Design Specification: Battle Checkers

------------------------------------------------------------------------------------

High Level Description:

Battle Checkers is an online web-based Checkers game where two players can play against each other in checkers.

The game runs through a webapp and users can login and then start a game.

After logging in, players are taken to a dashboard, where they can view their profile, view leaderboards, or play the game.

Players win by taking all their opponent's pieces.

The webapp is hosted on an AWS EC2 server and uses a MySQL database and Node.js, React Javascript for the frontend.

Approaches Considered-----------------------------------------------------

We had always held JavaScript as our primary language, as it’s the one we have the most familiarity. MySQL and Node.js were natural extensions. After talking with the professor, we are considering AWS or AWS LAMBDAS for server housing/ processing and Express for the node.js framework.

Mockups------------------------------------------------------------------------

Home page

Graphical user interface, text, application

Description automatically generated

* [/api/users/login]

Registration Page

Text

Description automatically generated with medium confidence

* [/api/users/registration]

User Dashboard

A picture containing shape

Description automatically generated

* [/api/game/startgame]
* [/api/user/userstat]
* [/api/game/tournamentmain]
* [/api/user/topwins]
* [/api/users/logout]

User Profile

A picture containing table

Description automatically generated

* [/api/game/startgame]
* [/api/user/userstat]
* [/api/users/friends]
* [/api/game/tournamentmain]
* [/api/user/topwins]
* [/api/users/logout]

Game Board

Diagram

Description automatically generated

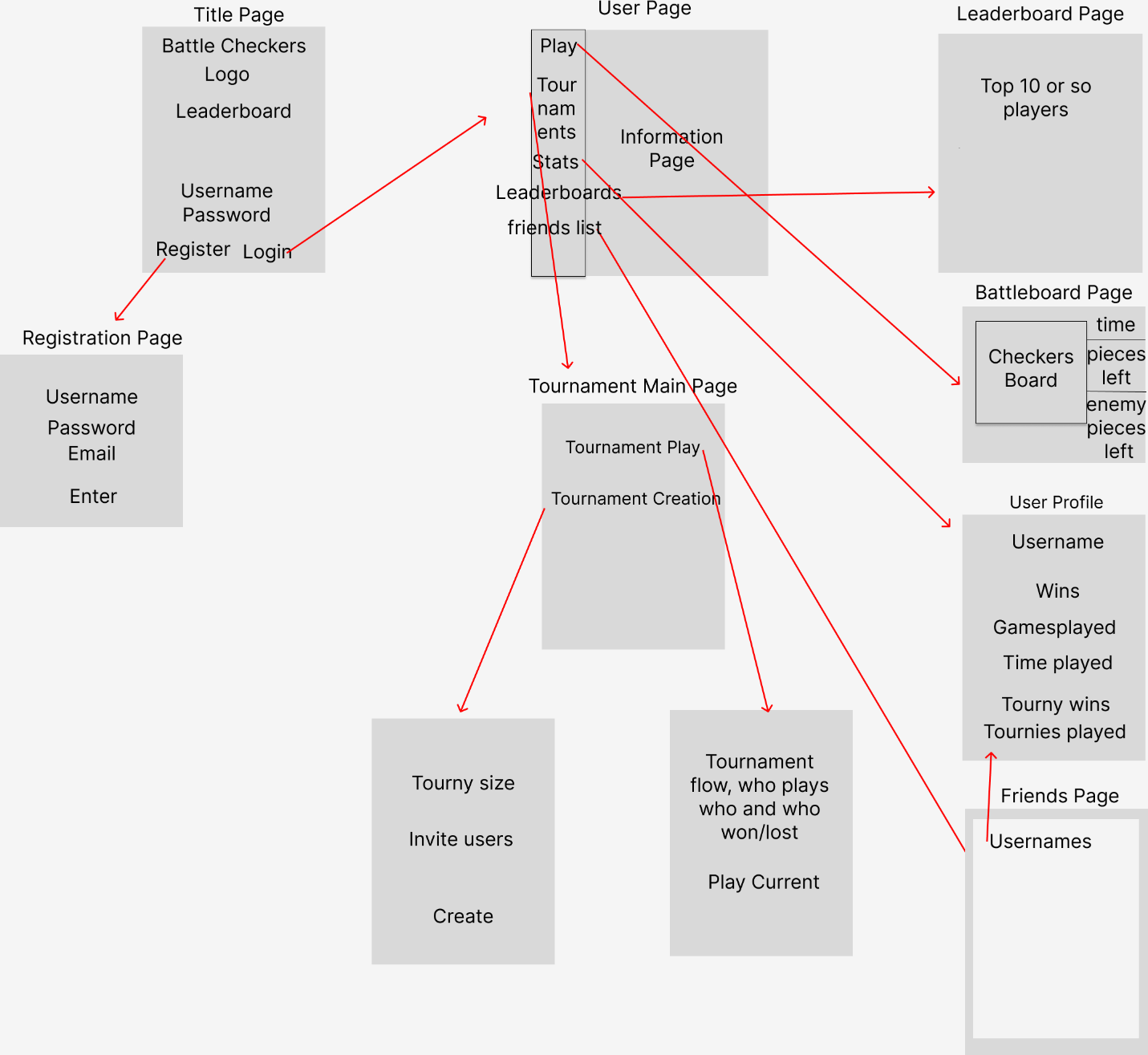
* [/api/game/startgame]
* [/api/user/userstat]
* [/api/game/tournamentmain]
* [/api/users/logout]

Tournament

Table

Description automatically generated with medium confidence

* [/api/game/startgame]
* [/api/user/userstat]
* [/api/game/tournamentmain]
* [/api/game/tournamentcreate]
* [/api/game/tournamentplay]
* [/api/users/logout]

Navigation Between Screens-----------------------------------------------

Backend Information---------------------------------------------------------

Graphical user interface

Description automatically generated with medium confidence

RESTful API---------------------------------------------------------------------

* A list of all RESTful endpoints that will be implemented. For each endpoint, list the endpoint’s functionality and it’s input and output payloads.
* [/api]
  + Returns browseable api documentation
* [/api/auth/login]
  + This is the endpoint that allows an admin to gain access to all restful endpoints, and manually hit them.
  + Payload:

{

username: user

password1: password1

}

* [/api/users]
  + Returns all the current registered users with their username and email address.
  + New users can also be added using POST.
* [/api/users/login]
  + Matches user credentials to the database and returns code if the user does not exist.
  + If the user does exist returns sessiontoken.
* [/api/users/topwins]
  + Update and fetch number of wins a user has in order to be used in a leaderboard.
* [/api/users/registration]
  + This is the endpoint to register a new user.
  + Payload:

{

username: user,

email: email,

password1: password1,

password2: password2

}

* [/api/users/logout]
  + Expires User's Sessiontoken after user request.
* [/api/game]
  + Returns all the current games that are active.
* [/api/game/startgame]
  + Creates an instance of a new game and matches a user with another.
* [/api/game/tournamentmain]
  + Returns the list of current tournaments a user is in
* [/api/game/tournamentcreate]
  + Creates a new tournament with parameters passed by the user
* [/api/game/tournamentplay]
  + Creates an instance of the game with the parameters passed by the tournament

Functionality Provided By Each Screen----------------------------------

\* Explain which RESTful endpoints are accessed by the screen to provide the specified functionality.

\* The Game Screen provides the main point of engagement with the user.

User Authentication / Data Security Considerations-----------------

\* All passwords will be hashed; plaintext password information will never be saved or manipulated.

\* UUIDs will be used instead of sequential user numbers.

\* Auth will hash passwords with a combination of metadata about the password and the password itself and store it in a database – Auth will not store raw passwords

\* Uses Password-Based Key Derivation Function 2 (PBKDF2) as the default algorithm of storing passwords

\* -algorithm-$-iterations-$-salt-$-hash-

\* Requires a massive amount of computing time to break

\* Auth controls the majority of authentication and authorization for the system

\* One can store and retrieve arbitrary data on a per-site-visitor basis

\* Data is stored server side and abstracts the sending or receiving of cookies

\* Cookies only contain the ID of the session; they do not contain any real data

Tech Stack----------------------------------------------------------------------

--Platform

\* WebApp, Platform independent, using Node.js LTS 12.16.1

\* The game itself runs inside the WebApp, using the JavaScript-React-WebGL library to integrate with the rest of the stack.

--Development Tools

Our team uses a variety of tools for local development. We ensure consistency in toolset only where required. Required areas of tool consistency include:

-Version Control: We use git, hosted on GitHub

-Areas where developers use the tools, they're most comfortable with:

-IDE/Text Editors (MySQL, VSCode)

--Backend

Programming Languages:

The Project uses a MySQL database to manage users and saved game state. We will also try to use Express Node.js for the framework of our backend. It’s something we don’t have experience with, but something that we can learn.

The Project's base infrastructure is primarily JavaScript, we chose this over Python strictly due our comfortability with JS. We have used it more often than Python.

EC2 / AWS:

We chose to use AWS to host our project because it was the online server host we had the most experience with via previous classes. We are considering AWS LAMBDA if we can figure it out in time.

Goals, and Who They've Been Assigned to-----------------------------

Jeremy Mongeau ([mongea74@students.rowan.edu](mailto:mongea74@students.rowan.edu)): Front-end, battle board programmer. He has experience from previous classes.

Avery Romano ([romano23@students.rowan.edu](mailto:romano23@students.rowan.edu)): Focus on tournament design and login/register. He has experience with backend SQL database work.

Kasso Peaks ([peaksk85@students.rowan.edu](mailto:peaksk85@students.rowan.edu)): Front-end Designer. Has experience with front-end programming through his job.

Enrico Gabri Duller ([duller98@students.rowan.edu](mailto:duller98@students.rowan.edu)): Database and backend. He has a little experience with it but wants to learn.

Andre Robinson ([robins85@students.rowan.edu](mailto:robins85@students.rowan.edu)): Front-end programmer, Tournament pages. He has a little experience with it but wants to learn.

Richard Erskine ([erskiner7@students.rowan.edu](mailto:erskiner7@students.rowan.edu)): Focus on Matchmaking concept, design and project document creator. While I have very little experience with JS or SQL, I’m eager to learn.