


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.

 CLARKSON
GOLDEN KNIGHTS

WOMEN'S HOCKEY RECRUITMENT DATABASE

PLAYERSGAMESTEAMS

Player Edit



Name:Player 1

Rank:99

Number:#99

Phone:(555) 555-5555

Position:Forward

Email:player@clarkson.edu

Year:2026

DOB:06/07/2004

Teams

Team Website:<https://clarksonathletics.com/sports>

Coach Email:coach@clarkson.edu

NotesVideosStats

Note

player 1's notes

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

2

CLARKSON GOLDEN KNIGHTS WOMEN'S HOCKEY RECRUITMENT DATABASE

PLAYERS GAMES

Team A vs. Team B

TEAM A

#5 PLAYER 1
POSITION: FORWARD
GRADUATION YEAR: 2025
NOTES: Top scorer

#28 PLAYER 2
POSITION: DEFENSE
GRADUATION YEAR: 2026
NOTES: Strong defense

TEAM B

#82 PLAYER 3
POSITION: GOALIE
GRADUATION YEAR: 2027
NOTES: Strong player

#99 PLAYER 4
POSITION: FORWARD
GRADUATION YEAR: 2025
NOTES: Fast

Figure 2: Game view showing two teams side by side. This allows for teams to be easily compared. It also allows for notes on players to be written.

CLARKSON GOLDEN KNIGHTS WOMEN'S HOCKEY RECRUITMENT DATABASE

PLAYERS GAMES TEAMS

Search by name All Teams All Positions Search by Grad. Year Add Player

Player Info

#5 Player 1

Team: Team A

Position: Forward

Graduation Year: 2025

Notes: Add notes here...

Save

Player Info

#28 Player 2

Team: Team A

Position: Defense

Graduation Year: 2026

Notes: Add notes here...

Save

Player Info

#82 Player 3

Team: Team B

Position: Goalie

Graduation Year: 2027

Notes: Add notes here...

Save

Player Info

#99 Player 4

Team: Team B

Position: Forward

Graduation Year: 2025

Notes: Add notes here...

Save

Player Info

#13 Player 5

Team: Team C

Position: Defense

Graduation Year: 2027

Notes: Add notes here...

Save

Player Info

#15 Player 6

Team: Team C

Position: Forward

Graduation Year: 2026

Notes: Add notes here...

Save

Figure 3: “Big list” view allows searching for players based on name, team, position, and graduation year. This page also allows players to be added to the database, notes on players to be written, and individual player info pages to open.

CLARKSON WOMEN'S HOCKEY RECRUITMENT DATABASE

PLAYERS GAMES TEAMS

Add Team ▾

Team Name:

Coach Name: First Last

Save

Figure 4: Team view where you can add teams to the database.

Django will serve a 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. A selection of implemented API paths, as well as some sample responses are included below.

`api/get/player`

- **GET** `/api/player/get?id=`
 - Returns a JSON object representing a player, containing a list of all teams

GET `api/get/player?id=0`

```
{
  "status": "success",
  "data": {
    "id": 0,
    "first_name": "Luke",
    "last_name": "Levine",
    "date_of_birth": "2004-10-17",
    "position": "Defence",
    "teams": [
      {
        "id": 0,
        "name": "Bruins",
        "coach_first_name": "John",

```

```

    "coach_last_name": "Smith",
    "coach_email": "jsmith@gmail.com",
    "last_updated": "2025-10-17"
  ]
}

```

api/get/team

- **GET** /api/get/team?id=
 - Returns a JSON object representing a team containing a list of players.

GET /api/team/get?id=0

```

{
  "status": "success",
  "data": {
    "id": 0,
    "name": "Bruins",
    "coach_first_name": "John",
    "coach_last_name": "Smith",
    "coach_email": "jsmith@gmail.com",
    "players": [
      {
        "id": 0,
        "first_name": "Luke",
        "last_name": "Levine",
        "date_of_birth": "2004-10-17",
        "position": "Defence",
        "number_on_team": 10,
        "last_updated": "2025-10-17"
      },
      {
        "id": 1,
        "first_name": "Bob",
        "last_name": "Levine",
        "date_of_birth": "2000-01-01",
        "position": "Goalie",
        "number_on_team": 12,
        "last_updated": "2025-09-23"
      }
    ]
  }
}

```

api/get/note

- **GET** /api/get/note?id=

api/search/player

- **GET** /api/search/player?first_name=a&last_name=b
 - Returns a JSON list of all players matching a query string. The search can be run with only one search parameter or both
 - A parameter of all can be used to get all players in the database.

GET /api/search/player?first_name=Luke

```
{
  "status": "success",
  "data": [
    {
      "id": 0,
      "first_name": "Luke",
      "last_name": "Levine",
      "date_of_birth": "2004-10-17",
      "position": "Defence"
    }
  ]
}
```

GET /api/search/player?all

```
{
  "status": "success",
  "data": [
    {
      "id": 0,
      "first_name": "Luke",
      "last_name": "Levine",
      "date_of_birth": "2004-10-17",
      "position": "Defence"
    },
    {
      "id": 1,
      "first_name": "Bob",
      "last_name": "Levine",
      "date_of_birth": "2000-01-01",
      "position": "Goalie"
    }
  ]
}
```

```
}
```

api/search/team

- **GET** /api/team/search?name=
 - Returns a JSON list of all teams where the name matches (TODO: partial match / fuzzy search?) NOTE: The results from this will not contain a list of players on the team. o get the players, look up the team on its own by id)

GET /api/team/search?name=Bruins

```
{
  "status": "success",
  "data": [
    {
      "id": 0,
      "name": "Bruins",
      "coach_first_name": "John",
      "coach_last_name": "Smith",
      "coach_email": "jsmith@gmail.com"
    }
  ]
}
```

api/search/note

- **GET** /api/note/search?query=
 - Returns a JSON list of all notes matching a query string.

api/update/player

- **POST** /api/player/update
 - Update fields on a player with the provided ID. Multiple fields can be updated at the same time. All fields are required as returned by GET /api/get/player

e.g. POST /api/player/update

BODY

```
{
  "id": 0,
```

```
"first_name": "Luke",
"last_name": "Levine",
"date_of_birth": "2004-10-17",
"position": "Forward" # This is the modified field
}
```

This will update the player ID 0 to have the values provided

api/update/team

- **POST** /api/team/update
 - 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.

api/update/team_membership

- **POST** /api/update/team_membership
 - Post a JSON object containing a player_id, team_id, and number_on_team.

api/update/note

- **POST** /api/note/update
 - 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.

api/create/player

- **POST** /api/player/create
 - 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.

api/create/team

- **POST** /api/team/create
 - 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.

api/create/team_membership

- **POST** /api/create/team_membership

- Post a JSON object containing a `player_id`, `team_id`, and `number_on_team`.

`api/create/note`

- **POST** `/api/note/create`
 - 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.

`api/delete/player`

- **DELETE** `/api/delete/player?id=`
 - Send a DELETE request to the above URI, with the player id in the id field.
 - Returns a JSON response confirming the delete.

`api/delete/team`

- **DELETE** `/api/delete/team?id=`
 - Send a DELETE request to the above URI, with the team id in the id field.
 - Returns a JSON response confirming the delete.

`api/delete/team_membership`

- **POST** `/api/delete/team_membership`
 - Send a JSON request containing the `player_id` and `team_id`.
 - Returns a JSON response confirming the delete.

`api/delete/note`

- **DELETE** `/api/delete/note?id=`
 - Send a DELETE request to the above URI, with the note id in the id field.
 - Returns a JSON response confirming the delete.

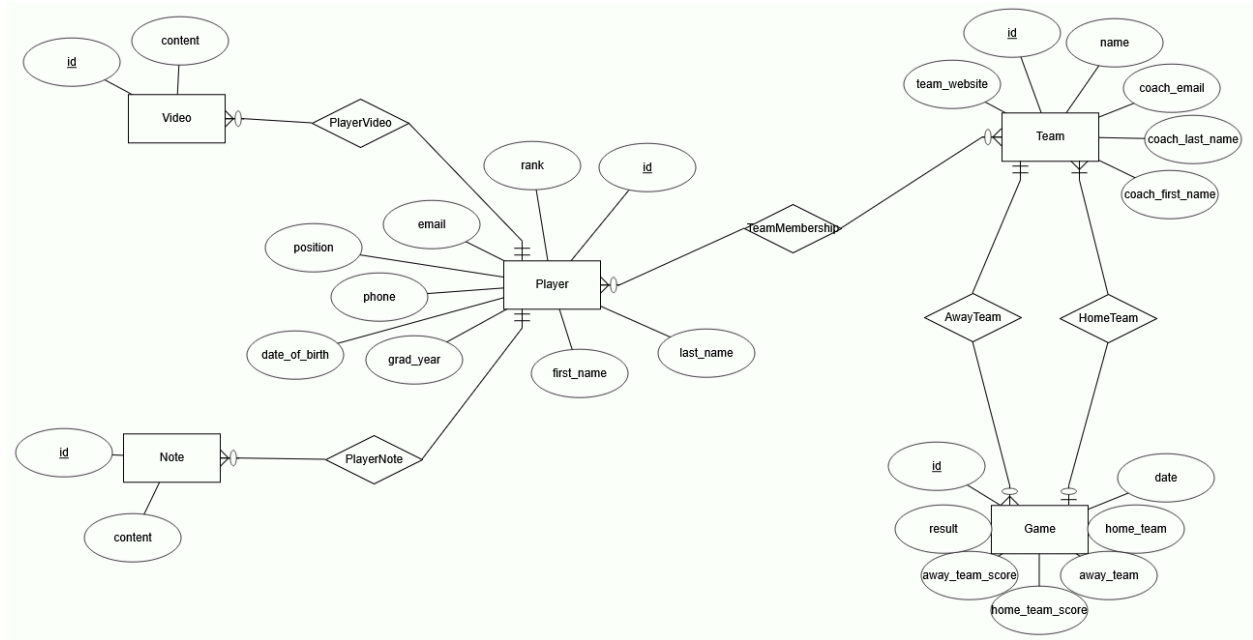


Figure 4: The entity relationship diagram for the hockey recruitment database.

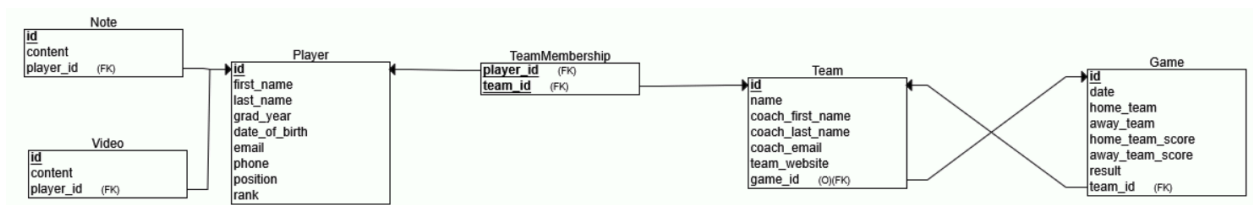


Figure 5: The relational schema for the hockey recruitment database.

Figures 4 and 5 depict the entity relationship diagram and relational schema for our database. These figures visually represent the structure that our database tables will have. The ER diagram in Figure 4 shows the five entities the database will record: the players, teams, games, player notes, and player videos. Each of these entities has its separate attributes.

- Players: Unique ID, first and last names, graduation year, date of birth, phone number, email address, position, and rank.
- Teams: Unique ID, name, coach first name, coach last name, coach email, and team website.
- Games: Unique ID, date of game, home team, away team, home team's score, away team's score, and result.
- Notes: Unique ID and content.
- Videos: Unique ID and content.

The ER diagram also depicts the entity relationships.

- PlayerVideo: A relationship between players and videos. A player can have many or no videos. A video must have one and only one player.
- PlayerNote: A relationship between players and notes. A player can have many or no notes. A note must have one and only one player.
- TeamMembership: A relationship between players and teams. A player can have one or many teams. A team can have many or no players.
- AwayTeam: A relationship between teams and games. A team can have many or no away games. A game must have one and only one away team.
- HomeTeam: A relationship between teams and games. A team can have many or no home games. A game must have one and only one home team.

The relational schema in Figure 5 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 and their foreign keys are as follows:

- Players
- Teams
 - Foreign key: game_id (connecting games to teams - optional)
- Games
 - Foreign key: team_id (connecting teams to games)
- TeamMembership
 - Foreign keys:
 - player_id (connecting players to teams)
 - team_id (connecting teams to players)
- Notes
 - Foreign key: player_id (connecting players to notes)
- Videos
 - Foreign key: player_id (connecting players to videos)