Exercise: Modular Applications

Exercise problem for the "JavaScript Applications" course @ SoftUni.

Working with Remote Data

For the solution of some of the following tasks, you will need to use an up-to-date version of the **local REST service**, provided in the lesson's resources archive. You can read the documentation here.

1. Team Manager

Create a JS application for managing teams.

Libraries

Use the **page** and **lit-html** libraries for routing and templating. Use **`npm install`** to install them, as they are included in **package.json**.

Data

Use the provided local REST service to store **data** and **users**. Note that there are some **special access rules** for this task – anyone can create a record in the requests collection, but only the **team admin** (team owner) can **approve requests** for membership. This check is provided **automatically by the server**, but you must implement **front-end checks**, in order to **display** the correct **controls**.

Users are stored similarly to previous tasks in this course, in the **`/users`** service. In this app, registration requires an extra field **username** , added along with the usual **email** and **password**.

Each user can be **owner** and **member** of multiple teams.

For the catalogue you can **optionally** implement **pagination**, or allow the user to scroll down infinitely while new data is being loaded (**infinite scroller**).

HTML & CSS

As usual you are provided with HTML and CSS. You may change the HTML by adding attributes, adding URL's where needed and so on as you see fit.

Application Requirements

Navbar

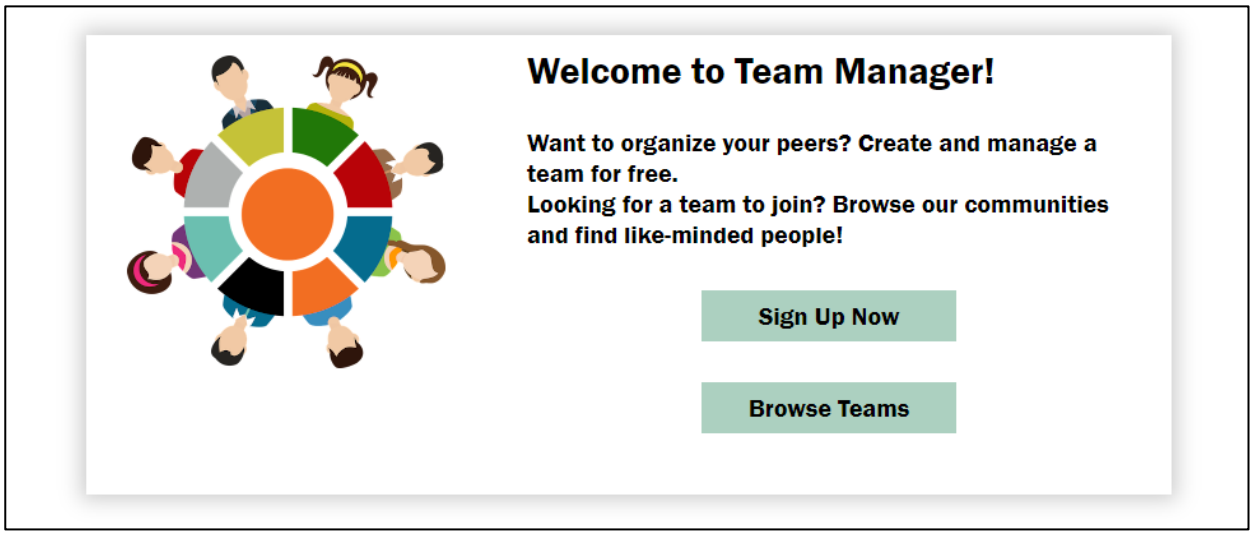
The application has a navigation bar, with **links to other pages**. You need to **implement** the **visibility** of the **buttons** in the header, depending on whether there is a **logged-in user** or a **guest**. This is how the header should look like for a **guest** user:

When there is a **logged-in** user, the header should look like this:

The "**Browse Teams**" button is **always visible**. The "**Team Manager**" is link to **home page**.

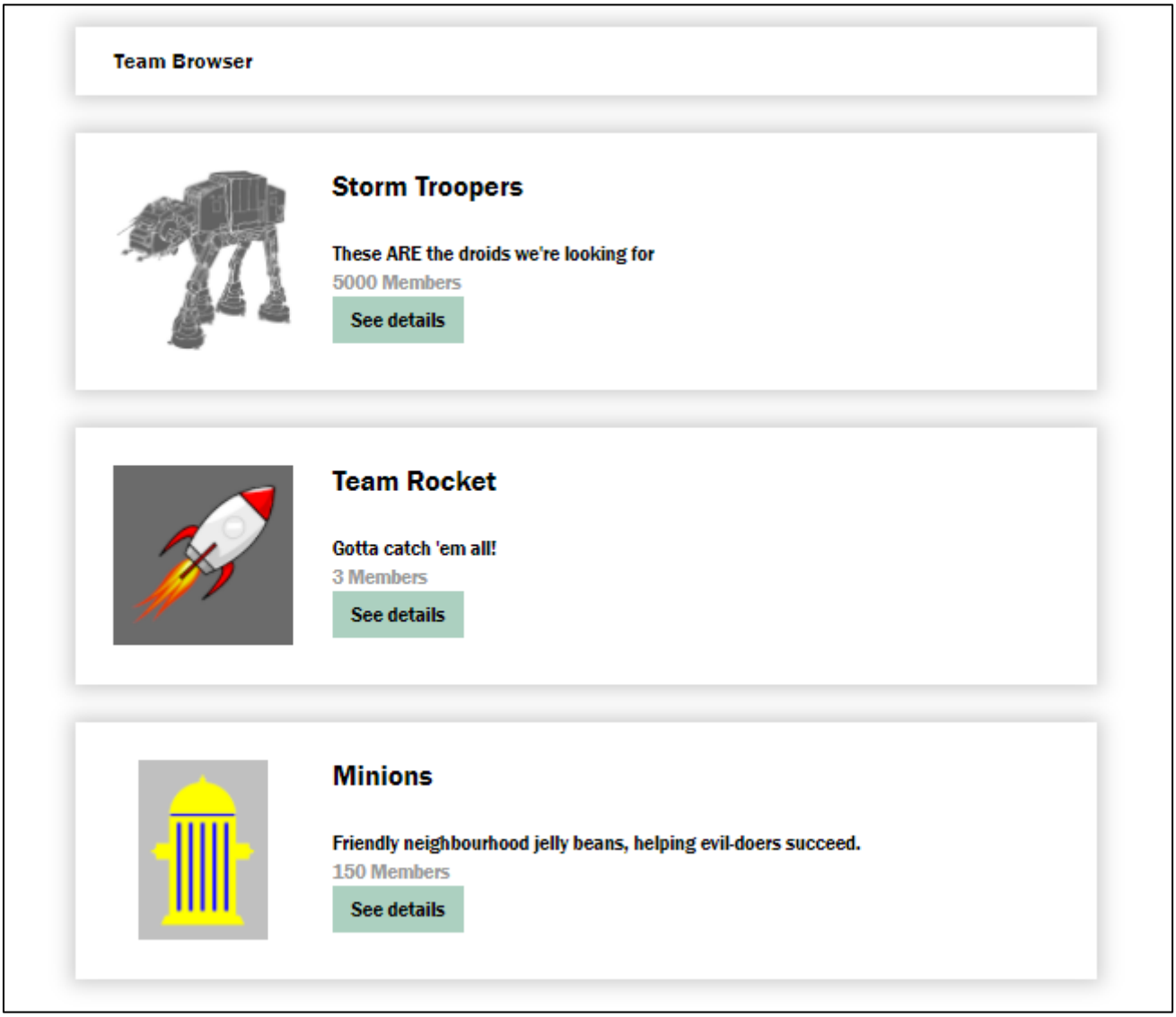
Home

Guest visitors see the "**Sign Up Now**" button, while logged-in users see the "**Browse Teams**" buttons.

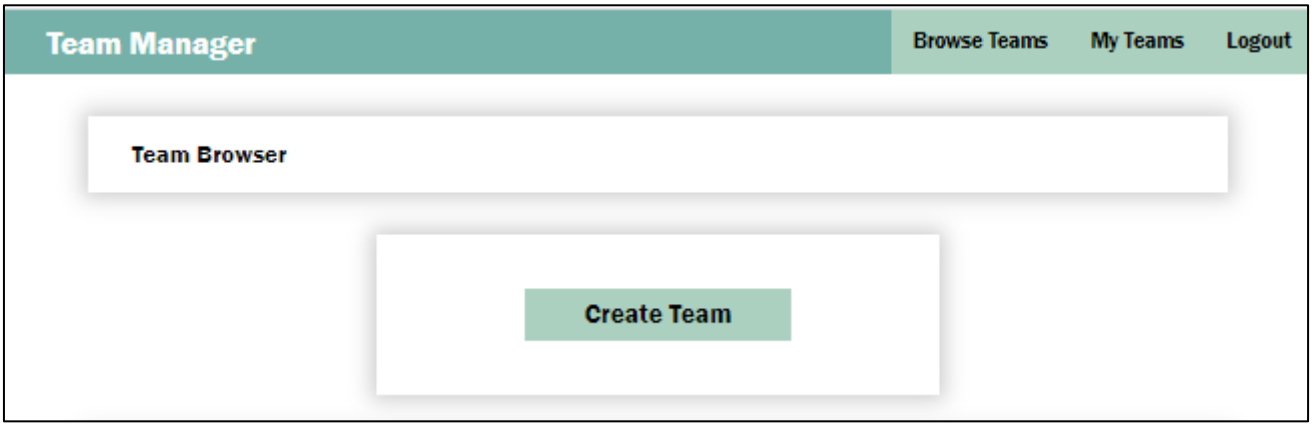


Browse Teams

Guest users can see a list of all teams. The button "Create Team" is not available for guest users.



Logged in users can see the button **"Create Team"**



Use the following endpoints to obtain information from the service:

- List of all teams:

```
GET /data/teams
```

- List of all members:

```
GET /data/members?where=status%3D%22member%22
```

Once you have the list of all members, you need to process the list and **count the members** for every team. Note that the previous request will return all members in all teams. If you want to use **pagination** for the catalog (optional), you need to modify the query so that it only includes the members of teams currently on display:

- List of all members in particular teams (unencoded):

```
GET /data/members?where=teamId IN ("{id1}","{id2}","{id3}",…) AND
status="member"
```

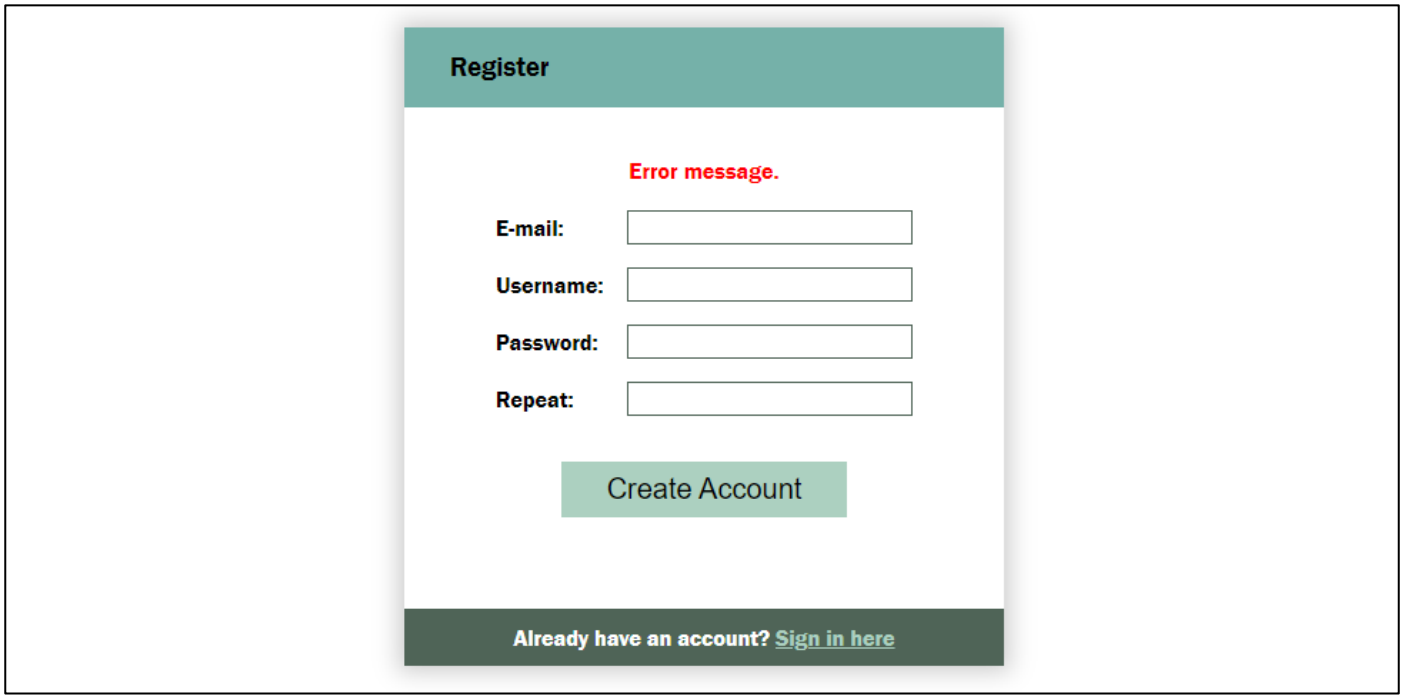
Where **{id1}**, **{id2}**, etc. is a list of comma-separated team Ids. Note that every Id **must be in double quotes** since object Ids are strings. To send this query, use the **`encodeURIComponent`** function on the query parameter value (the part after the first equal sign, beginning with **`teamId`**).

Register

You need to **write** the **functionality** for **registration** of new user. By **clicking** the "**Register**" button you have to load the **registration form**. When the "**Create Account**" button of the **form** is clicked you need to **send a post request** to register the new user. The fields by registration have validations as follows:

- **e-mail**: required, valid e-mail;
- **username**: required, at least 3 characters;
- **password**: required, at least 3 characters/digits.

If there is a **validation error** you need to show the **div with class "error"** to visualize the **message**, otherwise the **div is not displayed**.



If the registration is **successful** you can redirect to **My Teams page**.

Login

If the user **already has registration**, they **may login** by using the **login form**. If there is an error, it should be shown the same way as described for the registration.

After **successful** login the user should be **redirected** to the **My Teams page**.

Logout

The **logged in** user **can be logged out** by clicking the **logout button**. Write the functionality for this action. After logout **redirect** to **Home page**.

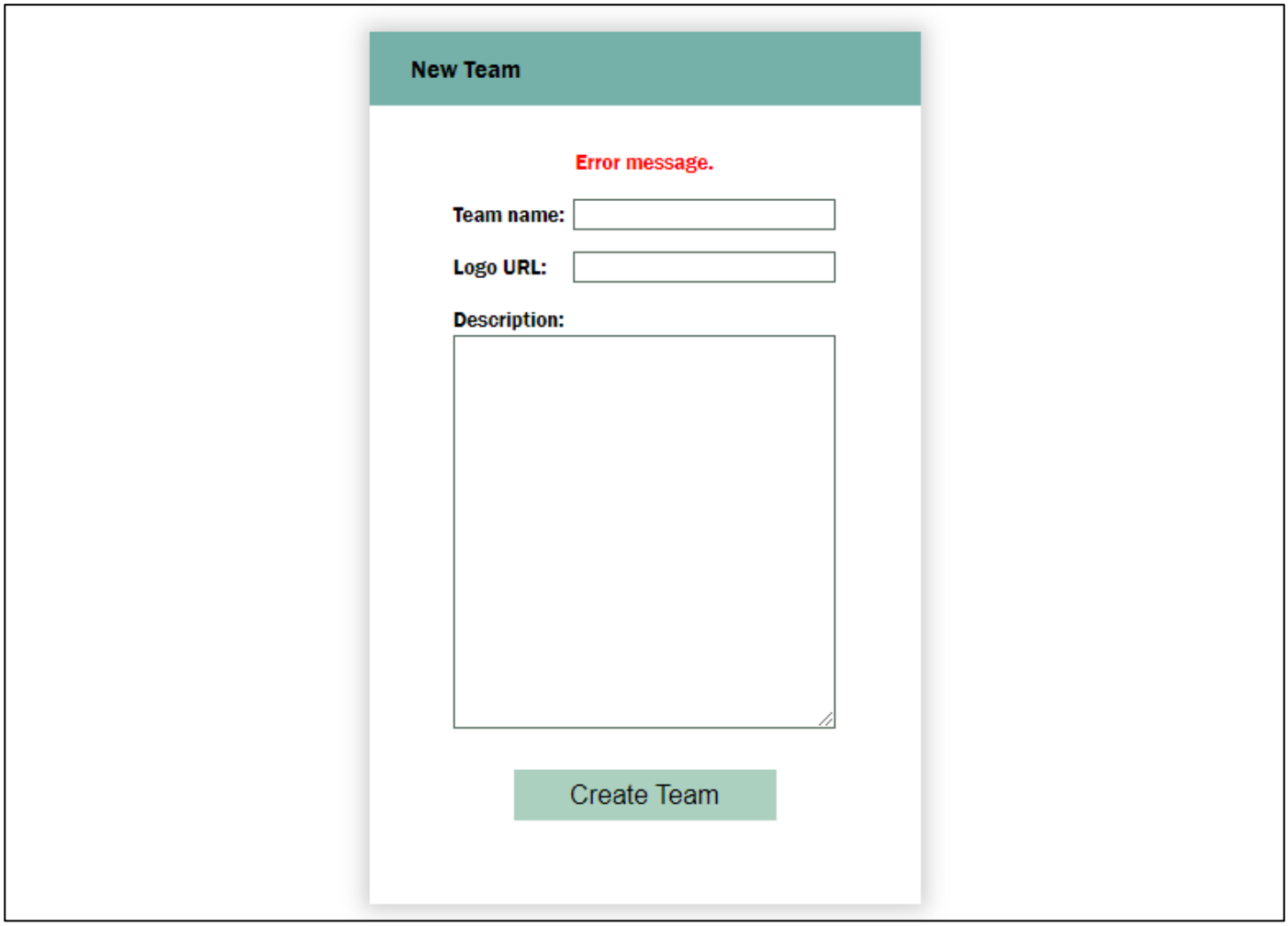
New Team

The Create new team page is a form with the following fields and validations:

- **name** (Team name): required, at least 4 characters,
- **logoUrl**: required,
- **description**: required, at least 10 characters

If there is an error by the creation of new team, display the error div with the corresponding message.

After successful creation of team, **redirect** to "**Team Details" page** of the new created team.



Use the following endpoint (body contains **`name`**, **`imageUrl`**, **`description`**):

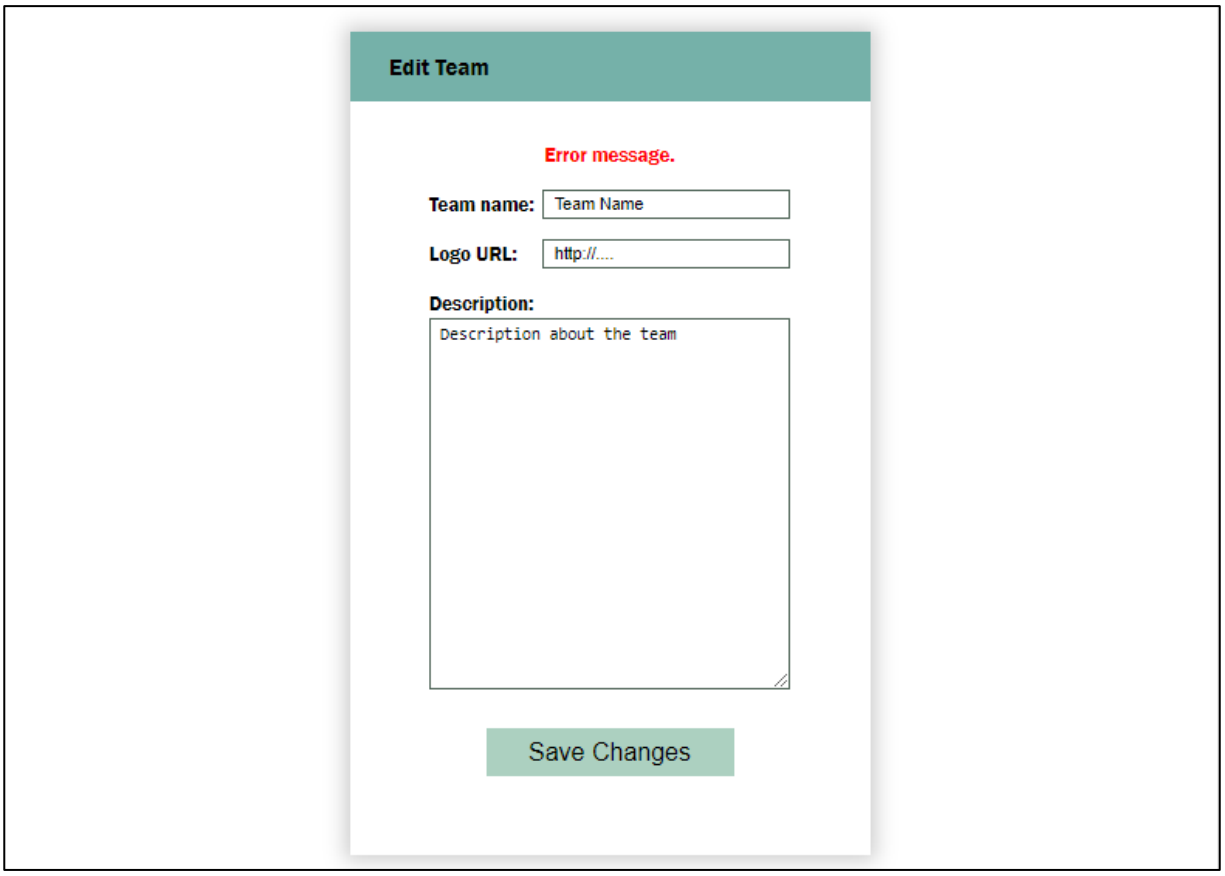
```
POST /data/teams
```

Note that when a new team is created, the **owner is not automatically added** as a member by the server – you need to **do this on behalf of the user**. Once you have obtained the newly created team (returned by the REST service), follow the instructions and endpoints listed in the section Team Details to first **send a membership request** for the user and then to **approve that request**.

Edit Team

If the **user** is the **creator** of the team, by clicking the "**Edit**" button in the details form, can see the **Edit form**. When loaded the **form fileds should be filled up** with the data of the team. After change and before sending the **PUT request** a **validation** should be made. The validation is the same as by creating a team. If **error** occurs it should be shown in the **div with class "error"**.

After **successful** edit, **redirect to Details page**.



Use the following endpoints:

- Load a selected team for editing:

```
GET /data/teams/:id
```

- Edit team:

```
PUT /data/teams/:id
```

Team Details

Another view is the Team Details page. It is accessible for logged-in, but also for guest users.

Use the following endpoints to implement page functionality:

- Load information for a single team:

```
GET /data/teams/:id
```

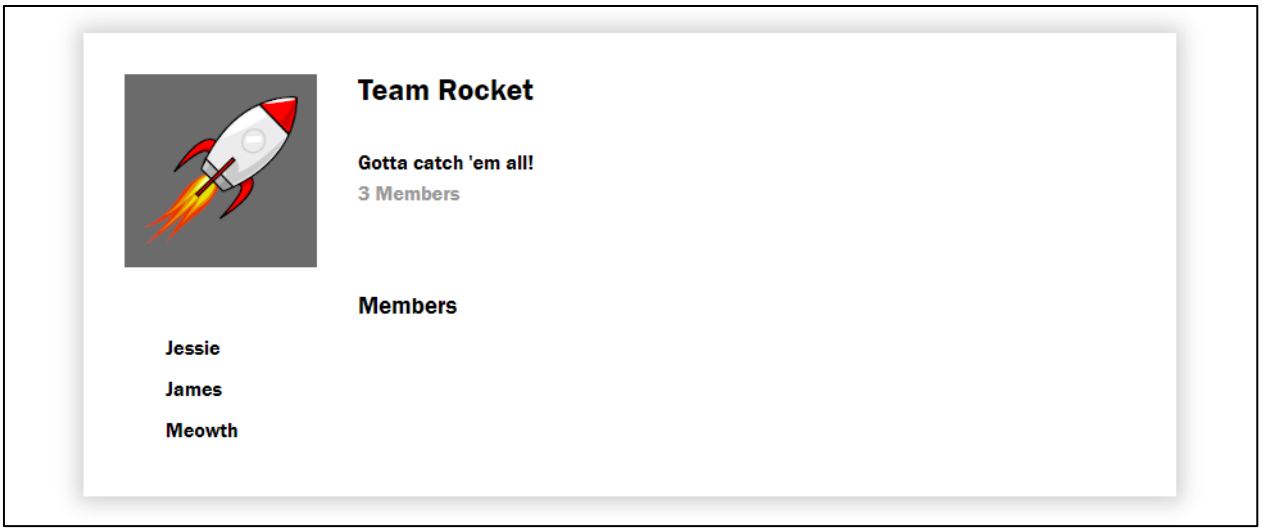
- Get a list of all memberships (both members and pending requests):

```
GET
/data/members?where=teamId%3D%22${teamId}%22&load=user%3D_ownerId%3Ausers
```

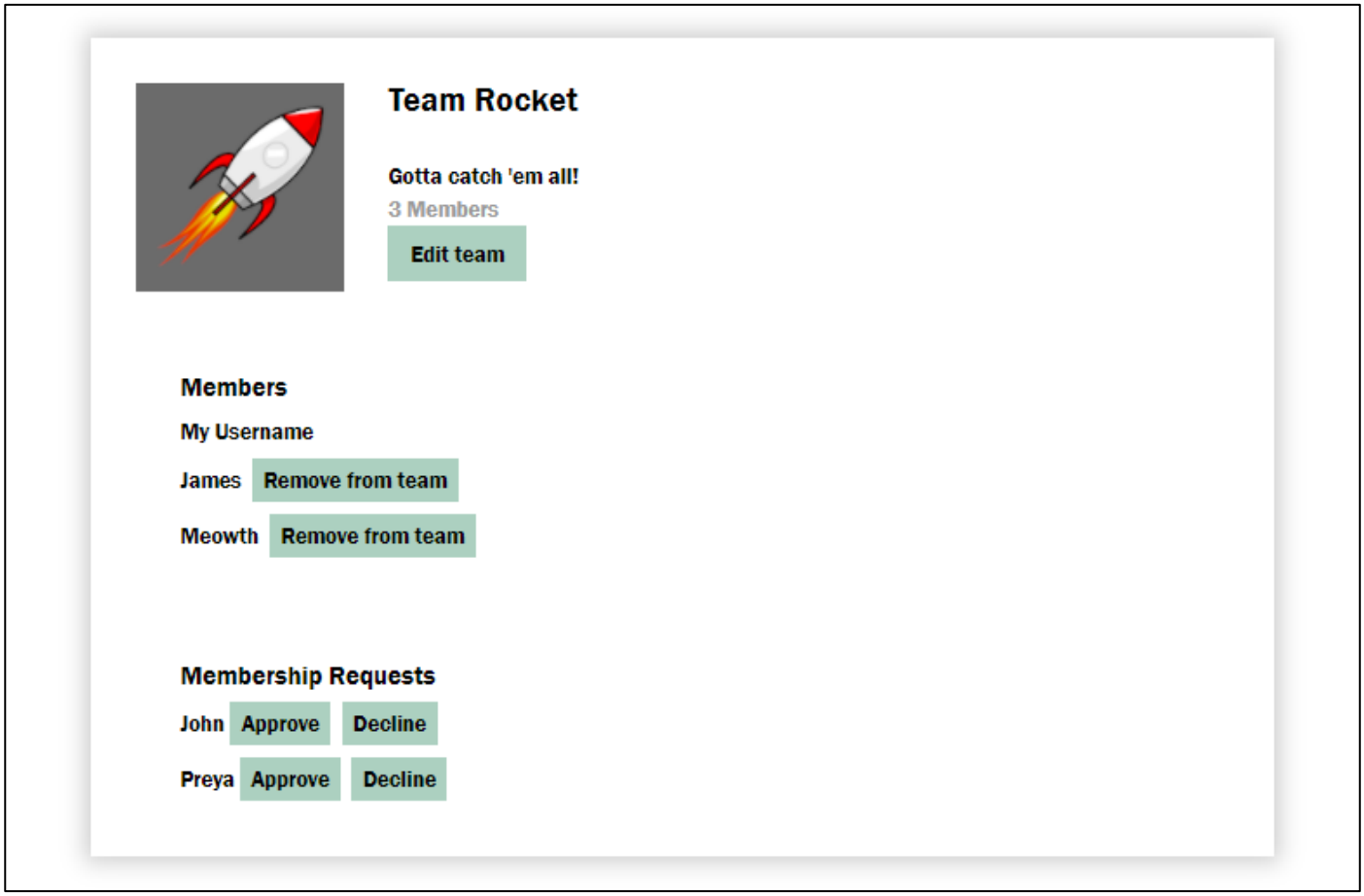
  This request will return an array of all memberships for the team, which you must them process, depending on the value of the **`status`** field. The returned objects will laso have a property **`user`**, which contains the user record for the membership. Use the property **`username`** of each user to display their record on the page.

Depending to the status of the current user, different actions are available on this page:

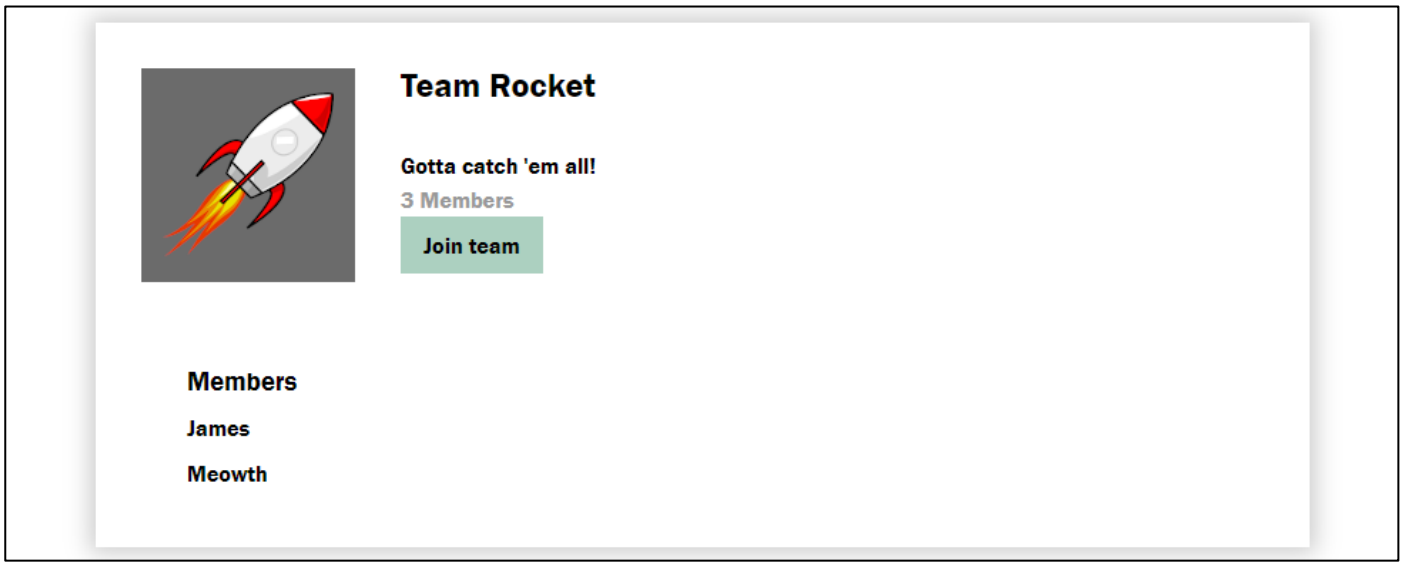
- The **GUEST users** (no user session present) can't see any buttons, but can see the names of the members (the names of the users are the usernames of the registered users).



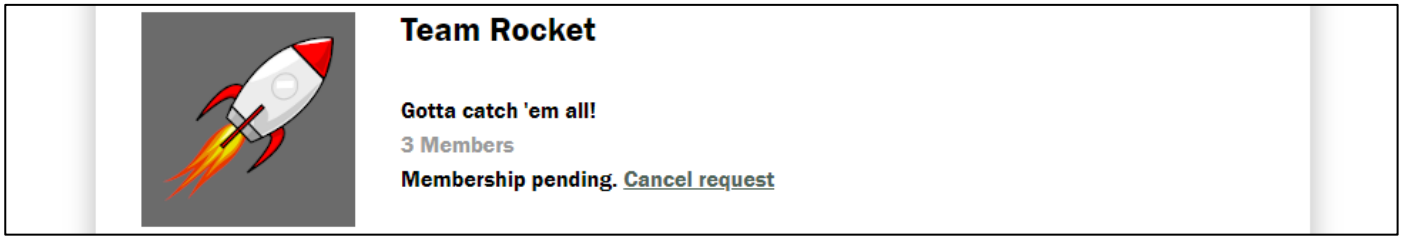
- The **owner** of the team (current user's Id matches the **`_ownerId`** of the team) can see the "Edit" button, the "Remove from team" button (for all users, except for himself) and a list with pending requests for joining the team (also the "Approve" and "Decline" buttons)



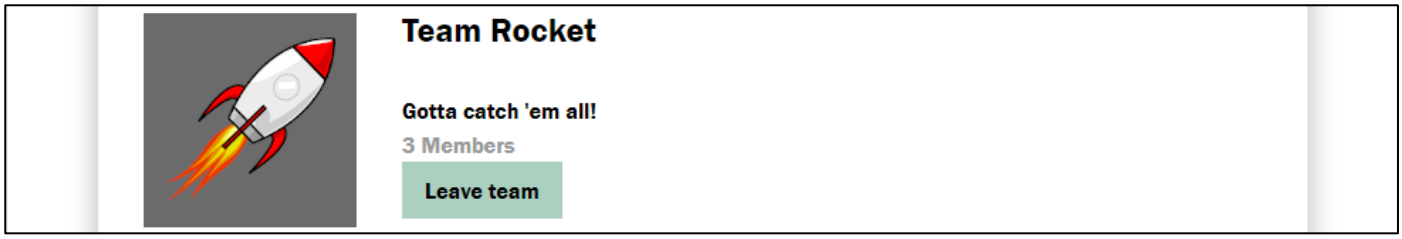
- **Logged user**, who is **not a member** of the team (there is no request with **`_ownerId`** matching the Id of the current user) can see the list with member names and the button "Join Team"



- **Logged user** with **PENDING** reques of the team (there is a matching request by **`_ownerId`** and it has a **`status`** "pending") can see the list with member names and Membership pending and "Cancel request" button.



- **Logged user**, who is **MEMBER** of the team (there is a matching request by **`_ownerId`** and it has a **`status`** "member") can see the list with member names of the team and the button "Leave Team". Note that **owner** of a team **should not** be able to leave the team.



Use the following endpoints for actions:

- Request to become a member (body contains only **`teamId`** as property):

```
POST /data/members
```

- Approve membership (update request by changing **`status`** to "member"):

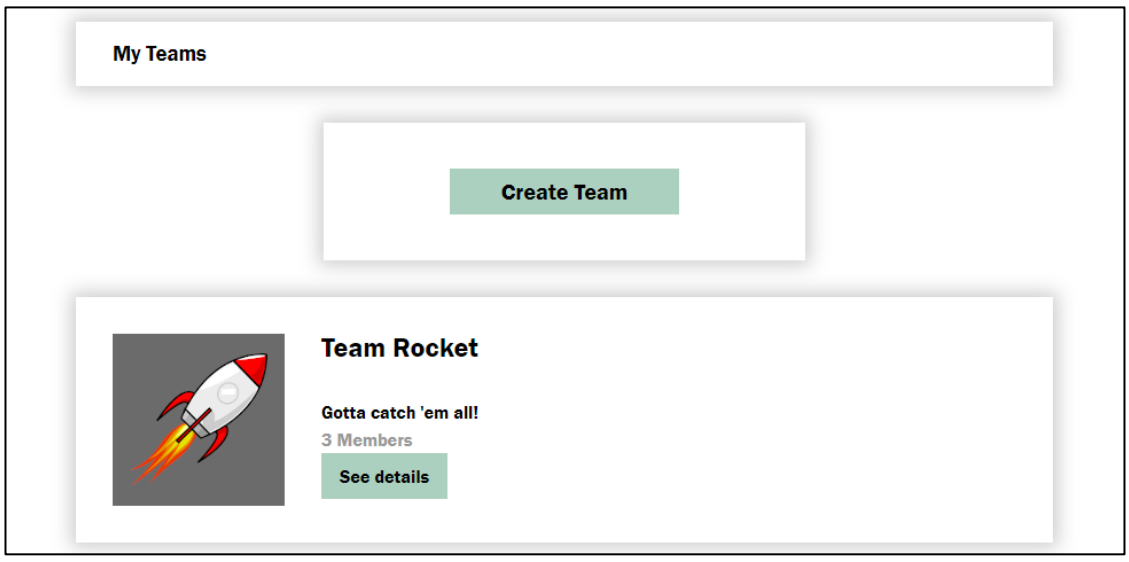
```
PUT /data/members/:id
```

- Cancel request/leave team (**user**); Decline request/remove member from team (**owner**):

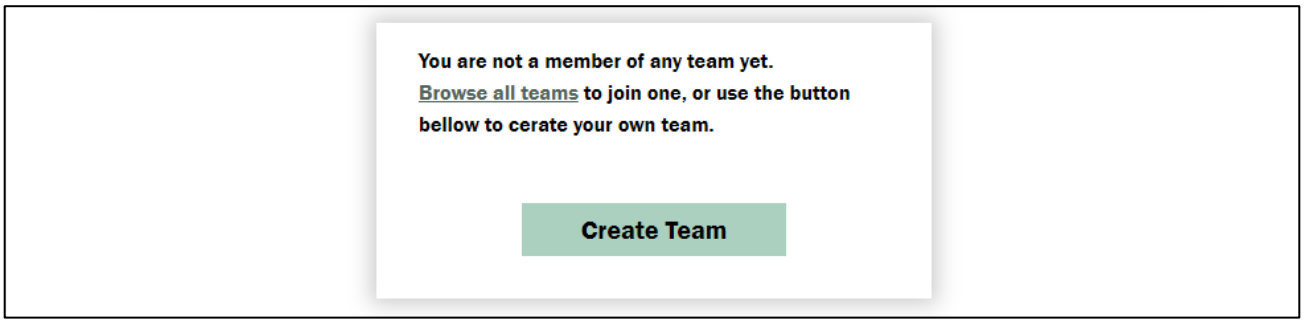
```
DELETE /data/members/:id
```

My Teams

This view displays a list of all teams of which the current user is a member.



If the logged-in user **is not a member** or **creator** of a team, you should show this view with the button "**Create Team**":



Use the following endpoints:

- List of all teams, where the current user is a member (this should include the owner, if you have followed the instructions to automatically add them after team creation):

```
GET
/data/members?where=_ownerId%3D%22{userId}%22%20AND%20status%3D%22member%22&load=team%3DteamId%3Ateams
```

  This request will return a **compound object** with information about the **membership** – you need to use the **`team`** property of each of the objects in the returned array to access team information, such as **`name`** and **`_id`**.

- List of all members in particular teams (unencoded):

```
GET /data/members?where=teamId IN ("{id1}","{id2}","{id3}",…) AND status="member"
```

Where **{id1}**, **{id2}**, etc. is a list of comma-separated team Ids. Note that every Id **must be in double quotes** since object Ids are strings. To send this query, use the **`encodeURIComponent`** function on the query parameter value (the part after the first equal sign, beginning with **`teamId`**).

Modal

An HTML example is included with **id "overlay"** (initially set to **`display:none`**). If there is a need of **confirmation** or there is an **error** (not an error in the forms validation) the **modal** should be shown with the corresponding **message** and **buttons**. Usage of this modal is optional.