**INSIDER**

**Backend-Case Solution :Football League Simulation Project**

**Code Link & Documentation:**

[**https://github.com/Gokay1904/Insider-Backend-Hiringday-Task**](https://github.com/Gokay1904/Insider-Backend-Hiringday-Task)

For detailed setup instructions, project structure, and technical explanations, please refer to the ***README.md*** file in the repository.

**Submitted by:**  
**Gökay Akçay**  
İstanbul Technical University  
Faculty of Arts and Literature, Physics Engineering

**GITHUB:** https://github.com/Gokay1904  
📧 gokay.akcay19@gmail.com  
📞 +90 534 583 15 77

**Submission Date: 29.05.2025**

**Technologies Used:**  
Go (Golang), SQLite, REST API

**Contents**

[1.Overview 2](#_Toc199429302)

[2. The Problem 2](#_Toc199429303)

[3. The Solution 2](#_Toc199429304)

[4. Project Structure 3](#_Toc199429305)

[5. Database Design 3](#_Toc199429306)

[6. API Endpoints Table 4](#_Toc199429307)

## 1.Overview

This project is a **Go-based backend system** designed to simulate a football league with a realistic schedule and dynamic match results. It demonstrates core backend development skills such as API design, HTTP server management, SQLite database integration, and modular project architecture.

The system models teams, matches, and league standings, and exposes RESTful API endpoints to simulate matches weekly or all at once, retrieve live league standings, and reset the simulation. This backend service forms the core of a case study for the Insider Development Intern Hiring Day.

## 2. The Problem

In many football league management scenarios, testing outcomes, standings, or match schedules requires complex manual calculations or expensive software. The problem was to **automate the simulation of football matches** realistically, considering team strength, and to maintain accurate league standings dynamically. The challenge was ensuring the system could:

* Simulate multiple weeks of matches quickly
* Reflect realistic outcomes based on team strengths
* Expose a clear API for integration and testing

## 3. The Solution

To address this problem, the backend was developed using Go, leveraging its efficiency and concurrency capabilities. The solution included:

* Assigning each team a **strength rating** to influence match results.
* Using a SQLite database to **persist teams, matches, and standings**.
* Implementing algorithms that simulate matches weekly, awarding points based on standard football rules (win, draw, loss).
* Creating RESTful API endpoints to:
  + Simulate matches by week or all at once
  + Retrieve current league standings
  + Reset the simulation

This approach ensured easy integration, extensibility, and fast response times.

## 4. Project Structure

Insider-Backend-Casestudy

├── handlers/ # HTTP handlers for API endpoints

│ ├── match\_handler.go

│ └── table\_handler.go

├── models/ # Data models and interfaces

│ ├── interface.go

│ ├── match.go

│ └── team.go

├── router/ # HTTP router setup

│ └── router.go

├── services/ # Business logic for simulation & standings

│ ├── match\_service.go

│ ├── simulation\_service.go

│ └── table\_service.go

├── league.db # SQLite database file storing all league data

├── reset.sql # SQL script to reset the database (truncate all data)

├── schema.sql # SQL script to create tables

├── seed.sql # SQL script to seed initial teams & data

├── main.go # Application entry point

├── Dockerfile # Docker container setup (optional)

├── go.mod # Go module definition

└── go.sum # Go dependencies checksum  
5. Database Design

The backend uses **SQLite** to persist the league data (league.db). Two main tables:

| **Table Name** | **Column Name** | **Data Type** | **Description** |
| --- | --- | --- | --- |
|  | id | INTEGER (PK) | Unique identifier for each team |
|  | name | TEXT | Team name (e.g., Arsenal, Chelsea) |
|  | position | INTEGER | Current league position |
|  | played | INTEGER | Number of matches played |
| **teams** | won | INTEGER | Number of matches won |
|  | drawn | INTEGER | Number of matches drawn |
|  | lost | INTEGER | Number of matches lost |
|  | gf | INTEGER | Goals scored (Goals For) |
|  | ga | INTEGER | Goals conceded (Goals Against) |
|  | gd | INTEGER | Goal difference (gf - ga) |
|  | points | INTEGER | Total points earned |
|  | strength | INTEGER | Team strength rating (used in simulations) |
|  | id | INTEGER (PK) | Unique identifier for each match |
|  | week | INTEGER | Match week number |
| **matches** | home\_team\_id | INTEGER (FK) | Foreign key to teams.id for home team |
|  | away\_team\_id | INTEGER (FK) | Foreign key to teams.id for away team |
|  | home\_goals | INTEGER | Goals scored by home team |
|  | away\_goals | INTEGER | Goals scored by away team |
|  | result | TEXT | Match result description (e.g., "Home Win") |

## 6. API Endpoints Table

| **Endpoint** | **HTTP Method** | **Description** | **Request Body** | **Response** |
| --- | --- | --- | --- | --- |
| /simulate/week | POST | Simulates match results for the specified week (via query param) | None | JSON: Simulated matches of the week |
| /simulate/all | POST | Simulates all remaining weekly matches | None | JSON: All remaining match results |
| /standings | GET | Retrieves the current league table standings | None | JSON: Team standings |
| /reset | POST | Resets match results and standings to initial state | None | Plain text: Confirmation message |