# Feature Overview and User Manual

**Url: https://summer25-project-summer25-project-zarude.onrender.com**

## Rock Paper Scissors Lizard Spock

### 1.0 Introduction / Overview

A new game called "Rock Paper Scissors Lizard Spock" (RPSLS) is available to play.

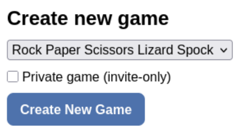


Figure 1: The Rock Paper Scissors Lizard Spock game in the dropdown menu.

### 1.1 Gameplay and Rules

RPSLS is played between two users. Each player selects a move, which can be one of rock, paper, scissors, lizard or spock. The win conditions are as follows: scissors beats paper, paper beats rock, rock beats lizard, lizard beats spock, spock beats scissors, scissors beats lizard, lizard beats paper, paper beats spock, spock beats rock, rock beats scissors. Any other combination of moves results in a tie. Both users must select a move before a round is processed.

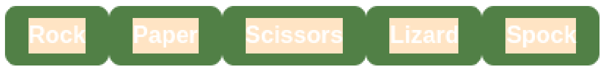


Figure 2: Buttons for selecting a move

### 1.2 Rounds

RPSLS can be set to go three, five or seven rounds. By default, it is set to three rounds. A "round" is defined as the two players each selecting a move which results in either a win or a tie. If one player "wins" the round, one point is awarded to them. If there is a tie, no players are awarded points. The overall winner is determined by who has the most points.

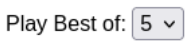
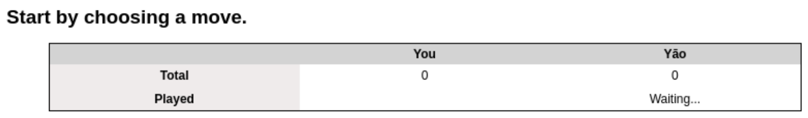


Figure 3: Dropdown menu for selecting the number of rounds



Start by choosing a move.

| | You | Yāo |
|---|---|---|
| Total | 0 | 0 |
| Played | | Waiting... |

Figure 4: The menu that keeps track of each players' points

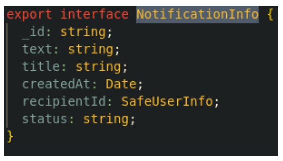
## Notifications

### 2.0 Introduction / Overview

Users will now receive notifications for game updates and direct messages.

### 2.1 Notifications data type

Notifications are a new data type, largely adapted from the pre-existing "thread" data type. Notifications in the database consist of the notification's id, the title of the notification, the actual notification message, how long ago it was created, the id of the recipient, and whether or not the notification has been read.



```
export interface NotificationInfo {
  _id: string;
  text: string;
  title: string;
  createdAt: Date;
  recipientId: SafeUserInfo;
  status: string;
}
```

Figure 5: The Notification data type

### 2.2 Reading Notifications

Notifications can be read in the notifications page, which contains all notifications whose "recipientId" matches the current user context. As shown in figure 6, the notifications page can be accessed on the

“Notifications” tab on the left-hand side bar, or the top right icon in the header, which is either a mail icon or a bell icon (see 2.4 for more on the bell icon).

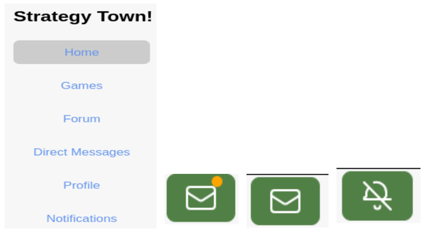
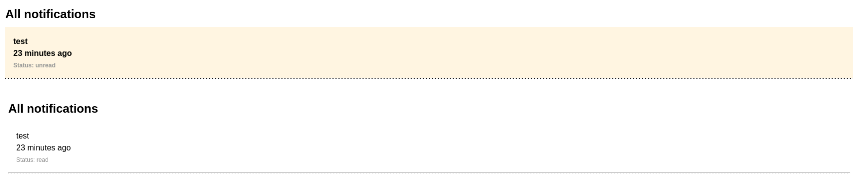


Figure 6: Ways to access the notifications page

### 2.3 “Read” and “Unread” Notifications

When notifications are created, they are set to be “unread” and a “ding” sound should play on the recipient’s side. When a notification is unread, it will be highlighted yellow in the notifications page, and unhighlighted when it is read (Figure 7). When a user has at least one unread notification, the mail icon will have a small orange dot next to it. A notification becomes “read” when a user clicks on it in the notifications page (Figure 8).



All notifications

test
23 minutes ago
Status: unread

All notifications

test
23 minutes ago
Status: read

Figure 7: “Read” vs “Unread” notification

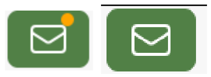
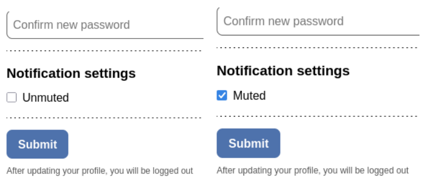


Figure 8: Mail icon with unread notifications vs mail icon with no unread notifications

### 2.4 “Muted” attribute

Users have the attribute of being “muted” or “unmuted”, and are “unmuted” by default. Users who are unmuted will have a mail icon that may or may not have a small orange dot next to it, depending on whether or not there are unread messages, while users who are muted will have a bell icon with a slash through it at all times. Furthermore, users who are muted will not hear a “ding” sound when a new notification comes in. Users have the option to change their “muted” or “unmuted” attribute via a checkbox on the profile page. Muted users will still receive notifications, and they will be highlighted and marked as unread.



Confirm new password

Notification settings

☐ Unmuted

Submit

After updating your profile, you will be logged out

Confirm new password

Notification settings

☑ Muted

Submit

After updating your profile, you will be logged out

Figure 9: “Unmuted” vs “Muted” state on profile page

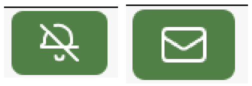


Figure 10: Bell vs mail icon

### 2.5 How Notifications are Created

There are two instances where notifications are created: a game that a user was watching ends, a new private chat message is created (see “Private Games / Chat). `user0` will also have a “test” notification by default for testing purposes. A user “watching” a game is defined as any user who joined the lobby before it reached the “done” state.

## Direct Messages/Private Games

### 3.0 Introduction / Overview

Users can create private games, and have the ability to create and receive private message, which go directly to a user’s inbox

### 3.1 Private Games

Private games are games that can only be accessed via a code. Users have the option to create a private game when making a new game. When a private game is created, an access code is automatically generated. When another user attempts to enter the private game, they must enter the access code correctly.

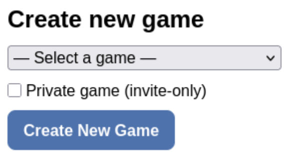
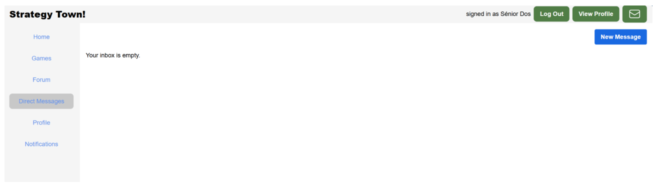


Figure 11: The private game check mark

### 3.2 Direct Message Page

Newly added to the sidebar navigation is a direct message tab which features the users inbox in addition to a button which allows them to send messages to other specified users.

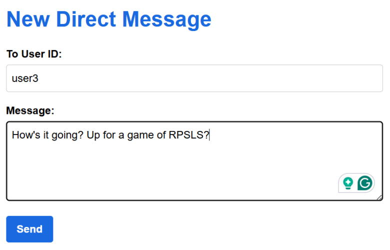


Your inbox is empty.

Figure 12: The inbox page

### 3.3 Sending Messages

Upon clicking the New Message button, users will be directed to a new page with a new message UI tool. They are prompted for a username and a message body.



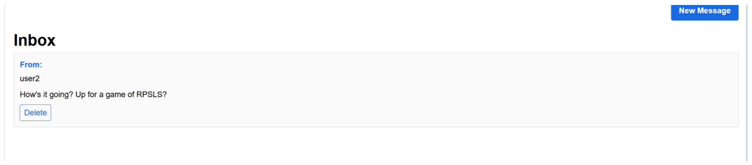
To User ID: user3

Message: How's it going? Up for a game of RPSLS?

Figure 13: An example prompt to send a direct message

### 3.4 Managing Messages

The message appears in the targeted user's inbox, allowing them to view it with their other messages and delete it if they wish to reduce clutter



From: user2

How's it going? Up for a game of RPSLS?

Figure 14: The inbox page with a message

## Technical Overview

1. **Summary of Major Changes**
    a. Our project introduced three major new features to the existing system: Rock Paper Scissors Lizard Spock, Notifications, and Private Games/ Direct Messages. Each of these required significant updates across the frontend and backend. The goal was to add fun and connectivity while maintaining a clean architecture.
2. **Key Code and Architectural Changes**
    a. **RPSLS**
        i. Game Logic and Validation: The game logic is handled centrally in a game logic module, which determines the outcome of a round based on both players' selections. The win/loss rules cover all 10 possible interactions between the 5 choices. A new validator using Zod was also added to ensure that only valid moves are processed by the backend.
        ii. Game Configuration: On the client side, a new dropdown menu was introduced that lets users select the total number of rounds (3, 5, or 7), with 3 rounds as the default. Each round is completed once both players have made a valid selection. The backend then calculates the winner of the round and updates the game state accordingly. Points are tracked per player and displayed in the game UI. A win is awarded a point; a tie results in no points for either side.
        iii. Frontend Integration: Several UI updates were made to accommodate the RPSLS feature:
            - A new dropdown in the game setup screen allows users to choose between Rock Paper Scissors and RPSLS
            - The in-game UI includes five buttons, one for each move, allowing users to make their selection.
            - A scoreboard was added to display each player's current point total. The system dynamically tracks when a player has won the majority of rounds and ends the game accordingly.
    b. **Notifications**
        i. A new "notifications" data type was added, heavily adapted from the "threads" data type. The details of this type are contained in **notification.types.ts**, as well as a zod validator that ensures all created notifications are well-formed
        ii. On the server side, a new notifications model was created in **notification.model.ts** with the proper schema, fields, and DB level validation of notifications. A notifications service was made to apply the business logic of notifications, including the DB creation of notifications and the DB retrieval of notifications, both individual and multiple notifications in **notification.service.ts**. A controller **notification.controller.ts** was also added to parse the inputs of and return responses for HTTP requests, in the new routes defined in **app.ts**.
        iii. On the client side, there was another service file for notifications called **notificationService.ts** to create notification-related REST API calls, including GET requests to obtain notification info, marking notifications as read, and listing notifications. Furthermore, there were various UI related changes, including a new Notifications Page in **NotificationPage.tsx**, a UI for individual notifications in **Notifications.tsx**, and avenues to access the notifications page, including a "Notifications" field in Sidebar.tsx and a new dynamic notifications button added to **Header.tsx** in **NotificationButton.tsx. useNotificationInfo.ts** and

**useNotificationList.ts** were useEffect hooks added to pull the information of an individual notification and a list of notifications respectively.

c. **Private games / chat**
To enable private gameplay, we modified the game creation flow to include a “Private Game” checkbox that auto-generates an access code. Only users with the code can join. We also added direct messaging:
   - A new inbox and messaging page lets users send and receive private messages.
   - Messages are stored and displayed using a new UI component. Users can also delete messages from their inbox to stay organized.

3. **Testing**

We made sure to follow the principles of test-driven development and to test early and often. Each backend service was tested using unit tests. On the frontend, we did UI walkthroughs and used dummy users to simulate gameplay, notifications, and private messaging. Finally, we made sure to rest using edge cases For example, we tested for ties in the RPSLS game, muted user states, missing IDs, etc.

## Process Overview

During the process of this project, our group was able to take advantage of the agile management model in order to organize our team work. We were able to utilize sprints, stand up, sprint reviews, and retrospectives to hold each other accountable and ensure that we stayed on track. When originally planning for our project, we broke down each user story and assigned tasks to each sprint. At the end of each sprint, we held sprint reviews to assess what was completed and retrospectives to not only reflect on our workflow, but adjust and plan for the upcoming sprint. When reviewing, we also made sure to use blameless reviews by focusing on problem solving instead of assigning blame when faced with challenges.

Looking back on our original plan for each sprint, we were able to mostly stay on track with our initial planning with some slight deviations. Our most noticeable change was that in our proposed sprint plan, we had broken tasks down by conditions of satisfaction. However, in practice, we first focused on the back end for all conditions before moving on to the front end. Starting with sprint one, we focused primarily on foundational tasks. We initially planned to have the research, planning, and begin implementing types for all three user stories done. However, due to midterms we were only able to implement the types for one of the user stories. To adapt to this, we decided to move the initial implementations of the remaining two stories to sprint two. The second sprint is where we began seeing a divergence from our initial planning. Originally, we had planned to be building the basic UI for our essential conditions during this sprint. Instead, we built the majority of our back end logic. This brought us into sprint three. Initially, we had planned to work through our desirable conditions during the sprint. This was replaced with the team working on developing the UI for the user stories. Moving into our final sprint, this week was intended to work on extension conditions as well as testing and documentation. However, due to time constraints, we were not able to implement the extension conditions of satisfaction. We instead pivoted to focus on debugging, testing, and finalizing our code. Although it may seem that a lot of changes were made along the way, we were still able to accomplish most of what we had set out to complete. The changes we made were instead an alteration of how we approached the project, not what we completed.

If we could offer advice to future teams, our biggest suggestion would be to have more consistent and scheduled stand up meetings. During our completion of the project, we mostly communicated virtually for stand ups, and were not consistent with our scheduling. As a result, we often had problems communicating what work each member of the group had completed thus far. By setting a specific time and even meeting in person, it can help the group better communicate the state of their working and give members a better overall understanding of the status of the project.