**WEC 2021**

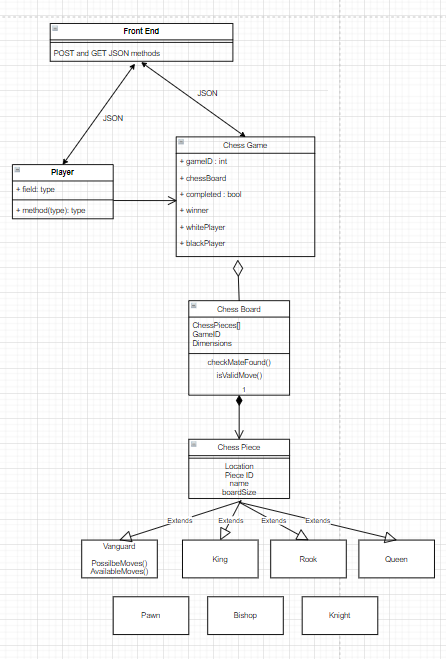
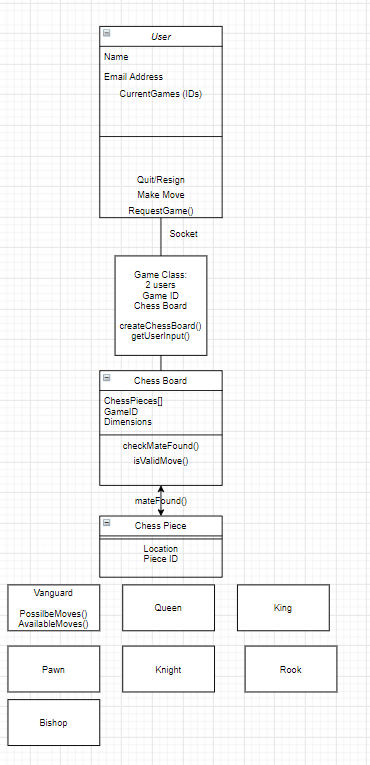
**Programming Division:**

Design Document

*Team: Flin Flon*

*Members: Andrew Gillis, Jared Gillis, Spencer Comin, Nathan Darby*

**Design process**

We followed the standard design process of starting by creating a UML Class diagram, and broad layout for our UI. We began with a UML diagram as it is a very standard way of thinking about and organizing a software project. It allows for a discussion on the broad strokes of the design, from what classes will be used to attributes of those classes. This allows for us to also be able to effectively divide labour among teammates, especially in the early stages of the project when it may not be clear what classes will need which information. Having a clear and concise UML diagram to guide the design allows us to actually split off in the beginning to be able to effectively work in parallel. To the **left** you can see our original rough draft of the UML diagram composed at the beginning of the competition while reading the prompt. **Below** you can see our more fully fledged, final draft of the UML diagram

**Build process**

Division of labour and clear communication on the interfacing between various parts of a software project are key to the success and timeliness of a project. Our goal has always been to be able to have members of the group be able to split off, work on separate parts, and only need to be heavily involved in writing code together when combining two people’s work together, or are stuck on an individual section.

A huge part of our process on this project was the need for a clear definition of an API (application programming interface) that enables the front and back end to be developed in parallel instead of in series. Thus our first big step after the design had been laid out was to develop the API to ensure that communication between the front end or our application and the back end would be standardized.

We split off into two sub teams, one to work on the frontend web application, and one to develop the backend and algorithms guiding the pieces movement and boardstate. This process allows for maximum parallelization and also ensuring that teammates have someone to speak to if they are encountering issues.

Final product:

As it currently stands we have a fully working database with connections to our backend which hosts the game board (and the gamestate, ie. piece placement associated), lists of games for each player, and can be accessed as is requested by the front end.

Front end:

Fully implemented authentication system which can be associated with google accounts. Then a working directory of pages, among which is functionality for requesting to create game with settings as required, request to join a game as invited by a friend, and to display the board and allow the player to make the moves that they want to.

Back end:

Fully implemented game board and pieces, with moves that can be made, and rejected based on whether they are valid moves or not (ie. blocked by other pieces).

Database:

Collections for users, boards and games

Connection from the front end to the back end:

This was surprisingly one of our biggest headaches. A lot of the issues arose from the presumption that JSON connection.

JSON communication from front to back end

* Fixed - Added Stringify and JSON items into POST requests in front end

Checkmate functionality

* In development.

**Design choices and rational :**

**Python:**

Python was a clear choice for development for our team. It is an especially well suited programming language for problems of this nature because of the ease and speed with which applications can be developed. It has downsides, such as slower run times, unnecessary system calls and other related issues. Many of these downsides only become issues when you are dealing with either large amounts of data or complicated OS level problems. Neither of these factors are present in this problem, so Python is perfectly suited for our solution.

**React:**

React is a very sleek Javascript library that is growing in popularity because of its usability. We wanted our design to have a simple and elegant front end, an area in which we wanted to improve from our universities competition. Through the last month we have been learning how to use the React library as we were confident that it was versatile enough to be usable for the problem that we were given at WEC.

**Material UI:**

Material UI is a library of React components. We had practice implementing these components, and knew that they provided a good framework within which we could develop a nice looking web app.

Auth0:

**Flask:**

Flask is a versatile framework that connects well with the react framework for the front end. It is an easy framework within which to employ Python as well.

**MogoDB:**

MongoDB is the most popular noSQL database, which allows for greater flexibility in development and allows for greater productivity as the application evolves. MongoDB has a free tier for their cloud based MongoDB Atlas service, this means MongoDB sets up the VM on an AWS server and takes care of all necessary “database back end” services such as load balancing, background update operations to ensure atomic operation (read/write to the database), and managing user connections giving us more time to work on other areas of the application. With this service we were able to test and debug our applications remotely from our own IDEs while ensuring we all had the exact same data to test our application. Scalability is also very easy with Atlas, as many tiers for different amounts of storage, connections, and retrieval speeds are available.

Thoughts process on data structures and algorithms:

Nothing too fancy was actually required for this project. The main challenge was constructing a system for finding valid moves for each piece. This was accomplished by mapping the potential path for a piece, and seeing if there were other pieces in the way along that path, so it could be rejected as a possible move. The algorithms for checkmate were more complicated.