to automatically calculate the game result based on the scores, enhancing the user experience.

8 Automated Testing

Our automated testing is conducted using Jest and React Testing Library for the frontend and Jest for the backend. We have not yet set up CI for our automated testing, but we plan to do so in the near future. Our unit tests and system tests are scripts to be run run locally using the command `npm test` in the respective directories. More details are found in the Unit Testing section above and the FR Evaluation section.

9 Trace to Requirements

Table 1: Traceability Matrix of Tests to Requirements

Test Case	Requirement ID	Description
Player Controller Tests		
Signup – should create a new player	FR-1	Verifies that a new player is created with valid data and returns a playerId, email, and token.
Login – should authenticate an existing player	FR-1	Validates that an existing player can log in with correct credentials and receives a token and playerId.
Login – should fail with invalid credentials	FR-1	Ensures that login fails with an incorrect password, returning a 403 error and an appropriate message.
Team Controller Tests		
Prevent multiple teams per captain	FR-5	Checks that a captain cannot create more than one team in the same season, returning a 403 error if attempted.
Prevent creating a team with a duplicate name (same captain)	FR-5	Verifies that attempting to create a team with an existing name results in a 400 error.
Prevent creating a team with a duplicate name (different captain)	FR-5	Ensures that duplicate team names are rejected even when a different captain is involved.

Continued on next page

Table – Continued

Test Case	Requirement ID	Description
Announcements Model Tests		
Create an announcement with required fields	FR-10	Confirms that an announcement is created with a title, content, and an automatically set createdAt timestamp.
Throw error if title is missing	FR-10	Ensures that the announcement model requires a title and throws a validation error when missing.
Throw error if content is missing	FR-10	Ensures that the announcement model requires content and throws a validation error when missing.
Set createdAt to current date by default	FR-10	Validates that the createdAt field is automatically set upon creation.
Allow announcements with non-empty content	FR-10	Checks that an announcement with non-empty content passes model validation.
GameSlot Model Tests		
Create a game slot with required fields	FR-7	Verifies that a game slot is created with a valid date, time, and field.
Allow game field to be optional	FR-7	Ensures that a game slot can be created without an associated game.
Enforce uniqueness of date, time, and field	FR-7	Confirms that duplicate game slots (same date, time, field) are not allowed.

Continued on next page

Table – Continued

Test Case	Requirement ID	Description
Game Model Tests		
Create a game with required fields	FR-7	Verifies that a game is created with all necessary fields (date, time, field, teams, etc.).
Default homeScore and awayScore to null	FR-7	Checks that when a game is created, its scores default to null if not provided.
Player Model Tests		
Create a player with required fields	FR-1	Ensures that a player is created with all required attributes (first name, last name, email, password).
Default gender to "other"	FR-1	Verifies that the gender field defaults to "other" if not specified.
Default waiverStatus to false	FR-1	Confirms that a new player's waiverStatus is set to false by default.
Throw error for missing required fields (e.g., password)	FR-1	Checks that omitting a required field (like password) results in a validation error.
Throw error for duplicate email	FR-1	Verifies that creating a player with an email already in use is rejected.
Reschedule Request Model Tests		
Create a reschedule request with required fields	FR-8	Validates that a reschedule request is created with all required fields and defaults its status to "Pending".

Continued on next page

Table – Continued

Test Case	Requirement ID	Description
Throw error if game is missing	FR-8	Ensures that a reschedule request without a game field throws a validation error.
Throw error if requestingTeam is missing	FR-8	Ensures that a missing requestingTeam field causes a validation error.
Throw error if recipientTeam is missing	FR-8	Confirms that omitting the recipientTeam field results in a validation error.
Throw error if requestedGameslot is missing	FR-8	Verifies that a missing requestedGameslot field triggers a validation error.
Default status to "Pending"	FR-8	Confirms that new reschedule requests default to a status of "Pending".
Accept valid status and reject invalid status	FR-8	Checks that only acceptable status values (e.g., "Accepted") are allowed and invalid ones are rejected.
Set createdAt to current date by default	FR-8	Validates that the createdAt timestamp is automatically set for reschedule requests.
Schedule Model Tests		
Create a schedule with required fields	FR-7	Ensures that a schedule is created with a seasonId, gameSlots, and games array.
Have timestamps createdAt and updatedAt	FR-7	Verifies that schedule documents automatically include createdAt and updatedAt timestamps.

Continued on next page

Table – Continued

Test Case	Requirement ID	Description
Season Model Tests		
Create a season with required fields	FR-7	Validates that a season is created with required fields (name, startDate, endDate, allowedDivisions, status).
Default allowedDivisions to 4	FR-7	Checks that the allowedDivisions field defaults to 4 when not explicitly set.
Default status to "upcoming"	FR-7	Verifies that the season status defaults to "upcoming" if not specified.
Throw error for invalid status	FR-7	Confirms that an invalid status value triggers a validation error.
Accept empty divisions array	FR-7	Ensures that a season can be created even when the divisions array is empty.
Team Model Tests		
Create a team with required fields	FR-5	Verifies that a team is created with required fields (name, divisionId, captainId, seasonId).
Default wins, losses, and draws to 0	FR-5	Confirms that a team's wins, losses, and draws default to 0.
Set preferredTimes to "Balanced" by default	FR-5	Checks that the preferredTimes field defaults to "Balanced" if not specified.
Throw error for invalid preferredTimes	FR-5	Ensures that invalid values for preferredTimes are rejected.
Accept empty blacklistDays array	FR-5	Verifies that an empty blacklistDays array is accepted.

Continued on next page

Table – Continued

Test Case	Requirement ID	Description
Throw error for invalid blacklistDays value	FR-5	Confirms that invalid values in blacklistDays trigger a validation error.

10 Trace to Modules

Traceability Matrix of Test Cases to Modules.

Module ID	Module Name	Related Test Cases	Verification Method
M1	User Interface (UI)	TC-01 TC-06 TC-07 TC-08	Manual Testing, Usability Testing
M2	Authentication	TC-02 TC-11	Unit Testing, System Testing
M3	Team Management	TC-03 TC-09 TC-12	Unit Testing, System Testing
M4	Game Management	TC-04 TC-10 TC-13	System Testing, Integration Testing
M5	Announcements	TC-05	Manual Testing, Functional Testing
M6	Standings	TC-06 TC-14	System Testing, Integration Testing
M7	Scheduling	TC-07 TC-12	System Testing, Performance Testing
M8	Waiver Module	TC-08 TC-09	Unit Testing, Functional Testing
M9	Player Module	TC-09 TC-10	Unit Testing, Integration Testing
M10	Notification Module	TC-10	Functional Testing, Manual Testing
M11	Backend API	TC-11 TC-12 TC-13	Unit Testing, Integration Testing
M12	Scheduling Algorithm	TC-12 TC-14	System Testing, Performance Testing
M13	Reschedule Request Module	TC-13 TC-15	Functional Testing, System Testing

Description of Modules and Their Validation

- **User Interface (M1):** This module ensures proper rendering and responsiveness of UI components. It is validated through manual usability testing and front-end unit tests.
- **Authentication (M2):** Handles user authentication, login/logout functionality, and security measures such as RBAC. Tested via unit tests and system authentication tests.
- **Team Management (M3):** Manages the creation, joining, and management of teams. Verified using unit tests and system tests.
- **Game Management (M4):** Allows game scheduling, score reporting, and result tracking. Validated through system tests and integration tests.
- **Announcements (M5):** Ensures that administrators can post and manage announcements across the platform. Verified through functional and manual testing.
- **Standings (M6):** Manages ranking calculations and updates based on reported game results. Tested via system and integration tests.
- **Scheduling (M7):** Handles automated game scheduling based on team preferences and available slots. Verified through system and performance testing.
- **Waiver Module (M8):** Manages player waivers, ensuring compliance with participation requirements. Validated via functional and unit testing.
- **Player Module (M9):** Stores and manages player information, including team assignments and personal details. Verified through unit and integration tests.
- **Notification Module (M10):** Sends system notifications regarding schedule updates, game changes, and announcements. Validated through manual and functional tests.

- **Backend API (M11):** Ensures smooth communication between the frontend and backend systems. Verified through unit and integration tests.
- **Scheduling Algorithm (M12):** Implements the logic for fair and balanced scheduling of games. Validated through system and performance testing.
- **Reschedule Request Module (M13):** Manages captain-initiated reschedule requests and commissioner approvals. Verified through functional and system testing.

11 Code Coverage Metrics

11.1 Method

The test coverage was generated using Jest, ensuring thorough validation of our codebase.

11.2 Analysis

The results indicate a good overall coverage, garnering great confidence in our foundational models. Some errors are causing the controllers to have some bugs when detecting coverage, and we are working on extending and fixing our system tests. We made up for the lower coverage by extensively manually testing the controllers and ensuring they work as expected.

11.3 Next Steps

As we continue to improve our controller tests, we will focus on writing more integration tests to ensure complete coverage. For now, we are confident in the overall stability of the system, thanks to the solid foundation provided by our model and acceptance tests.

References

Figure 3: Jest Code Coverage Metrics

File	% Stmts	% Branch	% Funcs	% Lines
All files	26.32	0.4	2.83	26.9
backend	84.09	12.5	40	84.09
server.js	84.09	12.5	40	84.09
...ntrollers	16.44	0	0	16.68
...llers.js	14.28	0	0	14.28
...llers.js	17.85	0	0	17.85
...oller.js	44	0	0	44
...oller.js	14.75	0	0	14.75
...llers.js	16.27	0	0	16.27
...oller.js	13.79	0	0	13.79
...oller.js	16.27	0	0	17.07
...llers.js	15.78	0	0	16.21
...oller.js	12.9	0	0	13.79
...llers.js	17.04	0	0	17.04
...nd/models	92.85	100	33.33	92.85
...ments.js	100	100	100	100
division.js	100	100	100	100
game.js	100	100	100	100
gameslot.js	100	100	100	100
...error.js	33.33	100	0	33.33
...ation.js	100	100	100	100
player.js	100	100	100	100
...quest.js	100	100	100	100
schedule.js	100	100	100	100
season.js	80	100	50	80
standing.js	100	100	100	100
team.js	100	100	100	100
...nd/routes	96.87	0	0	96.87
...outes.js	100	100	100	100
...outes.js	100	100	100	100
...route.js	100	100	100	100
...outes.js	100	100	100	100
...outes.js	100	100	100	100
...outes.js	100	100	100	100
...outes.js	100	100	100	100
...outes.js	83.33	0	0	83.33
...outes.js	100	100	100	100
...outes.js	100	100	100	100
...algorithm	5.94	0	0	6.38
helpers.js	5.94	0	0	6.38

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Reflection.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

   The process of gathering and documenting test cases went smoothly, particularly due to the VnV Plan that we created previously to look back onto and the effective communication with the team. The detailed structure for each test case was already there, so we just had to follow our original plan and that really helped streamline the writing process.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   One challenge we had was adapting to the requirements from our supervisor, which led to the adjustments in the test cases throughout development. We resolved this by maintaining flexibility in our code approach and updating our test plans to adjust for those implementation changes. Overall, there weren't too many issues that were a pain to deal with.

3. Which parts of this document stemmed from speaking to your client(s) or a proxy (e.g. your peers)? Which ones were not, and why?

   As mentioned before, the changes to test cases came directly from adapting to the requirements from our supervisor over the course of

many meetings. One good example is FR-6 in our original VnV Plan, which was the player's ability to request to join a team. Our supervisor thought it would be annoying for captains to see many team invites, as players could just constantly request to join all the teams. Therefore, he asked us to remove the functionality completely to only allow captains to invite players they wanted. Some other parts of the documents were changed purely by our own thoughts and opinions, such as removing the need for captains to input game results (Win/Lose/Tie) in FR-8. We decided that it was a good change to improve the user experience, and speed up the process for captains to submit scores.

4. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV? If there were differences, what changes required the modification in the plan? Why did these changes occur? Would you be able to anticipate these changes in future projects? If there weren't any differences, how was your team able to clearly predict a feasible amount of effort and the right tasks needed to build the evidence that demonstrates the required quality? (It is expected that most teams will have had to deviate from their original VnV Plan.)

   We have already discussed the modifications in the plan in the previous questions above, so we won't bother discussing it again in this section. Our VnV Plan originally had automated testing, but we had to deviate from that due to the superiority of manual testing for this website. It was extremely easy to test features, such as adding players to a team and verifying by looking at the database manually.

   We've set the basis up of automated testing using Jest and connected CI to it, but it is not entirely complete at this deliverable checkpoint. Further, the unit testing plan was slightly different from the actual activities conducted for VnV. We've removed a few unit tests that were decided to be better tested manually or in system tests, and added some unit tests that we deemed necessary for the foundation of our application.

   The rest of our VnV Plan did not have to change, so there weren't many differences. Our team was able to clearly predict the right tasks to build the evidence that demonstrated the required quality because we have tremendous experience with sports platforms, participating in sports

all our lives. Ever since coming to McMaster, we have regularly used McMaster IMLeagues to play intramurals, and we understand what makes a sports platform perform well from the users perspective. We also have experience with full-stack web development from our previous co-ops and internships, and we are familiar with building a big project from scratch. Testing was often regularly done before pushing our products to production.