This document will detail the high-level design philosophy behind our app for drafting Clarkson University Women’s Hockey League teams. First, the program’s different working parts (or “modules”) will be discussed, such as how the backend could look, how the frontend will provide a user interface, and how it will access necessary APIs on the backend. Then, the API will be discussed in detail. Third, the user interface itself will be outlined, such as the differences between casual user view and team-builder view. The external API(s) necessary for the function of the program will also be discussed, as well as how the data will be aggregated after being accessed. Lastly, the database schema will be gone into detail, as well. For example: what data structures will be used to store player data after import, as well as how teams will be stored internally (whether or not a player be a class object, etc).

The program will have two main modules, a frontend for interacting with data and a backend for storage of data. The frontend will be a progressive web app (PWA) designed primarily to run on an iPad. The frontend will be responsible for presenting the data to the user with a friendly interface, and will contain multiple views for the users to interact with, inspired by the current spreadsheet system in use. The frontend will also have a service worker script, which will cache certain data at the request of a user, and serve as a go-between for all communications with the server. When the service worker detects that it can no longer communicate with the backend server (for example, in arenas with poor WiFi) it will serve the user the cached version of the data, and cache changes for when communication is restored. The frontend will use a framework to improve development speed and to make a modern website. As of now, we are planning to use Vue.js as it offers a simple and flexible framework with a gentle learning curve, making development faster and more maintainable. Vue’s strong PWA support and optimized performance also make it a great fit for this project. The backend will consist of a Django application and a MySQL database. The Django application will serve pages to the frontend, and provide an API for creating, updating, reading, and editing data in a structured manner.

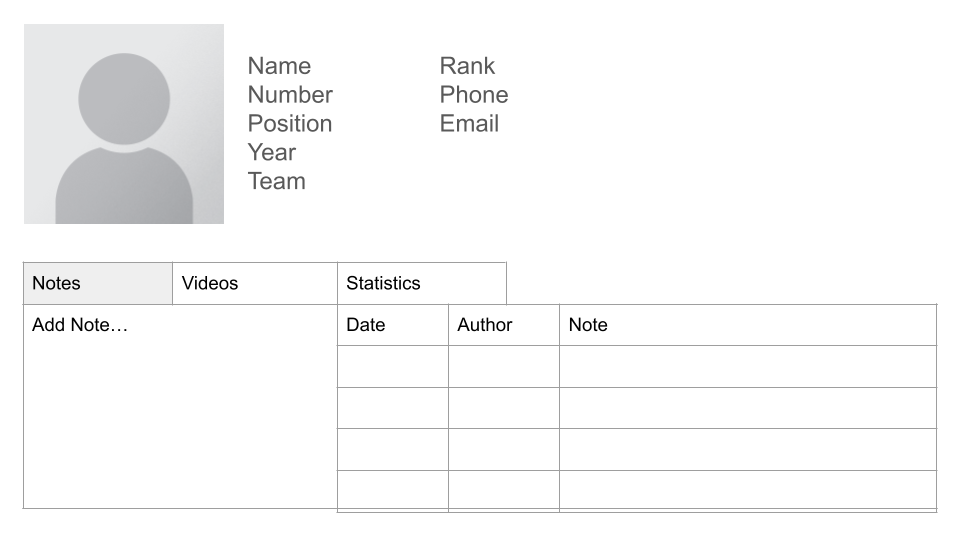


Figure 1: Player focus view for editing and viewing a specific player. A tabbed view on the bottom half of the page allows switching between editing/viewing notes, videos, and statistics

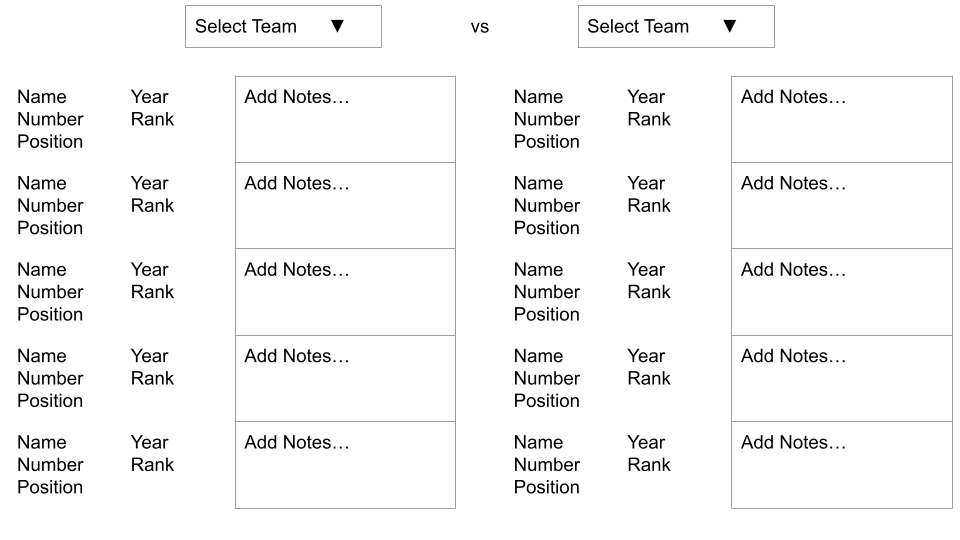


Figure 2: Game view showing two teams side by side for easy access to players

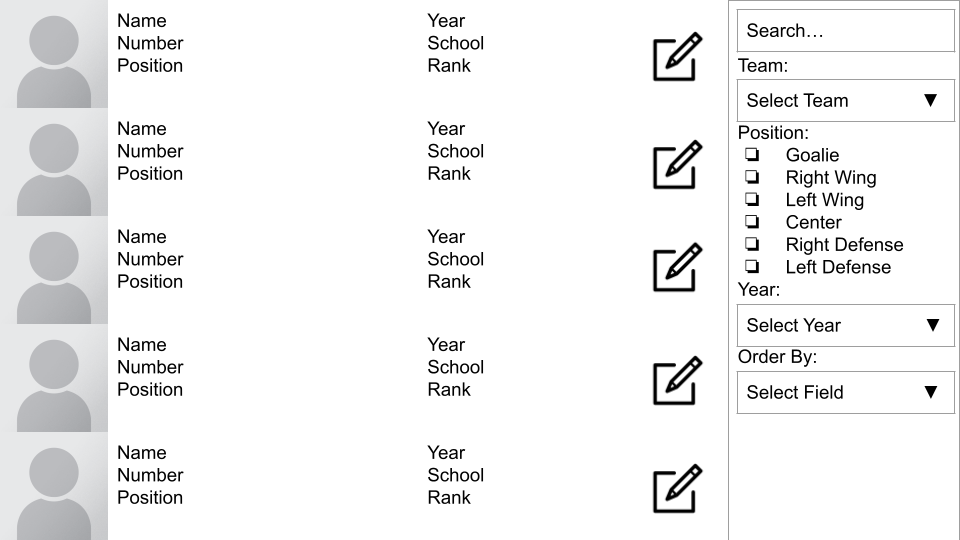


Figure 3: “Big list” view allows searching for players based on any number of parameters

Django will serve an RESTful API to the frontend so that it will have a structured way to interface with the database. The body of all API communication will be encapsulated in JSON, and the frontend will interact with the backend via GET and POST requests. While offline, the service worker should provide a nearly identical interface as the server, so only one schema will be discussed. Possible API paths and functionality are described as follows:

* **GET** /get/status
  + Returns a JSON object containing status information about the server, such as online/offline state.
* **GET** /get/player?id=
  + Returns a JSON object representing a player.
* **GET** /get/team?id=
  + Returns a JSON object representing a team containing a list of players.
* **GET** /search/player?query=
  + Returns a JSON list of all players matching a query string.
* **GET** /search/team?query=
  + Returns a JSON list of all teams matching a query string.
* **GET** /search/note?query=
  + Returns a JSON list of all notes matching a query string.
* **POST** /update/player
  + Post a JSON list containing player objects including only the ID and any modified fields.
  + With the exception of the ID, this method should not post fields that have not been modified.
* **POST** /update/team
  + Post a JSON list containing team objects including only the ID and any modified fields.
  + With the exception of the ID, this method should not post fields that have not been modified.
* **POST** /update/note
  + Post a JSON list containing note objects including only the ID and any modified fields.
  + With the exception of the ID, this method should not post fields that have not been modified.
* **POST** /create/player
  + Post a JSON list containing player objects including all fields except the ID.
  + This method should not post an ID, as this will be determined by the database.
* **POST** /create/team
  + Post a JSON list containing team objects including all fields except the ID.
  + This method should not post an ID, as this will be determined by the database.
* **POST** /create/note
  + Post a JSON list containing note objects including all fields except the ID.
  + This method should not post an ID, as this will be determined by the database.

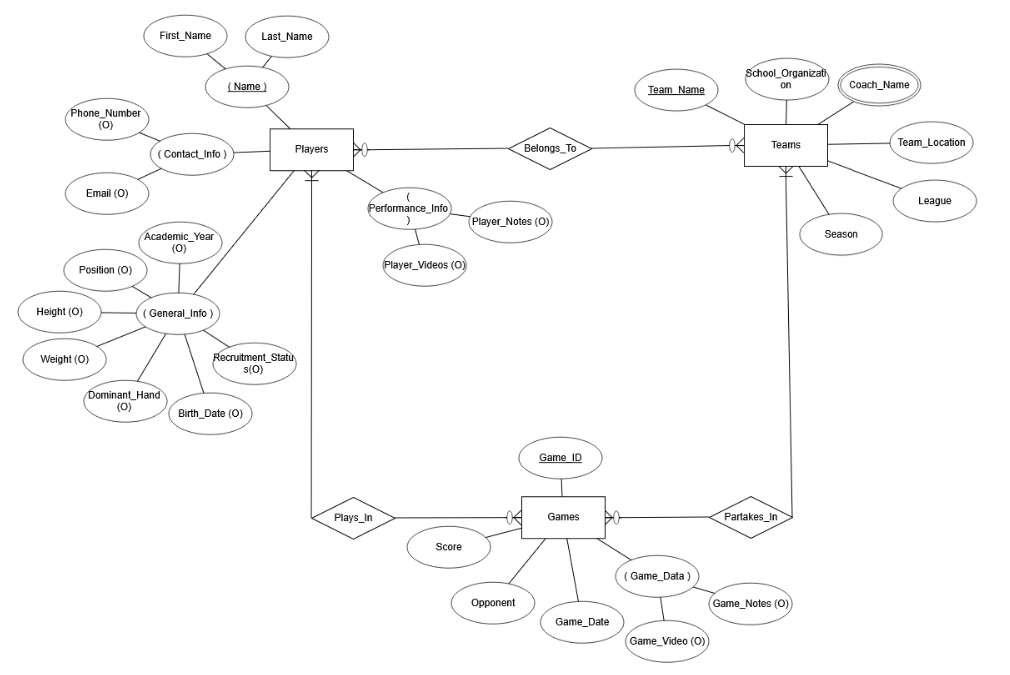


Figure 4: The entity relationship diagram for the hockey recruitment database.

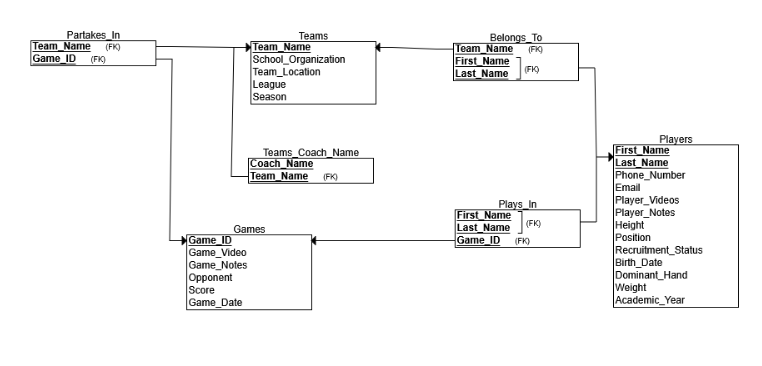


Figure 5: The relational schema for the hockey recruitment database.

Figure 4 and 5 depict the entity relationship diagram and relational schema for our database. These figures visually represent the structure our database tables will have. The ER diagram shows the three entities the database will record: the players, teams, and games. Each of these entities have their separate attributes.

* Players: Player ID, first and last names, phone number, email address, academic year, position, height, weight, dominant hand, birthdate, recruitment status, notes, and videos.
* Teams: name, school or organization, league, coach name(s), location, and season.
* Games: Score, date, opponent, video, and notes

The relational schema shows the tables (relations), their columns (which are made up of the entity attributes), and the table relationships, including the foreign keys that link the tables together. The tables are as follows:

* Teams
* Games
* Players
* Belongs\_To (relationship between players and teams)
* Partakes\_In (relationship between teams and games)
* Plays\_In (relationship between players and games)
* Teams\_Coach\_Name (multivalued attribute)