Premier Geek

Premier Geek is a Proof of Concept (POC) designed to provide users with accurate and up-to-date squad information for Premier League teams based on natural language queries. The application allows users to request detailed player information - including first names, surnames, dates of birth, and playing positions - by simply asking for it in plain language.

The goal of this POC is to demonstrate the integration of a **Large Language Model (LLM)** with third-party **Sport Monks** data sources to deliver reliable football squad details quickly and efficiently. This document outlines usage of Premier Geek, high-level design of the application, the architecture & coding techniques used underneath and the productization with future steps for the app.

The analysis in the documents shows that Premier Geek can evolve from a promising Proof of Concept into a robust, scalable, and user-friendly production application. The combination of **powerful language models**, **real-time sports data**, and **intuitive interaction modalities** offers a promising direction for **Premier League enthusiasts** seeking quick and accurate squad information.

Table Of Contents

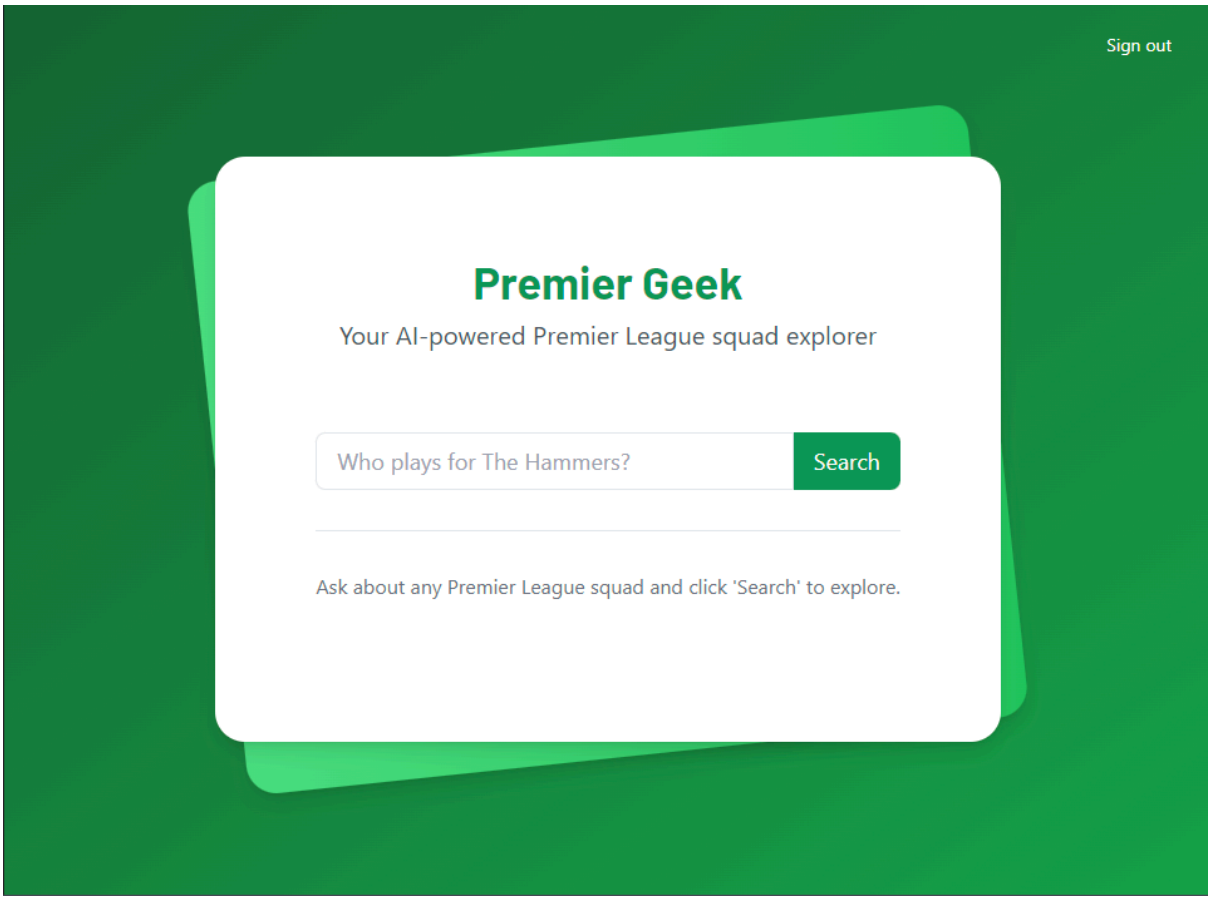
How Premier Geek Works

User Interaction Flow

Premier Geek provides a simple and intuitive interface for users to retrieve squad information about Premier League teams. Below is a step-by-step explanation of how the application processes user queries:

1. Entering the Website

When users first visit the Premier Geek website after logging in, they are greeted with a clean and straightforward interface. The homepage features a single input box where users can type their query. The interface is designed to be minimalistic, ensuring that users can focus on their primary task—asking about a Premier League team.



2. Asking a Question

Users can type their questions naturally, for example, "Who plays for The Hammers?" The application is designed to handle various ways of referring to a team—whether users use the full team name, a common abbreviation, or a nickname. This flexibility makes it easier for users to get the information they need without worrying about specific syntax.

3. Natural Language Processing

Once the user submits a query, the application leverages a Large Language Model (LLM) called Claude 3 Haiku to analyze the input. This step is crucial, as it involves understanding the user's intent and identifying the specific team they are asking about. The LLM is capable of recognizing various forms of team names:

- **Direct Names:** "West Ham United"
- **Abbreviations:** "West Ham", “WHU”
- **Nicknames:** "The Hammers"

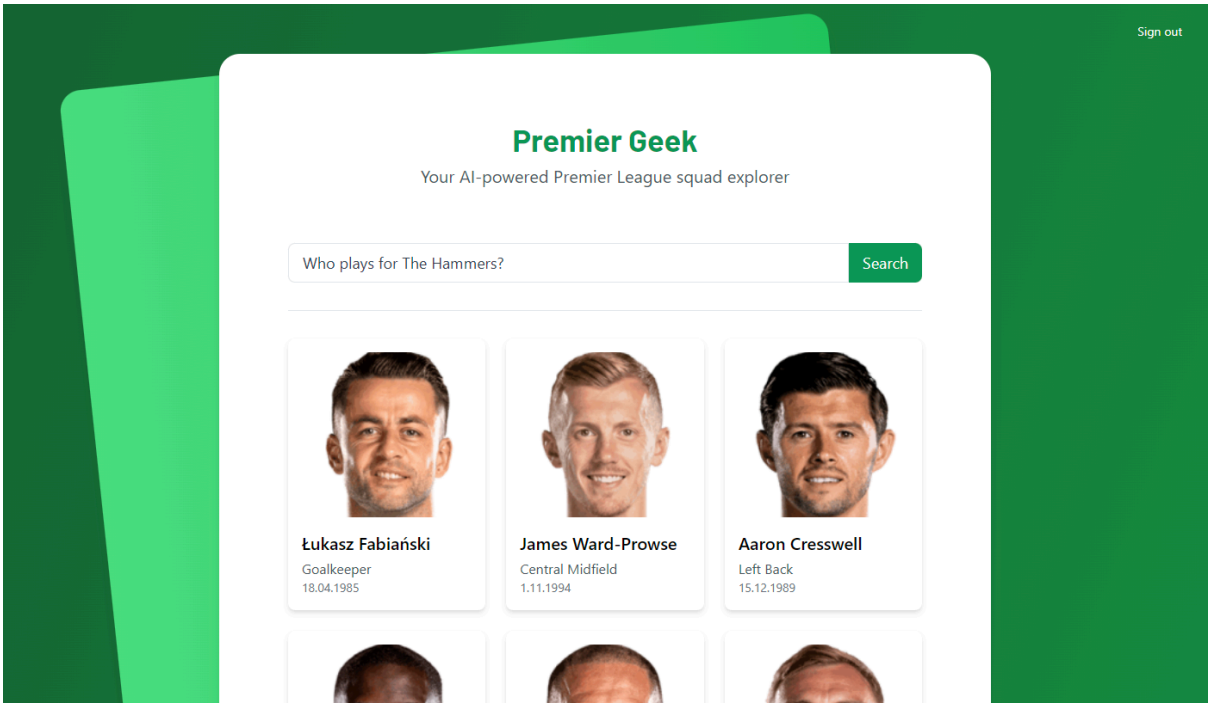
This robust recognition ensures that the application **accurately** interprets the query, even when the team name is presented in **less common forms**.

4. Team Identification and API Request

Once the team is identified, the application makes a call to the **Sport Monks API**. This API provides real-time data about Premier League squads. The application requests information specifically about the players currently on the roster of the identified team. The data retrieved includes the players' first names, surnames, dates of birth, and playing positions.

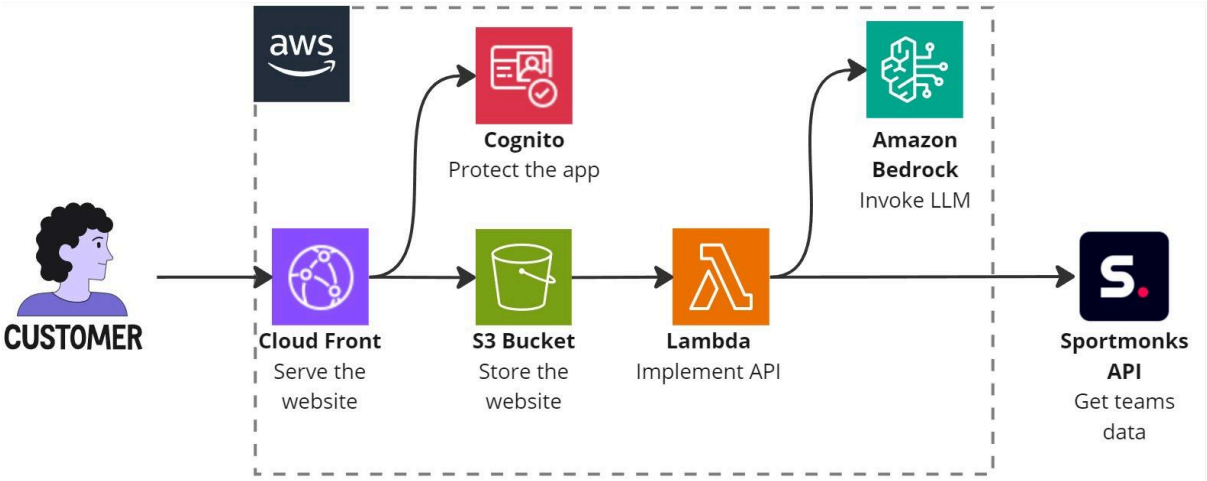
5. Displaying the Results

The results are then displayed on the screen. Each player's information is presented in a card format, making it easy for users to browse through the squad list. If the user’s query does not correspond to a Premier League team, the application will prompt the user to refine their question, ensuring that only relevant and accurate information is provided.



Infrastructure and Frameworks Overview

A high-level infrastructure overview is shown on the diagram below. In natural words, the user enters the website that is stored on an **S3 Bucket** and served through a **CloudFront Distribution**. The user needs to log in using **Amazon Cognito** so that he can begin interacting with the API deployed to a **Lambda Function**. The API calls both the **Sportmonks API** to retrieve the teams & players information as well as the model hosted on **Amazon Bedrock** to analyze user's query.



1. Amazon CloudFront & S3

- **Purpose**:
  - Amazon S3 hosts the static **website** for Premier Geek.
  - CloudFront provides **HTTPS** and caching, enhancing security and performance.
- **Rationale**:
  - S3 is a cloud-native service ideal for hosting static websites due to its **ease to use** and **low cost**, making it perfect for Proof of Concept (PoC) projects as well as future productization.
  - CloudFront ensures **secure data transmission** (HTTPS), which is important even for PoC to incorporate **security best practices**.
- **Alternatives**:
  - Self-managed hosting (e.g., EC2 or ECS) could be considered but dismissed due to **complexity** and **increased costs**, which are unnecessary at the PoC stage.

2. Amazon Cognito

- **Purpose**:
  - Provides user authentication and authorization, **securing** both the **website** and **API access** from unwanted visitors.
- **Rationale**:
  - Cognito is **easy to set up**, **integrates seamlessly** with other AWS services, and comes with a **generous free tier,** making it an excellent choice for PoC.
- **Alternatives**:
  - **Less secure** auth methods like Basic Auth.
  - Open-source, self-manages solutions like Keycloak but they would require **more time to configure and manage**, which isn't optimal for a PoC.

3. AWS Lambda & FastAPI

- **Purpose**:
  - Hosts the API that processes user queries and interacts with other services.
- **Rationale**:
  - AWS Lambda is a serverless compute service that requires little management effort, scales automatically and charges only for the compute time used, making it **fast to use**, **low-maintenance**, **cost-effective** and good for a PoC.
  - FastAPI was chosen for its **simplicity** and **speed**, allowing for **rapid development** while still being robust enough to **handle future production needs.**
- **Alternatives**:
  - Containerized solutions or virtual machines would require **more setup**, **maintenance**, and **cost**.
  - Other frameworks like Django or Flask could be considered but they have **higher complexity**, which isn't necessary for this PoC.

4. Amazon Bedrock (Claude 3 Haiku)

- **Purpose**:
  - **Processes natural language queries** to identify the Premier League team the user is asking about.
- **Rationale**:
  - Claude 3 Haiku is a **fast** and **cost-effective** language model, suitable for the straightforward task of team identification.
  - Bedrock offers **managed hosting** for the model, ensuring reliable performance **without the need** for complex infrastructure management.
- **Alternatives**:
  - More advanced models could be considered but they offer **higher cost** and **longer processing times**, which aren't necessary given the task's simplicity.

5. Sport Monks API

- **Purpose**:
  - Retrieves up-to-date information about Premier League squads, including **player names, positions**, **dates of birth**, and **photos**.
- **Rationale**:
  - Sport Monks API offers im**mediate access** to the necessary data, has **up-to-date information**, and provides a **14-day trial**, making it ideal for PoC.
- **Alternatives**:
  - Other APIs were considered but either **lacked real-time data**, **required multiple API** calls to retrieve the same information, or required making **many more API calls** to retrieve the same amount of information.

Estimated costs breakdown

Assumptions

- Production environment (free tier not considered)
- Only backend deployed to production (frontend & auth are in existing system)
- Traffic: **1,000,000** requests per month (e.g. **30,000 users** asking one question every day)

Cost Breakdown (per 1,000,000 requests)

Claude Haiku Model

- Prompt input: 220 tokens * $0.00025/1000 tokens = $0.000055 per request
- Prompt output: 1 token * $0.00125/1000 tokens = $0.00000125 per request
- Total per request: $0.00005625
- Monthly total: $0.00005625 * 1,000,000 requests = **$56.25**

Lambda API

- Compute: 128MB * 0.75s * $0.0000166667/GB-s = $0.000001562503125 per request
- Requests fee: $0.2 per 1M requests
- Monthly total: ($0.000001562503125 * 1,000,000) + $0.2 = **$1.76**

Network:

- 6.48KB per request = 0.0000061798095703125 GB per request
- $0.09 per GB
- Monthly total: 0.0000061798095703125 * $0.09 * 1,000,000 = $0.56

External API (Sport Monks): $66.00 (fixed monthly cost)

Total Monthly Cost per 1,000,000 requests

$56.25 (model) + $1.76 (compute) + $0.56 (network) + $66.00 (external API) = **$124.57**

Code Commentary

1. Repository URL

The source code for this project, including all relevant documentation, can be found in the GitHub repository: https://github.com/kucharzyk-sebastian/premier-geek

2. Development & Deployment Process

The entire development and deployment process is comprehensively documented in the README.md file within the repository. It includes step-by-step instructions on setting up the environment, deploying the application, and running tests. This ensures that anyone reviewing the project can easily replicate and understand the setup.

3. Prompt Engineering Techniques

Prompt Used

Below is the prompt used for the language model to correctly identify the English Premier League (EPL) team referred to in the user's query:

LLM Prompt

```
Your job is to identify the team's name from the English Premier League that the user
refers to in their query.  If you can't assign a team to the user's query, reply with
"None". For example:

Q: What is Manchester United's squad?
A: 14
Q: Show me The Hammers squad.
A: 1
Q: Who plays for Chelsea?
A: 18
Q: Do you like football?
A: None

Here is a JSON list of teams with corresponding IDs currently playing in the English
Premier League. You can use only those.
```
{teams}
```

You should reply only with the team ID. Don't add any more information.
```

Techniques Used

1. Few-shot Prompting

- **Purpose**: To guide the model in identifying the team name from the user's query.
- **Method**: Examples of potential user queries are provided within the prompt to give the model a clear understanding of the expected input-output behavior.
- **Rationale**: Few-shot examples demonstrate to the model **how to handle different types of queries**, including both direct team names and nicknames, enhancing its ability to make accurate predictions.

2. Retrieval-Augmented Generation (RAG)

- **Purpose**: To dynamically inject the **most up-to-date** list of EPL teams into the prompt.
- **Method**: The *{teams}* placeholder in the prompt is populated with a JSON object containing the current EPL teams and their corresponding IDs. This information is retrieved from the Sport Monks API.
- Sample JSON:

- **Rationale**: By embedding the list of current teams directly into the prompt, the model is constrained to choose from a **valid and up-to-date set of options**. This minimizes the chances of incorrect or irrelevant outputs, ensuring the accuracy of the model's predictions.

Productization and Future Steps

Premier Geek shows a lot of promise as a Proof of Concept, demonstrating the potential of **integrating Large Language Models with real-time sports data** to provide users with an intuitive and efficient way to access Premier League squad information. To take this application to the next level and prepare it for production, several optimizations and expansions could be considered.

Optimizations

1. Content Moderation and Safety

Optimize the prompt and implement additional **safeguards** to protect against the use of **toxic nicknames** for clubs or other **harmful content**. This could involve **fine-tuning** the language model to better recognize and filter out inappropriate language, as well as employing pre- and post-processing mechanisms like **guardrails** to ensure the safety and appropriateness of the generated responses.

2. Query Scope Optimization

Enhance the prompt to better **handle questions that fall outside the scope** of the team's squad information. For example, if a user asks about a team's stadium or history, the application should gracefully guide the user back to the intended use case, which is retrieving player details. This optimization will ensure a more focused and effective user experience.

3. Caching and Performance Optimization

Implement a caching strategy to **reduce the frequency of API calls to Sport Monks** and improve the application's response time. One approach could be to periodically ingest player data into a fast and cost-effective storage solution like DynamoDB, and serve the cached data to users. The frequency of data ingestion (e.g., daily, weekly) can be adjusted based on the expected rate of squad changes.

4. Comprehensive Testing

Develop a robust testing suite to **ensure the reliability and performance** of the application. This could include unit tests for individual components, functional tests for API endpoints, smoke tests for critical paths, end-to-end tests for user flows, and specialized tests for evaluating the language model's outputs. A comprehensive testing strategy will be crucial for maintaining the application's quality and reliability as it scales.

5. Dynamic Season ID Retrieval

Currently, the season ID used to retrieve the list of teams is hardcoded for the POC. To enhance the app's accuracy and maintainability, implement dynamic retrieval.

Potential Feature Enhancements:

1. Conversational Mode

Expand the application's capabilities by implementing a **conversational mode**, allowing users to ask follow-up questions about a team or player. This could involve maintaining context across multiple user queries and generating more dynamic and tailored responses.

2. Expanded Player and Team Data

Enrich the application by incorporating **additional data points about players and teams**, such as player statistics, injury reports, or recent performance. This will enable users to engage in more detailed and insightful discussions about their favorite Premier League clubs.

3. Agent Capabilities

Explore the possibility of turning Premier Geek into an agent that can **perform actions on behalf of the user**. For example, the application could allow users to add all strikers from a given team to their favorites or generate custom player comparisons based on user-specified criteria.

4. Voice Interaction

Enhance the user experience by adding **voice-based interaction capabilities**. Users could speak their questions to the application, which would then process the audio input, generate a response, and deliver it back to the user via voice output. This would make Premier Geek more accessible and convenient, especially for users on the go.

5. LLM Framework Integration

As the LLM-related features of the app grow, consider transitioning to more robust frameworks like LlamaIndex or LangChain. These frameworks offer a wide range of tools and capabilities specifically designed for building LLM-powered applications.

6. Architecture Flexibility: Serverless to Serverful Migration

While the current serverless architecture using AWS Lambda provides excellent scalability and cost-efficiency, it's important to note that there's an easy path to transition to a serverful architecture if cold start latency becomes a concern. Amazon ECS (Elastic Container Service) with Fargate offers a straightforward migration option.