Ariadne Management

Nikita Dubinin

CPSC 321, Fall 2024

1 Summary

Ariadne Management is a web app designed to help team managers to store and analyze team related data. It provides tools to conveniently store, visualize and analyze many different aspects of a racing team management.

2 Use Cases

2.1 User must be able to create an account

Users must be able to create an account that will be used to associate with other data. This use case involves creating data.

2.2 User must be able to create a team

Users must be able to create a team that will be used as a main object that will be associated with other objects. This use case involves creating data.

2.3 User must be able to create, update and delete a championship for a specific team

Users must be able to create a championship that users will be able to manage. This is the main object for data received from the track. This use case involves creating, updating and deleting data.

2.4 User must be able to create, update and delete a stage for a specific championship

Users must be able to create a stage that users will be able to manage. Stages contain sessions and will be associated with drivers in future. This use case involves creating, updating and deleting data.

2.5 User must be able to create, update and delete a session for a specific stage

Users must be able to create a session that users will be able to manage. Sessions contain laps and will be associated with drivers in future. This use case involves creating, updating and deleting data.

2.6 User must be able to create, update and delete a lap for a specific session

Users must be able to create a lap that users will be able to manage. Laps are the main unit for analysis. This use case involves creating, updating and deleting data.

2.7 User must be able to create, update and delete a car for a specific team

Users must be able to assign a car for a team that users will be able to manage. Cars functionality will be expanded in future. This use case involves creating, updating and deleting data.

2.8 User must be able to create, update and delete a part for a specific car

Users must be able to assign a part for a car that users will be able to manage. This use case is needed to manage the amount of parts in stock, so users will have a convenient way to track them. This use case involves creating, updating and deleting data.

2.9 User must be able to create, update and delete a tire for a specific car

Users must be able to assign a tire for a car that users will be able to manage. This use case is needed to manage the amount of tires and their condition in stock, so users will have a convenient way to track them. This use case involves creating, updating and deleting data.

2.10 User must be able to look for a best lap in a session

Users must be able to analyse any session for a best lap time. This use case will be expanded in future and the laps will be attached to a specific car. This use case involves analyzing data.

2.11 User must be able to look for an average lap in a session

Users must be able to analyse any session for an average lap time. This use case will be expanded in future and the laps will be attached to a specific car. This use case involves analyzing data.

2.12 User must be able to look for a number of parts assigned to a car

Users must be able to look for a number of parts assigned to a car. This use case involves analyzing data.

2.13 User must be able to look for a number of tires assigned to a car

Users must be able to look for a number of tires assigned to a car. This use case involves analyzing data.

2.16 User must be able to look for an average tire wear of a specific car

Users must be able to look for an average tire wear of a specific car. This use case involves analyzing data.

2.17 User must be able to look for a number of tires with tread remaining higher than x and compound y

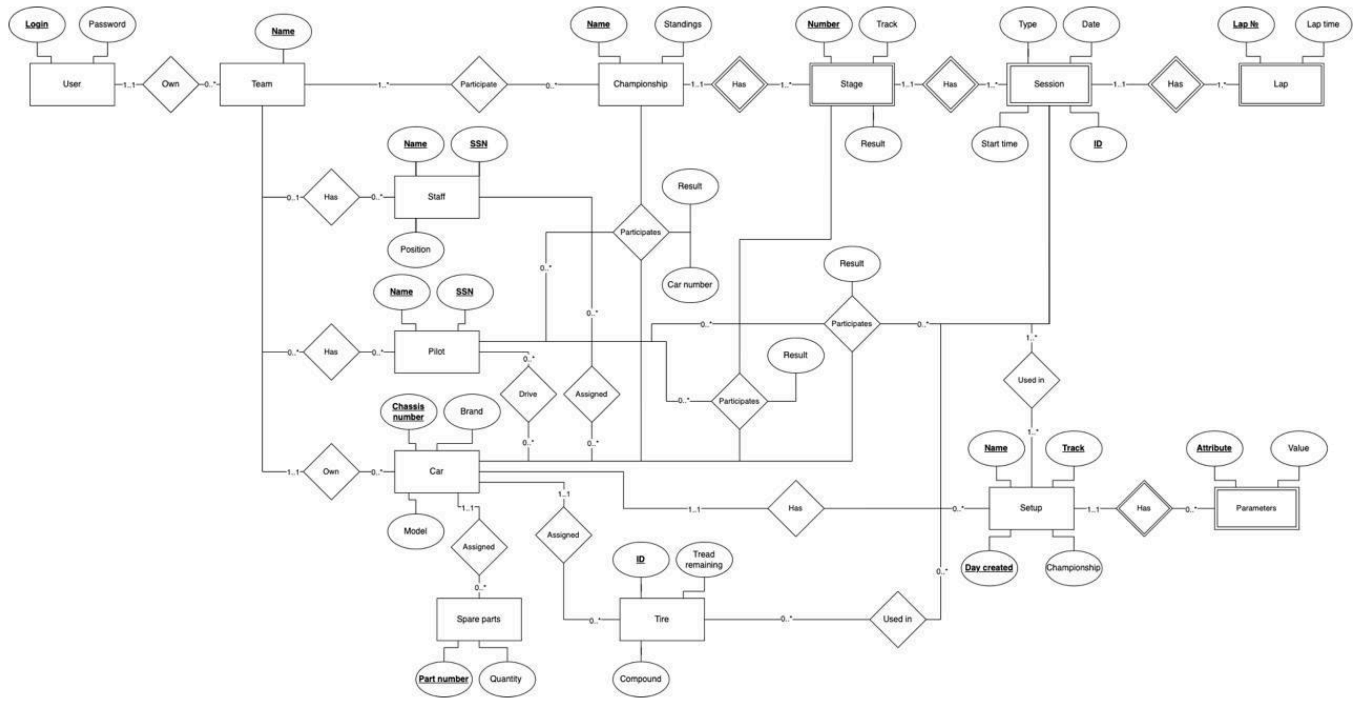
Users must be able to look for an amount of tires with at least x tread remaining attached to a specific car. This use case involves analyzing data.

2.18 User must be able get all the data inserted

This is a generalization that means that all the data inserted must be available for a user to get. Currently it’s only working with postman, but will be implemented to a front end later.

3 Logical Design

3.1 Entity-Relationship Diagram



3.2 Relational Schema

Users(***user_id***, username, email, first_name, last_name, password, created_at)

Teams(***team_id***, user_id (FK→Users.user_id), team_name, created_at)

Cars(***chassis_number***, make, model, team_id (FK→Teams.team_id), created_at)

Parts(***part_id***, part_name, quantity, chassis_number (FK→Cars.chassis_number), created_at)

Championships(***championship_id***, team_id (FK→Teams.team_id), championship_name, team_standings, created_at)

Stages(***stage_id***, stage_number, championship_id (FK→Championships.championship_id), track, start_date, end_date, created_at)

Sessions(***session_id***, stage_id (FK→Stages.stage_id), type, session_date, start_time, weather, temperature, humidity, created_at)

Laps(***lap_number***, ***session_id*** (FK→Sessions.session_id), lap_time, created_at)

Tires(***tire_id***, tread_remaining, compound, chassis_number (FK→Cars.chassis_number), created_at)

4 Use-Case SQL Statements

4.1 User Must Be Able to Create an Account

Description: Creates a new user account by inserting the user's details Into the users table.

SQL Statement: INSERT INTO users (username, email, first_name, last_name, password) VALUES ($1, $2, $3, $4, $5);

Parameters:

$1 - Username (String): The unique username for the account (e.g., 'johndoe').
$2 - Email (String): The user's email address (e.g., 'john@example.com').
$3 - FirstName (String): The user's first name (e.g., 'John').
$4 - LastName (String): The user's last name (e.g., 'Doe').
$5 - Password (String): The user's password (hashed for security).

4.2 User Must Be Able to Create a Team

Description: Creates a new team by inserting the team's details Into the teams table.

SQL Statement: INSERT INTO teams (user_id, team_name) VALUES ($1, $2);

Parameters:
$1 - User_ID (Int): The identifier of the user creating the team (e.g., 53).
$2 - Team_name (String): The name of the team (e.g., 'MyTeam').

4.3 User Must Be Able to Create, Update, and Delete a Championship for a Specific Team

Description: Manages championships by allowing users to create, update, or delete championships associated with a specific team.

Create Championship

SQL Statement: INSERT INTO championships (team_id, championship_name, team_standings) VALUES ($1, $2, $3) RETURNING championship_id;

Parameters:

$1 - Team_ID (Int): The identifier of the team (e.g., 4).
$2 - Championship_Name (String): The name of the championship (e.g., 'Piston Cup').
$3 - Team_Standings (Int): The team's standings in the championship (e.g., 1).

Update Championship

SQL Statement: UPDATE championships SET championship_name = $1, team_standings = $2 WHERE championship_id = $3;

Parameters:

$1 - Championship_Name (String): The updated name of the championship (e.g., 'Premier League').
$2 - Team_Standings (Int): The updated standings of the team (e.g., 8).
$3 - Championship_ID (Int): The identifier of the championship to update (e.g., 1).

Delete Championship

SQL Statement: DELETE FROM championships WHERE championship_id = $1;

Parameters:

$1 - Championship_ID (Int): The identifier of the championship to delete (e.g., 83).

4.4 User Must Be Able to Create, Update, and Delete a Stage for a Specific Championship

Description: Manages stages within a championship by allowing users to create, update, or delete stages associated with a specific championship.

Create Stage

SQL Statement: INSERT INTO stages (stage_number, championship_id, track, start_date, end_date) VALUES ($1, $2, $3, $4, $5) RETURNING stage_id;

Parameters:

$1 - Stage_Number (Int): The number of the stage (e.g., 1).
$2 - Championship_ID (Int): The identifier of the championship (e.g., 93).
$3 - Track (String): The name of the track (e.g., 'Motorsport Arena Oschersleben').
$4 - Start_Date (Date): The start date of the stage (e.g., '2024-05-20').
$5 - End_Date (Date): The end date of the stage (e.g., '2024-05-22').

Update Stage

SQL Statement: UPDATE stages SET stage_number = $1, track = $2, start_date = $3, end_date = $4 WHERE stage_id = $5;

Parameters:

$1 - Stage_Number (Int): The updated stage number (e.g., 2).
$2 - Track (String): The updated track name (e.g., 'Silverstone International Circuit').
$3 - Start_Date (Date): The updated start date (e.g., '2024-06-10').

$4 - End_Date (Date): The updated end date (e.g., '2024-06-12').
$5 - Stage_ID (Int): The identifier of the stage to update (e.g., 394).

Delete Stage

SQL Statement: DELETE FROM stages WHERE stage_id = $1;

Parameters:

$1 - Stage_ID (Int): The identifier of the stage to delete (e.g., 66).

4.5 User Must Be Able to Create, Update, and Delete a Session for a Specific Stage

Description: Manages sessions within a stage by allowing users to create, update, or delete sessions associated with a specific stage.

Create Session

SQL Statement: INSERT INTO sessions (stage_id, type, session_date, start_time, weather, temperature, humidity) VALUES ($1, $2, $3, $4, $5, $6, $7) RETURNING session_id;

Parameters:

$1 - Stage_ID (Int): The identifier of the stage (e.g., 4989).
$2 - Type (String): The type of session (e.g., 'Practice', 'Qualifying', 'Race').
$3 - Session_Date (Date): The date of the session (e.g., '2024-06-11').
$4 - Start_Time (Time): The start time of the session (e.g., '14:00:00').
$5 - Weather (String): The weather conditions during the session (e.g., 'Clear').
$6 - Temperature (FLOAT): The temperature in degrees Celsius (e.g., 25.5).
$7 - Humidity (FLOAT): The humidity percentage (e.g., 60.0).

Update Session

SQL Statement: UPDATE sessions SET type = $1, session_date = $2, start_time = $3, weather = $4, temperature = $5, humidity = $6 WHERE session_id = $7;

Parameters:

$1 - Type (String): The updated type of session (e.g., 'Race').
$2 - Session_Date (Date): The updated date of the session (e.g., '2024-06-12').
$3 - Start_Time (Time): The updated start time (e.g., '15:00:00').
$4 - Weather (String): The updated weather conditions (e.g., 'Cloudy').
$5 - Temperature (FLOAT): The updated temperature (e.g., 22.0).
$6 - Humidity (FLOAT): The updated humidity percentage (e.g., 55.0).
$7 - Session_ID (Int): The identifier of the session to update (e.g., 56).

Delete Session

SQL Statement: DELETE FROM sessions WHERE session_id = $1;

Parameters:

$1 - Session_ID (Int): The identifier of the session to delete (e.g., 99).

4.6 User Must Be Able to Create, Update, and Delete a Lap for a Specific Session

Description: Manages laps within a session by allowing users to create, update, or delete laps associated with a specific session.

Create Lap

SQL Statement: INSERT INTO laps (lap_number, session_id, lap_time) VALUES ($1, $2, $3);

Parameters:

$1 - Lap_Number (Int): The number of the lap (e.g., 1).
$2 - Session_ID (Int): The identifier of the session (e.g., 11).
$3 - Lap_Time (TIME): The time taken to complete the lap (e.g., '01:30.500').

Update Lap

SQL Statement: UPDATE laps SET lap_time = $1 WHERE session_id = $2 AND lap_number = $3;

Parameters:

$1 - New_Lap_Time (TIME): The updated lap time (e.g., '01:29.750').
$2 - Session_ID (Int): The identifier of the session (e.g., 339).
$3 - Lap_Number (Int): The number of the lap to update (e.g., 1).

Delete Lap

SQL Statement: DELETE FROM laps WHERE session_id = $1 AND lap_number = $2;

Parameters:

$1 - Session_ID (Int): The identifier of the session (e.g., 9393).
$2 - Lap_Number (Int): The number of the lap to delete (e.g., 1).

4.7 User Must Be Able to Create, Update, and Delete a Car for a Specific Team

Description: Manages cars within a team by allowing users to create, update, or delete cars associated with a specific team.

Create Car

SQL Statement: INSERT INTO cars (chassis_number, make, model, team_id) VALUES ($1, $2, $3, $4);

Parameters:

$1 - Chassis_Number (String): The unique chassis number of the car (e.g., 'CHASSIS123').
$2 - Make (String): The manufacturer of the car (e.g., 'Ferrari').
$3 - Model (String): The model of the car (e.g., '296 GT3).
$4 - Team_ID (Int): The identifier of the team (e.g., 399).

Update Car

SQL Statement: UPDATE cars SET make = $1, model = $2, team_id = $3 WHERE chassis_number = $4;

Parameters:

$1 - Make (String): The updated manufacturer of the car (e.g., 'Lamborghini').
$2 - Model (String): The updated model of the car (e.g., 'Huracán GT3 EVO2').
$3 - Team_ID (Int): The updated identifier of the team (e.g., 345).
$4 - Chassis_Number (String): The chassis number of the car to update (e.g., 'CHASSIS123').

Delete Car

SQL Statement: DELETE FROM cars WHERE chassis_number = $1;

Parameters:

$1 - Chassis_Number (String): The chassis number of the car to delete (e.g., 'CHASSIS123').

4.8 User Must Be Able to Create, Update, and Delete a Part for a Specific Car

Description: Manages parts within a car by allowing users to create, update, or delete parts associated with a specific car.

Create Part

SQL Statement: INSERT INTO parts (part_id, part_name, quantity, chassis_number) VALUES ($1, $2, $3, $4);

Parameters:

$1 - Part_ID (String): The unique identifier of the part (e.g., 'PART001').
$2 - Part_Name (String): The name of the part (e.g., 'Brake Pads').
$3 - Quantity (Int): The quantity of the part (e.g., 4).
$4 - Chassis_Number (String): The chassis number of the car (e.g., 'CHASSIS123').

Update Part

SQL Statement: UPDATE parts SET part_name = $1, quantity = $2, chassis_number = $3 WHERE part_id = $4;

Parameters:

$1 - Part_Name (String): The updated name of the part (e.g., 'Brake Calipers').
$2 - Quantity (Int): The updated quantity of the part (e.g., 6).
$3 - Chassis_Number (String): The updated chassis number of the car (e.g., 'CHASSIS124').
$4 - Part_ID (String): The identifier of the part to update (e.g., 'PART001').

Delete Part

SQL Statement: DELETE FROM parts WHERE part_id = $1;

Parameters:

$1 - Part_ID (String): The identifier of the part to delete (e.g., 'PART001').

4.9: User Must Be Able to Create, Update, and Delete a Tire for a Specific Car

Description: Manages tires within a car by allowing users to create, update, or delete tires associated with a specific car.

Create Tire

SQL Statement: INSERT INTO tires (tire_id, tread_remaining, compound, chassis_number) VALUES ($1, $2, $3, $4);

Parameters:

$1 - Tire_ID (String): The unique identifier of the tire (e.g., 'TIR041').
$2 - Tread_Remaining (FLOAT): The remaining tread depth of the tire (e.g., 2.8).
$3 - Compound (String): The compound type of the tire (e.g., 'dry').
$4 - Chassis_Number (String): The chassis number of the car (e.g., 'CHASSIS123').

Update Tire

SQL Statement: UPDATE tires SET tread_remaining = $1, compound = $2, chassis_number = $3 WHERE tire_id = $4;

Parameters:

$1 - Tread_Remaining (FLOAT): The updated tread depth (e.g., 2.6).
$2 - Compound (String): The updated compound type (e.g., 'wet').
$3 - Chassis_Number (String): The updated chassis number of the car (e.g., 'CHASSIS124').

$4 - Tire_ID (String): The identifier of the tire to update (e.g., 'TIR041').

Delete Tire

SQL Statement: DELETE FROM tires WHERE tire_id = $1;

Parameters:

$1 - Tire_ID (String): The identifier of the tire to delete (e.g., 'TIR041').

4.10 User Must Be Able to Look for the Best Lap in a Session

Description: Retrieves the fastest lap details for a specific session, including lap number, lap time, and lap time in seconds.

SQL Statement: SELECT lap_number, lap_time, EXTRACT(EPOCH FROM lap_time) AS lap_seconds FROM laps WHERE session_id = $1 ORDER BY lap_time ASC LIMIT 1;

Parameters:

$1 - Session_ID (Int): The identifier of the session (e.g., 101).

4.11 User Must Be Able to Look for an Average Lap in a Session

Description: Calculates the average lap time in seconds for all laps within a specific session.

SQL Statement: SELECT AVG(EXTRACT(EPOCH FROM lap_time)) FROM laps WHERE session_id = $1;

Parameters:

$1 - Session_ID (Int): The identifier of the session (e.g., 101).

4.12 User Must Be Able to Look for the Number of Parts Assigned to a Car

Description: Retrieves the total number of parts assigned to a specific car based on its chassis number.

SQL Statement: SELECT COUNT(*) FROM parts WHERE chassis_number = $1;

Parameters:

$1 - Chassis_Number (String): The chassis number of the car (e.g., 'CHASSIS123').

4.13 User Must Be Able to Look for the Number of Tires Assigned to a Car

Description: Retrieves the total number of tires assigned to a specific car based on its chassis number.

SQL Statement: SELECT COUNT(*) FROM tires WHERE chassis_number = $1;

Parameters:

$1 - Chassis_Number (String): The chassis number of the car (e.g., 'CHASSIS123').

4.16 User Must Be Able to Look for the Average Tire Wear of a Specific Car

Description: Calculates the average tread remaining across all tires assigned to a specific car.

SQL Statement: SELECT AVG(tread_remaining) FROM tires WHERE chassis_number = $1;

Parameters:

$1 - Chassis_Number (String): The chassis number of the car (e.g., 'CHASSIS123').

4.17 User Must Be Able to Look for the Number of Tires with Tread Remaining Higher Than X and Compound Y

Description: Retrieves the count of tires for a specific car that have a tread remaining greater than or equal to a specified value and match a particular compound type.

SQL Statement: SELECT COUNT(*) FROM tires WHERE chassis_number = $1 AND tread_remaining >= $2 AND compound = $3;

Parameters:

$1 - Chassis_Number (String): The chassis number of the car (e.g., 'CHASSIS123').
$2 - Tread_Threshold (FLOAT): The minimum tread remaining value (e.g., 2.5).
$3 - Compound (String): The compound type to filter by (e.g., 'dry').

4.18 User Must Be Able to Get All the Data Inserted

Description: Retrieves all related data for a user, including teams, championships, stages, sessions, laps, cars, parts, and tires.

SQL Statements:

Retrieve Teams for a User

SQL Statement: SELECT team_id, user_id, team_name FROM teams WHERE user_id = $1;

Parameters:

$1 - User_ID (Int): The identifier of the user (e.g., 893).

Retrieve Championships for a Team

SQL Statement: SELECT championship_id, team_id, championship_name, team_standings FROM championships WHERE team_id = $1;

Parameters:

$1 - Team_ID (Int): The identifier of the team (e.g., 32435).

Retrieve Stages for a Championship

SQL Statement: SELECT stage_id, stage_number, championship_id, track, start_date, end_date FROM stages WHERE championship_id = $1;

Parameters:

$1 - Championship_ID (Int): The identifier of the championship (e.g., 54546).

Retrieve Sessions for a Stage

SQL Statement: SELECT session_id, stage_id, type, session_date, start_time, weather, temperature, humidity FROM sessions WHERE stage_id = $1;

Parameters:

$1 - Stage_ID (Int): The identifier of the stage (e.g., 4235).
Retrieve Laps for a Session

SQL Statement: SELECT lap_number, lap_time FROM laps WHERE session_id = $1;

Parameters:

$1 - Session_ID (Int): The identifier of the session (e.g., 101).

Retrieve Cars for a Team

SQL Statement: SELECT chassis_number, make, model, team_id FROM cars WHERE team_id = $1;

Parameters:

$1 - Team_ID (String): The identifier of the team (e.g., 4356).

Retrieve Parts for a Car

SQL Statement: SELECT part_id, part_name, quantity, chassis_number FROM parts WHERE chassis_number = $1;

Parameters:

$1 - Chassis_Number (String): The chassis number of the car (e.g., 'CHASSIS123').

Retrieve Tires for a Car

SQL Statement: SELECT tire_id, tread_remaining, compound, chassis_number FROM tires WHERE chassis_number = $1;

Parameters:

$1 - Chassis_Number (String): The chassis number of the car (e.g., 'CHASSIS123').

5 Applications

My application is a web service that allows users to have their data at any place with the only requirement being a stable Internet connection. It consists of a frontend user Interface, backend server that handles the main logic and a database that stores the data.

6 Conclusions

During the development time, I had time to introduce the bulk of the backend features I wanted to create. On the frontend there are much less things ready, but still there are some successes and some base was successfully created.
I've really learned a lot while working on this project. I learned a lot about the Go programming language and how full-fledged web applications work, and most importantly I learned how to develop them, as well as how to design interactions between a large number of databases.
From the plans for the future, I will definitely finish the stated but not realized functions, completely finish the development of the user interface, as well as add additional degrees of protection and verification to database queries. I would also like to expand the analysis functionality even more and of course agree with a real racing team to test the use of my product to get real feedback.