I have used Google Gemini which is part of firebase to help me create the correct rules for my database on the backend. These are the rules they suggested for my specific app and I will adjust them if need be.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    // --- General Rule for all data: Must be authenticated ---
    // Any path not explicitly matched below will fall to this, or be implicitly denied.
    // However, it's best to be explicit for your main collections.
    match /{document=**} {
      allow read, write: if request.auth != null;
    }

    // --- Specific Rules for 'teams' collection ---
    // Authenticated users can read team details (e.g., to select a team)
    match /teams/{teamId} {
      allow read: if request.auth != null;

      // Authenticated users can create and update team information.
      // We'll add a simple validation: 'name' must be a non-empty string.
      allow create, update: if request.auth != null
                            && request.resource.data.name is string
                            && request.resource.data.name.size() > 0;

      // Generally, disallow deletion of teams by regular users in a project
      allow delete: if false;
    }

    // --- Specific Rules for 'players' sub-collection within each team ---
    // Authenticated users can read player details (e.g., to search for a person)
    match /teams/{teamId}/players/{playerId} {
      allow read: if request.auth != null;

      // Authenticated users can create new players and update existing player details.
      // Simple validation: 'name' must be a non-empty string.
      allow create, update: if request.auth != null
                            && request.resource.data.name is string
                            && request.resource.data.name.size() > 0;

      // Disallow deletion of players by regular users
      allow delete: if false;
    }

    // --- Specific Rules for 'records' sub-collection within each team ---
```

```
    // Authenticated users can read record details (e.g., to search for a record)
    match /teams/{teamId}/records/{recordId} {
      allow read: if request.auth != null;

      // Authenticated users can create new records and update existing record details
      // (e.g., if someone breaks a record).
      // Simple validation: 'name' must be a non-empty string.
      allow create, update: if request.auth != null
                            && request.resource.data.name is string
                            && request.resource.data.name.size() > 0;

      // Disallow deletion of records by regular users
      allow delete: if false;
    }
  }
}
```

In doing research about putting the text files into the database. I have found that I needed to use my security code from my database into python and use a technique called parse. I used chat GPT to help me understand how to implement the security code into python.

✅ Step 1.

You clicked "Generate new private key" in Firebase, and your browser downloaded a file that ends with .json.
It probably looks something like:

my-project-123456-firebase-adminsdk-abc12-34def56789.json

✅ Step 2. Find where it downloaded

Open your computer's Downloads folder and locate that .json file.

On a Mac, it's usually at:

/Users/yourname/Downloads/

✅ Step 3. Move it into your Python project folder

Now, you need to move (or copy) that file into the same folder as your Python script.

For example, if your project folder looks like this:

```
Documents/
└── firebase_upload/
    ├── upload_to_firestore.py   ← your Python script
```

├── serviceAccountKey.json ← your Firebase key (move it here)
└── text_files/
├── team1.txt
├── team2.txt

👉 Rename the key file to something simple like:

serviceAccountKey.json

That makes it easier to reference in your Python code.

✅ Step 4. Update your Python script to match the location

In your upload_to_firestore.py, make sure this line matches the name of the JSON file you placed in the same folder:

cred = credentials.Certificate('serviceAccountKey.json')

I realized that I needed to do patterns for putting the text files into the database correctly so I looked up what the best patterns would be.

```
patterns = {
    # Special cases - these must come FIRST (most specific patterns first)
    "player_multiple_years": re.compile(

r"^(\d+)\.?\s+(\d+)\s*[–-]\s*([^,]+),\s*(\d{4}(?:\s*-\s*\d{2,4})?(?:,\s*\d{4}(?:\s*-\s*\d{2,4})?)
+)\s*$",
        re.IGNORECASE
    ),
    "player_multiple_years_with_details": re.compile(

r"^(\d+)\.?\s+([\d\.]+)\s*[–-]\s*([^,]+),\s*(\d{4}(?:\s*-\s*\d{2,4})?(?:,\s*\d{4}(?:\s*-\s*\d{2,4})
?)+)\s*\(([^)]+)\)\s*$",
        re.IGNORECASE
    ),
    "game_with_parenthetical_info": re.compile(
        r"^(\d+)\.?\s+(\d+)\s*[–-]\s*([^,]+),\s*(\d{4}(?:\s*-\s*\d{2,4})?)\s*\((.+?)\)\s+(.+)$",
        re.IGNORECASE
    ),

    # Game format with opponent that has state in parentheses
```

```
    "game_with_state_opponent": re.compile(

r"^(\d+)\.?\s+(\d+)\s*[–-]\s*([^,]+?)\s+(?:vs\.?|at)\s+([^,]+?\([A-Za-z\.]+\)),?\s*\(([^)]+)\)\s*$
",
        re.IGNORECASE
    ),

    # Hits game format
    "hits_game_format": re.compile(

r"^(\d+)\.?\s+(\d+)\s*[–-]\s*([^,]+?)\s+(?:vs\.?|at)\s+([^(]+?)\s*\((\d+-\d+)\),?\s*\(([^)]+)\)\s*
$",
        re.IGNORECASE
    ),

    # Team game format with value
    "team_game_with_value": re.compile(
        r"^(\d+)\.?\s+(\d+)\s+(?:vs\.?|at)\s*([^,]+),\s*(\d{1,2}/\d{1,2}/\d{2,4})\s*$",
        re.IGNORECASE
    ),

    # Team game format with dash (no vs.)
    "team_game_with_dash": re.compile(
        r"^(\d+)\.?\s+(\d+)\s*[–-]\s*([^,]+),\s*(\d{1,2}/\d{1,2}/\d{2,4})\s*$",
        re.IGNORECASE
    ),

    "team_vs_format": re.compile(
        r"^(\d+)\.?\s+(\d+)\s+(?:vs\.?|at)\s+([^,]+),\s*\(([^)]+)\)\s*$",
        re.IGNORECASE
    ),

    # Year range with possible spaces
    "stat_year_range_with_spaces": re.compile(
        r"^(\d+)\.?\s+([\d\.]+)\s*[–-]\s*(\d{4})\s*-\s*(\d{2,4})\s*$",
        re.IGNORECASE
    ),

    "stat_year_only": re.compile(
        r"^(\d+)\.?\s+([\d\.]+)\s*[–-]\s*(\d{4})\s*$",
```

```
    re.IGNORECASE
),

# Wins with record
"wins_with_record": re.compile(
    r"^(\d+)\.?\s+(\d+)\s*[—-]\s*([^,]+?),?\s*(\d{4})\s*\((\d+-\d+)\)\s*$",
    re.IGNORECASE
),

"simple_player_year_no_value": re.compile(
    r"^(\d+)\s*[—-]\s*([^,]+),\s*(\d{4})\s*$",
    re.IGNORECASE
),
"complex_game_list": re.compile(
    r"^(\d+)\.?\s+(\d+)\s*[—-]\s*([^,]+),\s*(\d{4}),\s*(.+)$",
    re.IGNORECASE
),

# Original patterns
"ratio_stat": re.compile(
    r"^(\d+)\.?\s+(\d+/[\d\.]+)\s*[—-]\s*(.+?)\s*\(([^)]+)\)\s*$",
    re.IGNORECASE
),
"game": re.compile(
    r"^(\d+)\.?\s+(\d+)\s*[—-]\s*([^–]+?)\s+(?:vs\.?|at)\s*([^(]+?)\s*\(([^)]+)\)\s*$",
    re.IGNORECASE
),
"team": re.compile(
    r"^(\d+)\.?\s+(?:vs\.?|at)\s*([^,]+),\s*(\d{1,2}/\d{1,2}/\d{2,4})\s*$",
    re.IGNORECASE
),

# Career
"career": re.compile(
    r"^(\d+)\.?\s+(\d+)\s*[—-]\s*([^,]+),\s*(\d{4}(?:\s*-\s*\d{2,4})?)\s*$",
    re.IGNORECASE
),

"player_year": re.compile(
    r"^(\d+)\.?\s+(\d+)\s*[—-]\s*([^,]+),\s*(\d{4})\s*$",
```

```
        re.IGNORECASE
    ),
    "stat_with_year_and_details": re.compile(
        r"^(\d+)\.?\s+([\d\.]+)\s*[—]\s*([^,]+),\s*(\d{4}(?:\s*-\s*\d{2,4})?)\s*\(([^)]+)\)\s*$",
        re.IGNORECASE
    ),
    "stat_with_details": re.compile(
        r"^(\d+)\.?\s+([\d\.]+)\s*[—]\s*([^,\(]+?)\s*\(([^)]+)\)\s*$",
        re.IGNORECASE
    ),
    "stat_with_year": re.compile(
        r"^(\d+)\.?\s+([\d\.]+)\s*[—]\s*([^,]+),\s*(\d{4}(?:\s*-\s*\d{2,4})?)\s*$",
        re.IGNORECASE
    ),
    "year_value": re.compile(
        r"^(\d+)\.?\s*[—]\s*(\d{4})\s*$",
        re.IGNORECASE
    ),
    "simple": re.compile(
        r"^(\d+)\.?\s*[—]\s*(.+)\s*$",
        re.IGNORECASE
    )
}
```

I also needed help to create the correct groups using these patterns so it showed me this and I figured out how to apply it.

```
 for ptype, pattern in patterns.items():
                m = pattern.match(line)
                if not m:
                    continue

                matched = True
                groups = m.groups()
                record = {"type": ptype, "raw_text": line}
```

```
if ptype == "player_multiple_years":
    rank, value, player, years = groups
    record.update({
        "rank": int(rank),
        "value": int(value),
        "player": player.strip(),
        "years": years.strip()
    })

elif ptype == "player_multiple_years_with_details":
    rank, value, player, years, details = groups
    record.update({
        "rank": int(rank),
        "value": value.strip(),
        "player": player.strip(),
        "years": years.strip(),
        "details": details.strip()
    })

elif ptype == "game_with_parenthetical_info":
    rank, value, player, years, game_info, additional = groups
    record.update({
        "rank": int(rank),
        "value": int(value),
        "player": player.strip(),
        "years": years.strip(),
        "game_info": game_info.strip(),
        "additional": additional.strip()
    })

elif ptype == "game_with_state_opponent":
    rank, value, player, opponent, date = groups
    record.update({
        "rank": int(rank),
        "value": int(value),
        "player": player.strip(),
        "opponent": opponent.strip(),
        "date": date.strip()
    })
```

```
elif ptype == "hits_game_format":
    rank, value, player, opponent, hit_record, date = groups
    record.update({
        "rank": int(rank),
        "value": int(value),
        "player": player.strip(),
        "opponent": opponent.strip(),
        "hit_record": hit_record.strip(),
        "date": date.strip()
    })

elif ptype == "team_game_with_value":
    rank, value, opponent, date = groups
    record.update({
        "rank": int(rank),
        "value": int(value),
        "opponent": opponent.strip(),
        "date": date.strip()
    })

elif ptype == "team_game_with_dash":
    rank, value, opponent, date = groups
    record.update({
        "rank": int(rank),
        "value": int(value),
        "opponent": opponent.strip(),
        "date": date.strip()
    })

elif ptype == "team_vs_format":
    rank, value, opponent, date = groups
    record.update({
        "rank": int(rank),
        "value": int(value),
        "opponent": opponent.strip(),
        "date": date.strip()
    })

elif ptype == "stat_year_range_with_spaces":
    rank, value, year_start, year_end = groups
```

```
            record.update({
                "rank": int(rank),
                "value": value.strip(),
                "years": f"{year_start}-{year_end}"
            })

        elif ptype == "stat_year_only":
            rank, value, year = groups
            record.update({
                "rank": int(rank),
                "value": value.strip(),
                "year": year.strip()
            })

        elif ptype == "wins_with_record":
            rank, value, player, year, win_record = groups
            record.update({
                "rank": int(rank),
                "value": int(value),
                "player": player.strip(),
                "year": year.strip(),
                "win_record": win_record.strip()
            })

        elif ptype == "simple_player_year_no_value":
            rank, player, year = groups
            record.update({
                "rank": int(rank),
                "player": player.strip(),
                "year": year.strip()
            })

        elif ptype == "complex_game_list":
            rank, value, player, year, game_details = groups
            record.update({
                "rank": int(rank),
                "value": int(value),
                "player": player.strip(),
                "year": year.strip(),
                "game_details": game_details.strip()
```

```
        })

    elif ptype == "ratio_stat":
        rank, ratio, info, details = groups
        record.update({
            "rank": int(rank),
            "ratio": ratio.strip(),
            "info": info.strip(),
            "details": details.strip()
        })

    elif ptype == "game":
        rank, value, player, opponent, date = groups
        record.update({
            "rank": int(rank),
            "value": int(value),
            "player": player.strip(),
            "opponent": opponent.strip(),
            "date": date.strip()
        })

    elif ptype == "career":
        rank, value, player, years = groups
        record.update({
            "rank": int(rank),
            "value": int(value),
            "player": player.strip(),
            "years": years.strip()
        })

    elif ptype == "team":
        rank, opponent, date = groups
        record.update({
            "rank": int(rank),
            "opponent": opponent.strip(),
            "date": date.strip()
        })

    elif ptype == "player_year":
        rank, value, player, year = groups
```

```
        record.update({
            "rank": int(rank),
            "value": int(value),
            "player": player.strip(),
            "year": year.strip()
        })

    elif ptype == "stat_with_year_and_details":
        rank, value, player, year, details = groups
        record.update({
            "rank": int(rank),
            "value": value.strip(),
            "player": player.strip(),
            "year": year.strip(),
            "details": details.strip()
        })

    elif ptype == "stat_with_details":
        rank, value, player, details = groups
        record.update({
            "rank": int(rank),
            "value": value.strip(),
            "player": player.strip(),
            "details": details.strip()
        })

    elif ptype == "stat_with_year":
        rank, value, player, year = groups
        record.update({
            "rank": int(rank),
            "value": value.strip(),
            "player": player.strip(),
            "year": year.strip()
        })

    elif ptype == "year_value":
        rank, year = groups
        record.update({
            "rank": int(rank),
            "year": year.strip()
```

```
        })

    elif ptype == "simple":
        rank, info = groups
        record.update({
            "rank": int(rank),
            "info": info.strip()
        })
```

I used AI to help me figure out how to put an image for my login screen and sign up screen just so it would look a little more professional.

Add it to the LoginView

At the very end of your VStack in LoginView, right before the closing brace, add:

```
Spacer()

Image("lawrence_logo")
    .resizable()
    .scaledToFit()
    .frame(height: 120)
    .padding(.bottom, 20)
```

Do the same in SignUpView

Just add the same block right before the closing VStack brace:

```
Spacer()

Image("lawrence_logo")
    .resizable()
    .scaledToFit()
    .frame(height: 120)
    .padding(.bottom, 20)
```

I also wanted to change the problem of in my team records it was showing an Unknown which is my coded default for a player, but in team records there are no players so I wanted to know how to change it to a category instead of a player while keeping the names for all the other categories.

Replace this line:

```
let player = (d["player"] as? String) ?? (d["info"] as? String) ?? "Unknown"
```

With this:

```
var player = (d["player"] as? String)
        ?? (d["info"] as? String)
        ?? "Unknown"

// Team record fallback → show category/subcategory name instead of "Unknown"
if player == "Unknown" {
   player = subcategoryId.replacingOccurrences(of: "_", with: " ")
}
```

The rest of my AI use was just for errors. I had a lot of problems when doing this project and I thought that I could save some time by just trying to get to the bottom of the errors with the help of AI. I do not have all of the errors that I asked to help me fix, but here is what I do have below.

A lot of my errors were because I had missing imports with firebase..

The error Cannot find 'Auth' in scope just means you haven't imported the Firebase Auth framework in that file.

Let's fix it 👇

Add this import to the top of SportsSelectionView.swift

import SwiftUI
import FirebaseAuth // 👈 Add this line
import FirebaseFirestore

I am getting this error "Missing argument for parameter 'authVM' in call"

ChatGPT said:

Perfect — that's a really common one, and it's easy to fix 👏

That error means you're trying to create or navigate to a view that requires an authVM parameter (like this 👇):

SportsSelectionView(authVM: authVM)

…but somewhere in your code, you're calling just:

SportsSelectionView()

I had some errors when I would run my code and I wanted to make sure it didn't mean anything bad.
I am receiving this error "nw_socket_set_connection_idle [C14.1.1.1:3] setsockopt SO_CONNECTION_IDLE failed [42: Protocol not available]" when I try to sign in or sign up

ChatGPT said:

Good catch — that error looks scary 😅 but the good news is:
✅ it's not actually breaking your sign-in or network connection.

As I mentioned I do not have a lot of what I did ask, but it was just solving error problems that I was having and also checking and running tests if my firebase was connecting to xcode correctly.