# RallyRank Project Plan

Table of Contents

[RallyRank Project Plan 1](#_Toc177980096)

[Project Overview 1](#_Toc177980097)

[Goals and Objectives 1](#_Toc177980098)

[Technology Stack 1](#_Toc177980099)

[Project Breakdown 2](#_Toc177980100)

[Step 1: Set Up the Development Environment 2](#_Toc177980101)

[Step 2: Build the Backend API with Flask 2](#_Toc177980102)

[Step 3: Build the Frontend with React 2](#_Toc177980103)

[Step 4: Connect Frontend and Backend Locally 3](#_Toc177980104)

[Step 5: Deploy the Backend 3](#_Toc177980105)

[Step 6: Deploy the Frontend 3](#_Toc177980106)

[Step 7: Testing and Iteration 3](#_Toc177980107)

[Elo Rating System Details 4](#_Toc177980108)

[Learning Resources 4](#_Toc177980109)

[Notes and Ideas 4](#_Toc177980110)

## Project Overview

* **Project Name**: RallyRank
* **Description**: A web application that allows coworkers to track ping pong game results and view player rankings using an Elo rating system.
* **Purpose**: To enhance engagement and friendly competition among coworkers while learning programming skills.

## Goals and Objectives

* **Primary Goals**:
  + Develop a functional web app for tracking ping pong rankings.
  + Learn and apply programming skills in Python (Flask) and JavaScript (React).
* **Secondary Goals**:
  + Host the app for free so coworkers can access it on their devices.
  + Implement an Elo rating system tailored to the office environment.

## Technology Stack

* **Frontend**: React
* **Backend**: Flask (Python)
* **Database**: SQLite (or alternative)
* **Hosting Services**:
  + **Frontend**: GitHub Pages, Netlify, or Vercel
  + **Backend**: PythonAnywhere, Render.com, or Deta.sh

## Project Breakdown

### Step 1: Set Up the Development Environment [DONE]

**Tasks:**

* Install Python and set up a virtual environment.
* Install Flask.
* Install Node.js and create a React app using `create-react-app`.

**Progress**:

* Python, Flask, and Node.js installed
* Blank React app started

### Step 2: Build the Backend API with Flask [DONE]

**Tasks**:

* Set up Flask project structure.
* Create API endpoints:
  + /players (GET, POST, DELETE)
  + /games (POST)
  + /rankings (GET)
* Implement the Elo rating logic.
* Set up the database (SQLite for simplicity).

**Progress**:

* Flask project structure set up
* Placeholder API endpoints done
* Elo rating logic done
* Database done

### Step 3: Build the Frontend with React [DONE]

**Tasks**:

* Set up React project structure.
* Create components/pages:
  + Rankings display
  + Add/Remove players form
  + Submit game results form
* Implement state management (useState, useEffect).
* Integrate API calls to the Flask backend.

**Progress**:

* Done

### Step 4: Connect Frontend and Backend Locally [DONE]

**Tasks**:

* Configure CORS in Flask to allow requests from React app.
* Test all API endpoints from the React frontend.
* Debug any issues with data fetching or state updates.

**Progress**:

* [Updates here]

### Step 5: Deploy the Backend [DONE]

**Tasks**:

* Choose a free hosting service (e.g., PythonAnywhere).
* Set up an account and configure the environment.
* Deploy the Flask app.
* Test API endpoints in the live environment.

**Progress**:

* [Updates here]

### Step 6: Deploy the Frontend [DONE]

**Tasks**:

* Choose a hosting service (e.g., GitHub Pages, Netlify).
* Build the React app for production.
* Deploy the app and ensure it's accessible publicly.
* Update API endpoint URLs to point to the live backend.

**Progress**:

* [Updates here]

### Step 7: Testing and Iteration

**Tasks**:

* Remove duplicate headers for Current Rankings/Player Rankings and Recent Game Results/Game History
* In Game History, reformat results like:
  + Matthew (score) defeated Sam (score), (date, time)
* Sort Game History with most recent games at top
* Limit game history on home page to 10 games
* Add page for full game history
* Make player names clickable links to player profile pages
* Player profile page should include:
  + Player Name
  + Rating
  + Match History
* Change UI to a “Players” page, which shows a table of players, their ratings, and their records (wins and losses). Have buttons on that page to add/edit/delete players
* Add a full game history page (with pagination)
* Make player rankings on home page formatted like this:
  + 1. Sam (rating)
  + 2. Matthew (rating)
* Migrate SQL database to PostgreSQL using Tembo
* Add theme, make everything look nice and mobile-optimized

**Progress**:

* [Updates here]

## Elo Rating System Details

**Description**:

* Outline how the Elo rating system will be implemented.
* Define initial ratings, K-factor, and how ratings update after each game.
* Include any custom rules or adjustments specific to your office environment.

**Implementation Plan**:

* [Plan here]

## Learning Resources

* **Python and Flask**:
  + Official Flask documentation
  + [Flask Mega-Tutorial by Miguel Grinberg](https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world)
* **JavaScript and React**:
  + [Official React documentation](https://react.dev/learn)
  + [freeCodeCamp React Course](https://www.freecodecamp.org/)
* **Flask + React**:
  + [How to Create a React + Flask Project by Miguel Grinberg](https://blog.miguelgrinberg.com/post/how-to-create-a-react--flask-project)

## Notes and Ideas

* Feature Ideas:
  + Add match history for players
  + Include player profiles with stats
  + Implement doubles matches support
  + Program dot matrix display to show rankings near table
* Design Considerations:
  + Keep the UI simple and user-friendly
  + Use office-themed styling for engagement
* Potential Challenges:
  + Managing concurrent data updates