SOEN 487 Microservices Project
MY Game Schedules
Application / Microservices Documentation

Asif But (ID: 24532139)
Andrew Sidhom (ID: 40011469)
Corey Cohen (ID: 40006084)

Project / Application Description

The My Game Schedules project is a personalized dashboard tailored to needs of soccer enthusiasts. The goal of the project is to provide a personalized experience by delivering game schedules of the selected teams of the registered members.

Registered users will be able to view game schedules of their selected soccer teams from the English Premier League, as well as the current ranking of these teams. Although this basic version of the application is restricted to only one league and few features, the application can be easily extended to include other competitions as well as customized features such as an email notification service about the upcoming games that a user is interested in.

User Stories

The following are small set of user stories from which some high level components of the application will be derived:

- A user can create an account on My Game Schedule.
- A registered user can log in and see on their homepage all the scheduled games of teams that they are personally following which are to be played over the following two weeks.
- A user who hasn't logged in in a while and sees outdated games can click a "Refresh" button which causes the application to refresh its data and display the most up-to-date information.
- A logged in user can at any time add or remove teams from their personalized list of teams that they follow, as they please.

Microservices Description

- Authentication Service: Authenticating users and services.
- Retrieval Service: Periodically talks to external APIs to retrieve game schedules and then talks to the Database Management service to make it update the database.
- Database Management: Manages a local database containing info about teams and their current rankings, about all games to be played soon, and about each of the app's users and their favorite teams.
- Front-End Service: Provides standard and basic user interface (UI) and is responsible for interacting with the rest of the services in servicing requests from clients.

API

Method	URI	Description
GET	http://mygameschedules/welcome	Homepage / Dashboard which displays a personalized schedule of games to the logged in user
POST	http://mygameschedules/login	Logs the user in
POST	http://mygameschedules/register	Registers new users
POST	http://mygameschedules/myteams/	Adds a new team to the user's personalized list of teams
DELETE	http://mygameschedules/myteams/<team_id>	Removes a team from the user's personalized list of teams

Notes for running the project

- The Authentication microservice needs to be run before the other three services as they will call it to each get a service token when they first run.
- In the DB Management microservice, if not using the included sqlite database file, the main() function in the db_init.py module can be executed to create the database schema as well as populating the team table with the English Premier League teams with proper ids as used throughout the app.

Retrieval Microservice

This is the microservice responsible for calling an external API (which holds a large amount of up-to-date soccer-related data) and extracting from it the data that will be relevant and useful for our app.

The external API used is http://api.football-data.org

The data we extract concerns the English Premier League. What we are interested in is current league rankings for all 20 teams participating in the competition and up-to-date game schedules for all these teams over a period of two weeks starting from the day the call is made.Thus, the service exposes two endpoints to be used by the Front End:

URL	**Accepted method**	**Function**
/games	PUT	Retrieves game schedules and makes sure they are persisted in our app
/teams	PUT	Retrieve Steam rankings and makes sure they are persisted in our app

In each endpoint, before any of the main code is executed, we first make sure that the endpoint is called with a valid token. This is to ensure that no calls from outside the app are made to these endpoints. They can only be called from within the app (by the Front End).

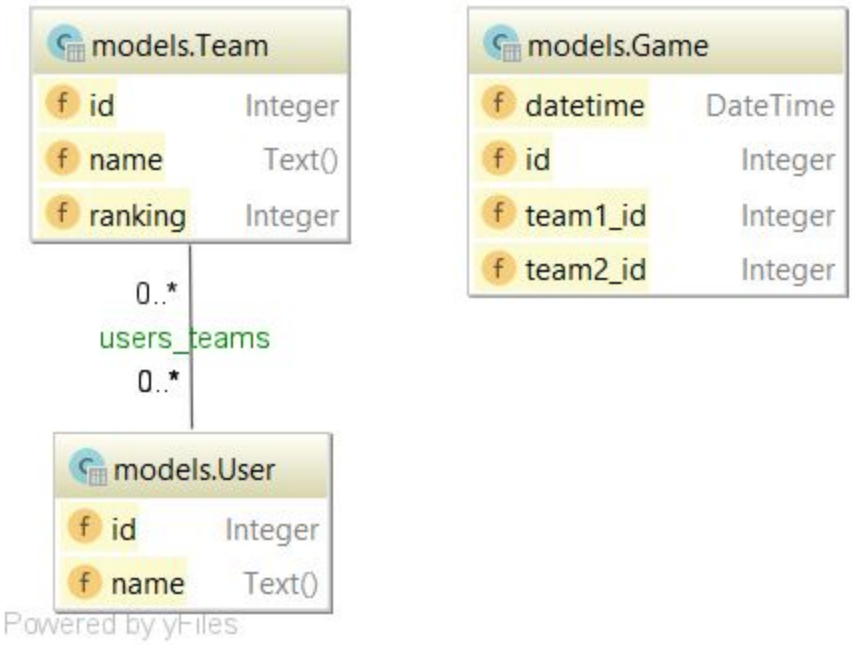
Once that token verification is over, we proceed to request the data from the external API. We receive JSON back. We then filter that JSON to keep only the information we are interested in.

Once we have the filtered data in JSON format, we need to persist it in our app. That is the job of another microservice: the Database Management microservice. But the Retrieval microservice needs to ensure that data has been persisted there before returning a response to the Front End that the whole operation has been successful. Soit makes a call to the endpoints exposed (for internal app use) by the DB Management service to allow updates to the database. The call is made with a token, and with the JSON as data. The response we get from the DB Management service, indicating success or failure, is forwarded back to the Front End.

Dates are retrieved from the external API in UTC time zone and converted by this service into local time before sending them to the database.

Database Management Microservice

This microservice defines models (in models.py) using SQLAlchemy for object-relational mapping. The following are the models defined.



They correspond to 4 tables in the database: *user*, *team*, *users_teams* (which is a many-to-many relationship table between user and team, keeping track of which users are following which teams) and *game*.

The endpoints of this microservice (defined in app.py) expose many possible CRUD operations on these models (i.e. resources), even though we do not use all of these operations in this app. However, they are there to permit possible expansions of the app.

As in the Retrieval Microservice, each endpoint can only be accessed from within the app with a valid token. For example, the endpoint to add a user to the database is called by the authentication microservice when a user is registered in the app (to persist their user is and user name).

The most notable logic that is in this microservice and is heavily used by our app is found in the following endpoint:

URL	Accepted method	Function
/user/<user_id>/game	GET	Returns a JSON of all scheduled games to be played by teams followed by the user

This endpoint provides the pivotal functionality of our app: to display to a user, on their homepage, a table of all games to be played soon by any team that they are following. This endpoint is called by the Front End every time the logged in user's homepage needs to be displayed.

To accomplish that, we use two SQLAlchemy joins as well as a subquery. All of that is translated to one SQL query issued to the database, which means it is optimized by the SQL engine.

The subqueries purpose is to list all teams that the user is following (let us call this result "my_teams"). Then, the main query selects all games from the game table where team1_id is in my_teams or team2_id is in my_teams. But that result set would refer to teams by id. We want to return to the Front End team names and their rankings instead. That is why the joins are used with the *team* table.

Another important endpoint is the following:

URL	Accepted method	Function
/games	PUT	Updates the games in the database with the data supplied (in JSON)

This endpoint is called by the Retrieval microservice (see above) to persist the data it has just retrieved in the app's main database. This is done as follows: First, games that are still in the database but have already been played (having a date and time that precedes the current moment in time) are deleted. Second, the game table is updated with any games that have a date and time following the "maximum" date and time currently existing in the table. In other words, games that are scheduled after the ones already in the table are simply added on.

Authentication Service

This service provides the user registration and login mechanism. It provides all of it's functionality primarily through 4 use cases.

In order to provide secure access to the application and all of its features, a valid login is required. The application will use basic authentication, a simple authentication scheme assuming the application will service all requests over secure connection using https. Additionally, the application will implement OAuth2 protocols standard to secure all interactions, whether they are users-to-service or service-to-service. Json Web Token (JWT), has been selected to represent tokens.

The service also connects to its own database storing primarily user information to support its use cases. Following points briefly describe the use cases, protocols behind their implementation and the end points serviced by this service:

Authentication Service Use Cases:

- A user can register on the games schedule application by creating an account.
- A user can login to games schedule application by providing the user name and password created during account creation process.
- User will obtain a Json Web Token (JWT) token upon successful login. This token is required to access the different features of the application serviced by other microservices.
- Authentication service will also issue JWT tokens to all microservices for secure service to service interactions.

User Registration:

Method	Url	Function
POST	http://mygameschedules/register	Registers new users

One of the key feature of the application serviced by authentication service is user registration. The feature relies on existence of the user model as an abstract representation of users. The

user model consists of username, id, admin and password attributes. Passwords in the database will be stored as hashes and not plain text, while the admin field is a boolean value that makes a distinction between normal user and administrator of the site with extra privileges. The following steps describe the control flow of the registration use case:

- User name and password are retrieved from front-end service.
- Password hash is computed and stored in the database along with other user information.
- Upon successful registration, success message will be returned to front-end service.

User Login:

Method	Url	Function
POST	/login	Logs the user in

The login feature was implemented using basic authentication assuming the application will run over https. Unauthenticated and unauthorized users are redirected to the endpoint above. User supplied credentials are verified and additionally a user token is generated and sent back to the front-end service which stores the user token in the flask session object.

Service to Service Authentication:

Method	Url	Function
POST	/oauth/token	Issue tokens to services.

The final use case serviced by the authentication is service to service authentication. Securing service to service interaction is a key security feature in implementing web services and OAuth2 protocol was the preferred choice for this functionality. Client Code Grant (CCG) flow of the OAuth2 protocol was implemented using Json Web Tokens (JWT) as representation for issuing tokens to other services. Services were identified by their secret keys stored in configuration module of authentication service. Any service verified by its secret key is issued a token that they can present to other services.

The final piece of service to service authentication was token verification and validation. The authentication service secret key was shared and stored in configuration of all interacting and

independent services. With the secret key of authentication service, every other service is able to identify and ensure token authenticity and validity.

Front-End Service

This microservice enables the users to interact with the other relevant microservices present in the program by providing them with an interface to do so.
It enables users to have the desired functionality the program was designed to allow them by taking advantage of endpoints created in the Auth, Retrieval and Database Management microservices, specifically the ones as follows:

Register

Method	Url	Function
POST	http://mygameschedules/register	Registers new users

When a user makes a GET request then the template is rendered and returned to them. They can then create and enter a new username and password. This data is pulled from the template and then sent over to the Auth service which issues back a response of registration success or failure. In case of success, the user is now able to attempt to login.

Login

Method	Url	Function
POST	/login	Logs the user in

If a GET request is made for login a template is rendered and returned so they can fill in their credentials. Once they do so and submit a POST request, the credentials they entered are pulled from the form and sent to the Auth service where they are validated and a user token is returned. Next, they are redirected to the homepage where they can view their games.

Homepage

Method	Url	Function
GET	http://mygameschedules/welcome	Returns users list of

		personalized games and rankings

Once the homepage is rendered, it displays the user's list of games for teams (if any) that have been chosen, by making a request to the DB Management service. The associated template makes use of Flask's Jinja to iterate through and display the received Json. From here, users can choose to add or remove teams for their personalized list with the help of the included navigation bar that lies within the HTML template.

Add team

Method	Url	Function
POST	http://mygameschedules/myteams/	Adds a new team to the user's personalized list of teams

Should a user wish to add a new team, they can select their desired team from a dropdown menu and submit a POST request. Here user token validation is essential and is retrieved from the session. Once validated, the POST can be executed. The chosen value is taken from the dropdown menu located in the template form and is then passed through a switch statement inside the endpoint. The index is matched with its corresponding team id and sent to the Database Management service where it is then added to the users list of teams.

Delete team

Method	Url	Function
DELETE	http://mygameschedules/myteams/<team_id>	Removes a team from the user's personalized list of teams

Here the same logic applies as the add team, however a delete team template is rendered instead and the selected team id is sent to the Database Management service to be deleted from the users personal list.

Logout

Method	Url	Function
DELETE	http://mygameschedules/logout	Logs out user

Finally, when the user wishes to log out, they are “popped” from the session and directed back to the login template.