
Submission 2

Rhys Harvey

February 18, 2024

Contents

1	Tech Report - REST API	3
2	Class Diagram - Match Page	5
2.1	Class Diagram	5
2.2	JDL	6
3	Kanban Feature Cards	8
3.1	Linked Issues	10
4	Timesheets	12

1 Tech Report - REST API

APIs (Application Programming Interface) are a way for two computers to communicate. This is done by one device sending a request and the other receiving the request, acting out the request, and sending back a response. This is similar to a client server model. Most APIs are RESTful which means they follow a set of rules known as Representational State Transfer. A RESTful API will organise data into URIs (Uniform Resource Identifiers) to differentiate between different sets of data. Clients make requests to the URI end point over HTTP. This architecture is stateless, meaning that all the data needed to identify the client and the session's current state is contained within the request, so the server does not need to store any data about the client's session.

A Request consists of:

- Start line ⇒ Contains the Request Method, URI and HTTP protocol
- Headers ⇒ Contains Metadata about the request, such as the format of the response, or authorisation tokens
- Body ⇒ Contains a custom data payload, usually in JSON format (optional)

Common Request Methods:

- GET ⇒ Request to retrieve data from the endpoint, no data is changed
- POST ⇒ Request to create a new object at the endpoint, e.g. adding a new record to database. Repeating a POST request will add multiple new records
- PUT ⇒ Request to create a resource, or update it if it exists, similar to POST but is idempotent, meaning that multiple PUT requests will always leave the resource in the same state
- PATCH ⇒ Request to update a resource, is more efficient at updating than PUT as only the changes need to be sent, not the whole resource
- DELETE ⇒ Request to delete a resource from the endpoint

A Response consists of:

- Status Code ⇒ Tell the client what happened to their request
- Headers ⇒ Contain information about the server
- Body ⇒ Contains data payload, usually in JSON format

Status codes starting with 2** mean that the request was successful, starting with 4** means that there was a problem with the client's request, and 5** means that the server failed to complete the request.

Example

Here is an example of a POST Request made to a locally hosted API to add a new "Match" and the Response received, made using Node.js. In this example the server saves the data from the request into a text file that can then be requested back later with a GET request. The response given by the server confirms that a new "Match" has successfully been added.

Request

```
> POST /match/1 HTTP/1.1
> Host: localhost:8080
> Content-Type: application/json
> User-Agent: insomnia/8.6.1
> Accept: */*
> Content-Length: 27
```

```
| {
|   "matchData": "Match 1"
| }
```

Response

```
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Content-Type: application/json; charset=utf-8
< Content-Length: 36
< ETag: W/"24-0DiQQ5KGE9g7Jz+mC6BTsDAOxs"
< Date: Sun, 11 Feb 2024 13:34:29 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
```

```
| {
|   "match": "Created Match: Match 1"
| }
```

In the given Jhipster template, the REST API is defined in "team02/src/main/java/team/bham/web/rest". Java Annotations are used to define the endpoints. The `@RequestMapping(..)` annotation is added to the class to set the base path for all mappings, and the annotations `@GetMapping(..)`, `@PutMapping(..)`, `@PostMapping(..)`, e.t.c. are used for each method. The methods also have a `@PreAuthorize(..)` annotation that requires the request to include a Authorisation token in the header, this corresponds to an account on the app, and their permissions are checked to see if they are allowed to make the request. Without this header the request is refused.

Example: `/api/admin/users` definition

Here is the Method: `getAllUsers` from within the class `UserResource.java` (with mapping `/api/admin`). It defines the response given when a GET Request is made to `/api/admin/users`.

```
1| @GetMapping("/users")
2| @PreAuthorize("hasAuthority(\"\" + AuthoritiesConstants.ADMIN + "\")")
3| public ResponseEntity<List<AdminUserDTO>> getAllUsers(@org.springdoc.api.annotations.ParameterObject
↪ Pageable pageable) {
4|     log.debug("REST request to get all User for an admin");
5|     if (!onlyContainsAllowedProperties(pageable)) {
6|         return ResponseEntity.badRequest().build();
7|     }
8|
9|     final Page<AdminUserDTO> page = userService.getAllManagedUsers(pageable);
10|    HttpHeaders headers =
↪    PaginationUtil.generatePaginationHttpHeaders(ServletUriComponentsBuilder.fromCurrentRequest(),
↪    page);
11|    return new ResponseEntity<>(page.getContent(), headers, HttpStatus.OK);
12| }
```

- Lines 1-2 are the annotations for the method specifying the U.R.I. (Uniform Resource Identifier) for the request as well as the Request Method this method will handle, and that the request must have admin authority.
- Line 3 is the method signature, which states that the method will return a `ResponseEntity` containing a list of `AdminUserDTOs` (Data Transfer Object), and will take a `Pageable` parameter. This allows the client to specify parameters such as the number of users returned, or how to sort them.
- Line 4 sends a debug message to the log to indicate a request to retrieve all users is being processed.
- Line 5 checks that the `Pageable` parameter only contains allowed properties, if not the request is refused.
- Lines 6-8 generate the response to the request and returns it to the client.

To make a request to `/api/admin/users` we need to include an Authorization token in the Headers of the request. To get this token, we first need to make a POST request to `/api/authenticate` with login details for a user account in the body. The server will respond with an Authorization Token that can be used in future requests to authenticate as this user.

If we give no Authentication Token, the server gives status code: 401 Unauthorised

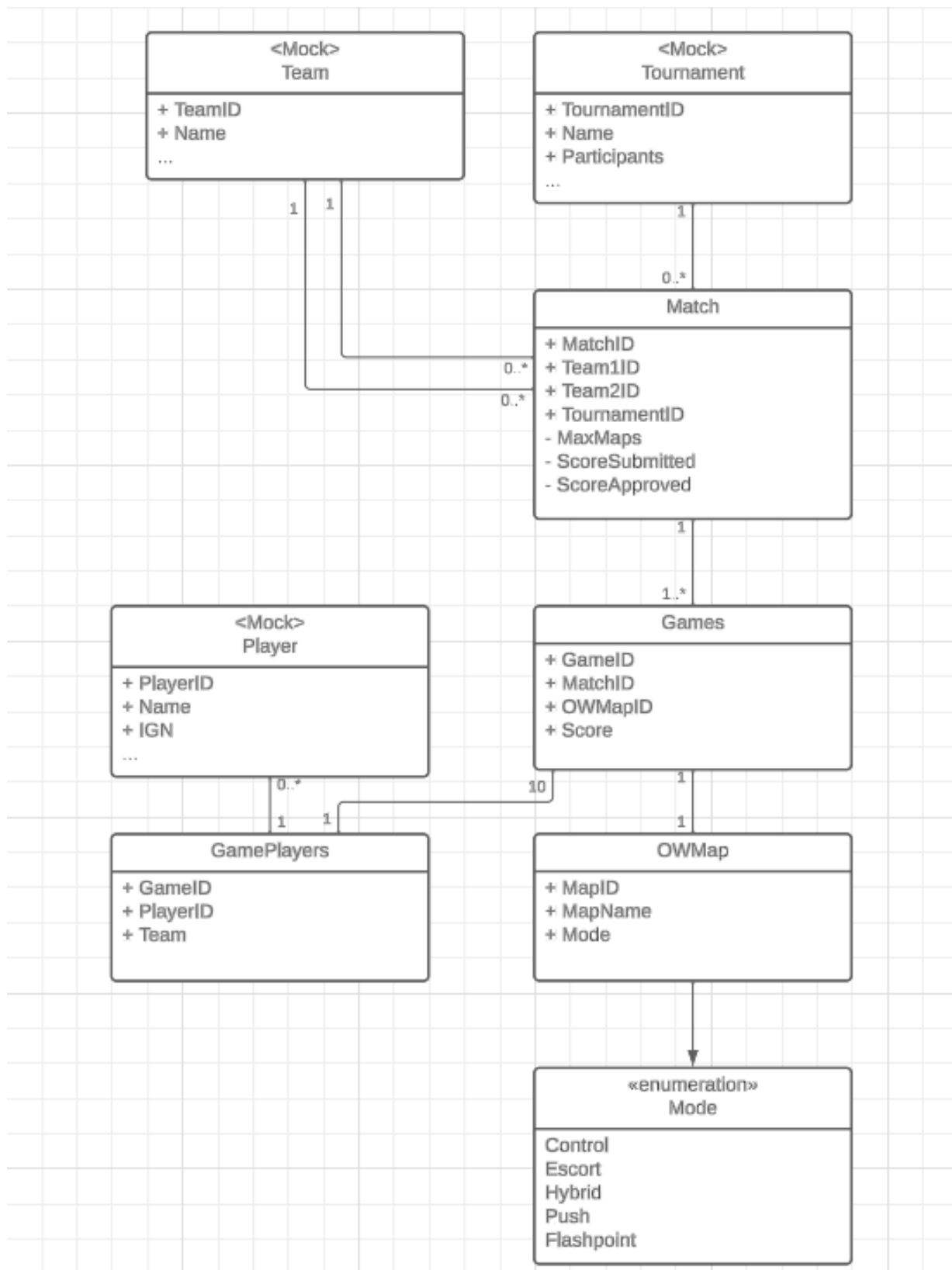
If we give an Authentication Token for a user that doesn't have admin privileges, the server gives status code: 403 Forbidden

Here is an example of a Request to this API with the parameter: `size=1`, and the Response body given:

<u>Request</u>	<u>Response Body</u>
<pre>> GET /api/admin/users?size=1 HTTP/1.1 > Host: localhost:8080 > User-Agent: insomnia/8.6.1 > Authorization: <TOKEN> > Accept: */*</pre>	<pre>{ "id": 1001, "login": "testuser", "firstName": null, "lastName": null, "email": "rxh293@student.bham.ac.uk", "imageUrl": null, "activated": true, "langKey": "en", "createdBy": "anonymousUser", "createdDate": "2024-02-12T13:19:54.305186Z", "lastModifiedBy": "admin", "lastModifiedDate": "2024-02-12T13:21:00.350725Z", "authorities": ["ROLE_USER"] }</pre>

2 Class Diagram - Match Page

2.1 Class Diagram



2.2 JDL

```
//Mock Entity
entity TournamentMock {
  name String
  participants Integer
  //...
}

//Mock Entity
entity TeamMock {
  name String
  //...
}

entity Match {
  maxMaps Integer
  scoreSubmitted Boolean
  scoreApproved Boolean
}

entity Game {
  scoreOne Integer
  scoreTwo Integer
}

//Mock Entity
entity PlayerMock {
  name String
  ign String
  //...
}

entity GamePlayer {
  team Integer
}

enum Mode {
  Control,
  Escort,
  Hybrid,
  Push,
  Flashpoint
}

entity OWMMap {
  name String
  mode Mode
}

relationship OneToMany {
  TournamentMock{matches} to Match{tournament}
}

relationship OneToMany {
  TeamMock{matches} to Match{teamOne}
}

relationship OneToMany {
  TeamMock{matches} to Match{teamTwo}
}

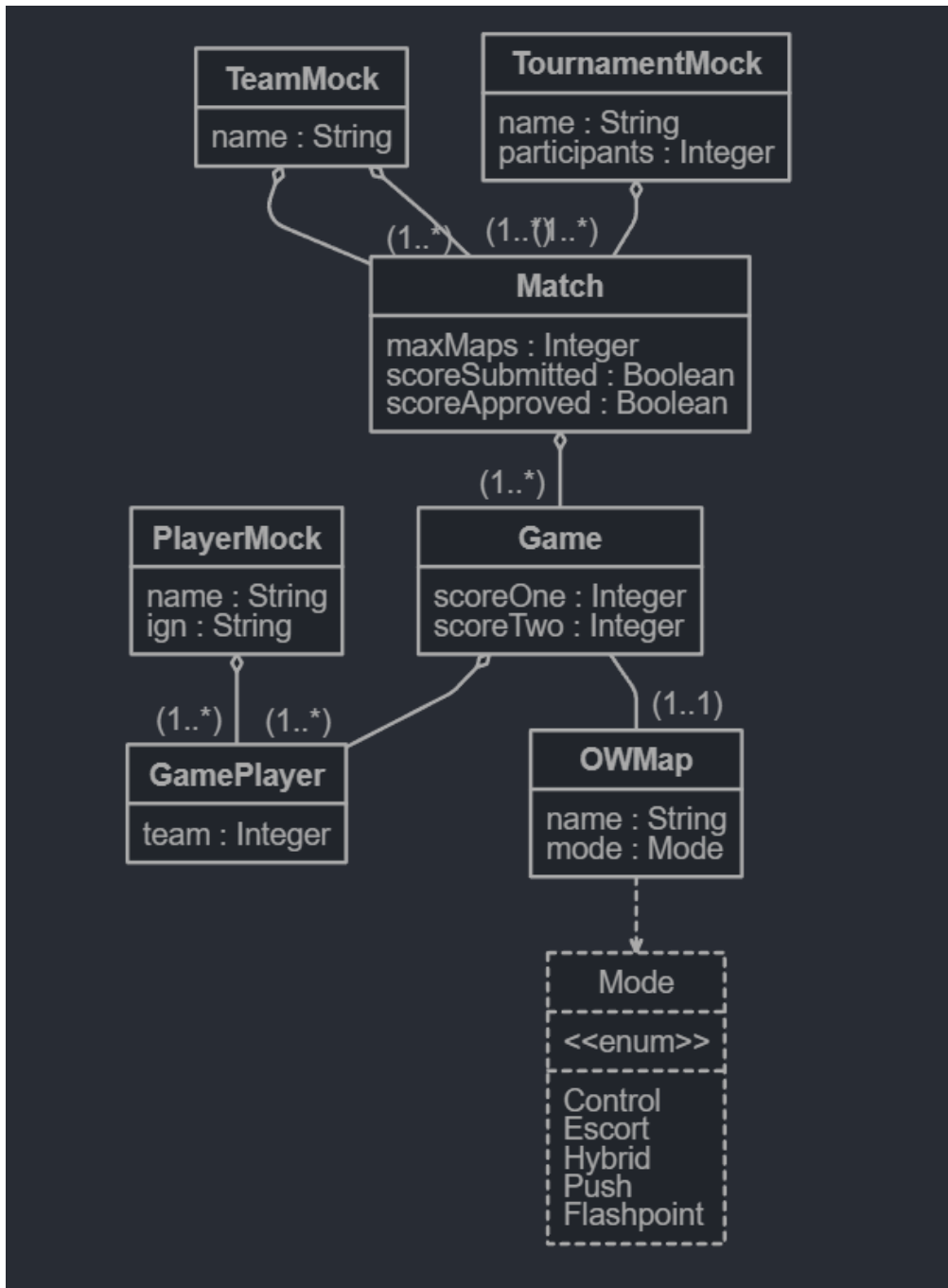
relationship OneToMany {
  Match{games} to Game{match}
}

relationship OneToMany {
  Game{players} to GamePlayer{game}
}

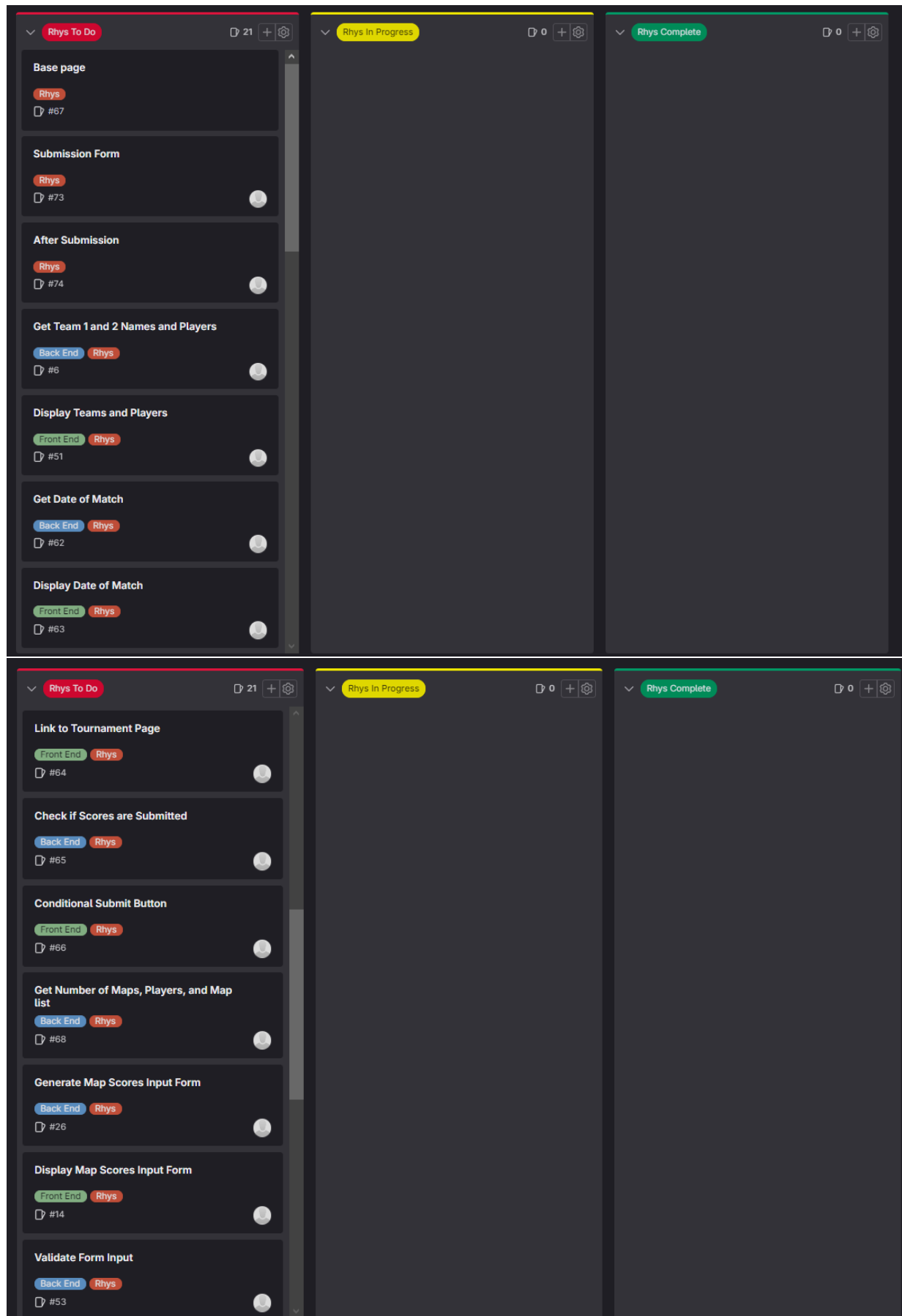
relationship OneToMany {
  PlayerMock{games} to GamePlayer{player}
}

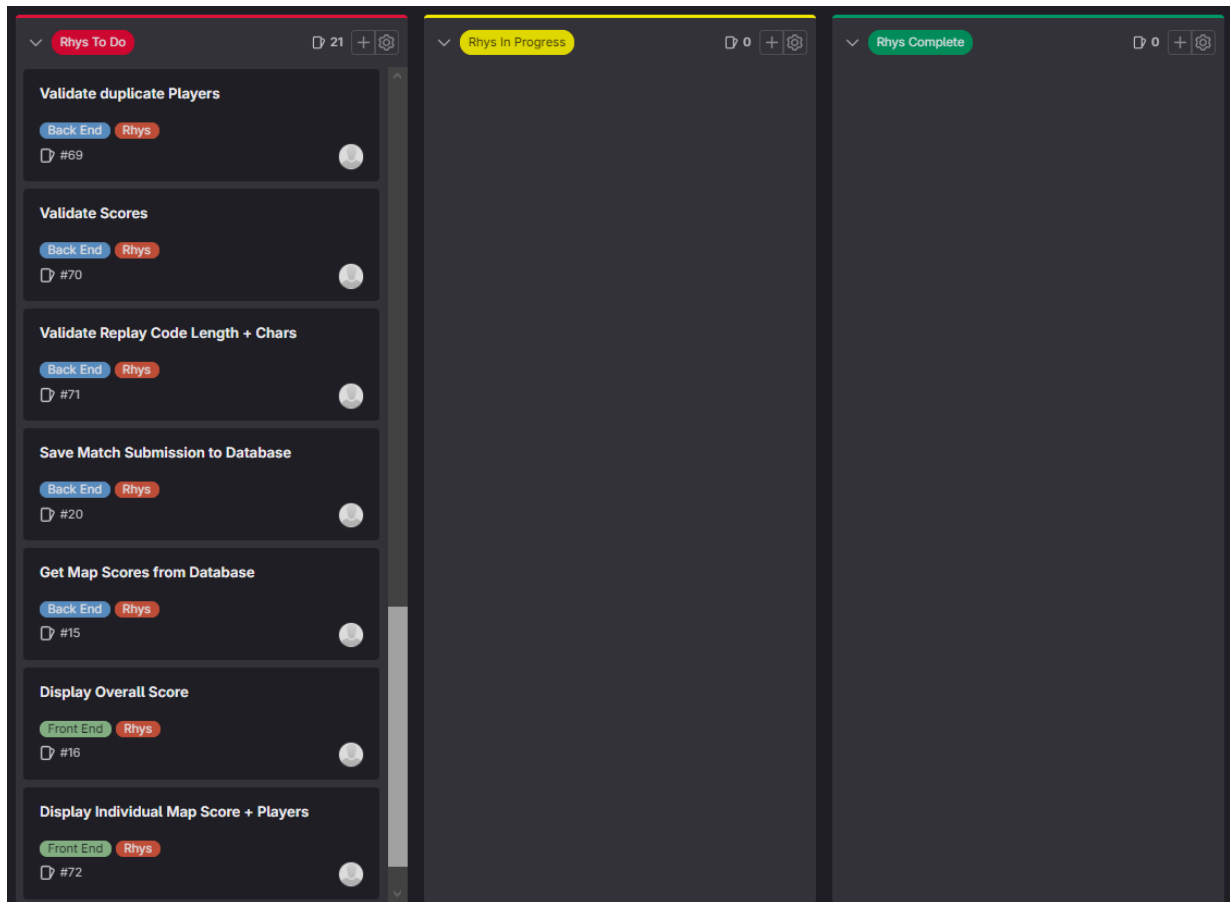
relationship OneToOne {
  Game{map} to OWMMap{games}
}
```

Here is the Class Diagram generated by JDL Studio



3 Kanban Feature Cards





3.1 Linked Issues

Base page

Open Issue created 21 minutes ago by Rhys Harvey

0

0

Create merge request

Drag your designs here or [click to upload](#).

Child items 0

Show labels

Add

^

No child items are currently assigned. Use child items to break down this issue into smaller parts.

Linked items 7

Add

^

Get Team 1 and 2 Names and Players #6

Display Teams and Players #51

Get Date of Match #62

Display Date of Match #63

Link to Tournament Page #64

Check if Scores are Submitted #65

Conditional Submit Button #66

Submission Form

Open Issue created 9 minutes ago by Rhys Harvey

0

0

Create merge request

Drag your designs here or [click to upload](#).

Child items 0

Show labels

Add

^

No child items are currently assigned. Use child items to break down this issue into smaller parts.

Linked items 5

Add

^

Get Number of Maps, Players, and Map list #68

Generate Map Scores Input Form #26

Display Map Scores Input Form #14

Validate Form Input #53

Save Match Submission to Database #20

After Submission

Open Issue created 9 minutes ago by Rhys Harvey

0 0 Create merge request

Drag your designs here or [click to upload](#).

Child items 0 **Show labels** ✓ Add ^

No child items are currently assigned. Use child items to break down this issue into smaller parts.

Linked items 4 Add ^

- Get Map Scores from Database #15 ×
- Display Overall Score #16 ×
- Display Individual Map Score + Players #72 ×
- Conditional Submit Button #66 ×

Validate Form Input

Open Issue created 1 week ago by Rhys Harvey

0 0 Create merge request

Drag your designs here or [click to upload](#).

Child items 0 **Show labels** ✓ Add ^

No child items are currently assigned. Use child items to break down this issue into smaller parts.

Linked items 4 Add ^

- Validate duplicate Players #69 ×
- Validate Scores #70 ×
- Validate Replay Code Length + Chars #71 ×
- Submission Form #73 ×

4 Timesheets

Team sheet Number/ID: Rhys2

Team member name: Rhys

Team representative (secretary): Haiwei

Team meeting sign off date: 08/02/24

Date from: 2/1/2024

Date until: 06/02/24

Task	Date	Start time	End time	Total Hours
Team Meeting - Ranking S1	01/02/24	11:30 AM	1:00 PM	1:30
Create Figma Template	01/02/24	2:30 PM	3:15 PM	0:45
Create Mockup for Match Page with Template	01/02/24	3:15 PM	4:15 PM	1:00
Recreate Template + Mockup With Heading	02/02/24	2:15 PM	4:30 PM	2:15
Team Meeting	05/02/24	1:30 PM	3:00 PM	1:30
Team Meeting - Prepare M1	06/02/24	12:30 PM	1:30 PM	1:00
				0:00

Total Hours **8:00**

Team sheet Number/ID: Rhys3

Team member name: Rhys

Team representative (secretary): Garence

Team meeting sign off date: 16/02/24

Date from: 2/8/2024

Date until: 12/02/24

Task	Date	Start time	End time	Total Hours
Team Meeting with Tutor - talk about S2	08/02/24	11:30 AM	12:30 PM	1:00
Researching + Experimenting with REST API	10/02/24	10:00 AM	11:40 AM	1:40
Start Creating UML Class Diagram	11/02/24	10:00 AM	11:00 AM	1:00
Draft out Research Report on REST API	02/02/24	11:00 AM	1:00 PM	2:00
Finalise Research Report on REST API	12/02/24	5:30 PM	6:15 PM	0:45
Finalise Research Report on REST API	12/02/24	9:00 PM	12:00 AM	3:00
				0:00

Total Hours **9:25**

Team sheet Number/ID: Rhys4

Team member name: Rhys

Team representative (secretary): Talha

Team meeting sign off date: 20/02/24

Date from: 2/13/2024

Date until: 18/02/24

Task	Date	Start time	End time	Total Hours
Team Meeting with Tutor - Discuss S2 / M2	13/02/24	2:00 PM	2:40 PM	0:40
Improve Kanban Cards	16/02/24	11:00 AM	11:30 AM	0:30
Team Meeting - Progress Check + plan for M2	16/02/24	12:30 PM	1:30 PM	1:00
Finalise Class Diagram and write JDL	18/02/24	11:00 AM	12:30 PM	1:30
				0:00
				0:00
				0:00

Total Hours **3:40**