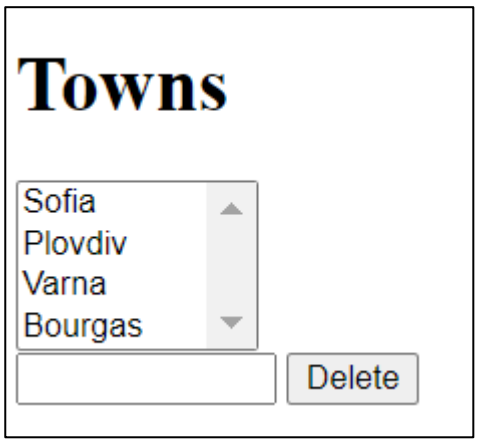
Exercises: Containers and Docker

Problems for exercises for the "Software Engineering and DevOps" course @ SoftUni.

The "Towns" Project

Note that for this exercise you should have a GitHub account.

The **"Towns" project** provides a simple HTML and JavaScript based Web front-end interface to **view**, **add**, **delete** and **shuffle** a **list of towns**. The project is **unfinished**, so some of the **functionality is already implemented** (view and delete) and **other functionality is to be implemented** (add and shuffle towns + CSS styling). This is how the "Towns" project looks like at the beginning:



At the start, in the GitHub repo you will have **3 source code files**: `towns.html` + `towns.css` + `towns.js`, and a `README.md` documentation file:

1. Roles Assignments

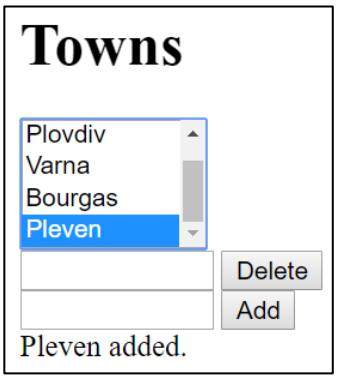
Your task is to **work in teams of 3 students or work alone with several roles to simulate multi-user interaction**, where each role follows the instructions for certain team member – **Editor**, **Shuffler** and **Styler**.

If you work in a **team of 3 students**, one of you should also take the role of the **Team Leader**.

If you work **alone**, you should work with a **fourth** role – the **Team Leader**.

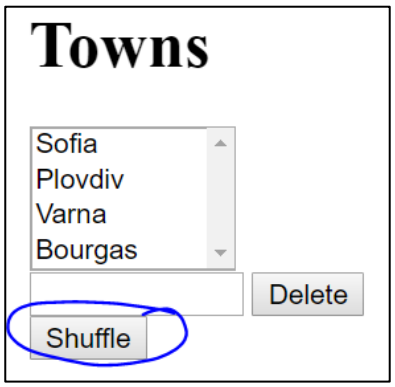
Editor

The Editor should implement "**add new town**" functionality:



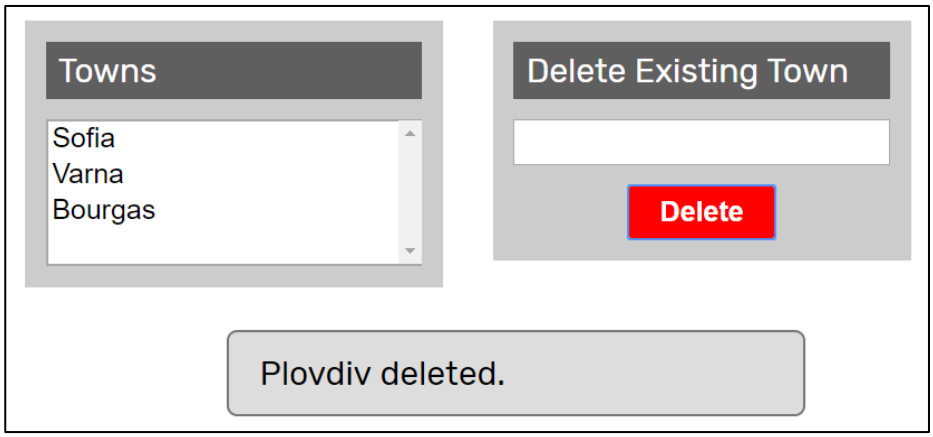
Shuffler

The **Shuffler** should implement "**shuffle towns**" functionality.



Styler

The **Styler** should add some **CSS styling** and **improve** the **HTML UI**.



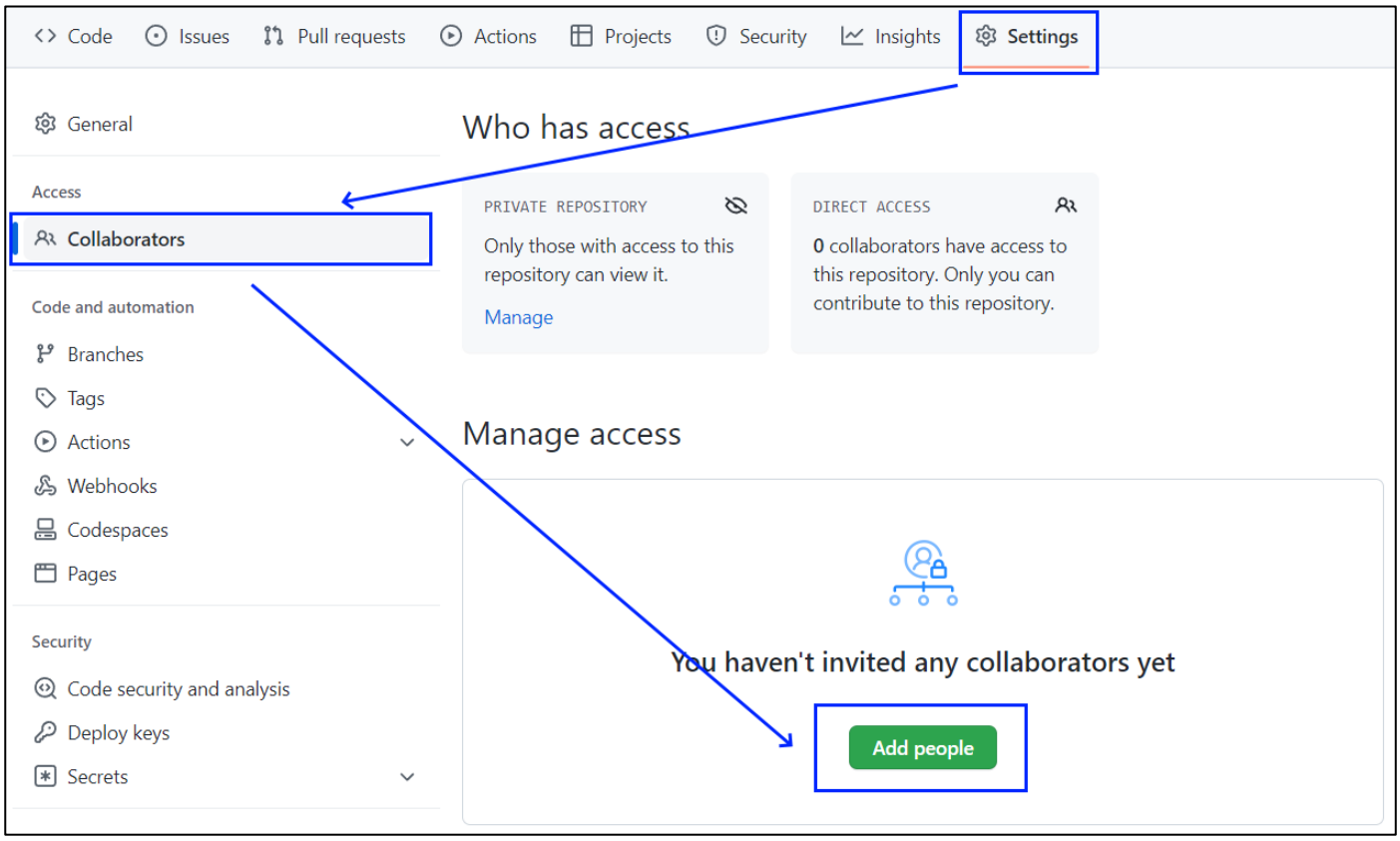
2. Fork the Repo

The **team leader** does the following:

Fork the "**Towns**" repo from GitHub: https://github.com/SUContent/Towns

3. Invite the Team Members

The **team leader** invites the **other team members** as **collaborators** in the new GitHub repo.



- An **email invitation** will be sent for each **invited collaborator**.

4. Team Members Should Clone the Project

Each team member clones the project from the **team leader's GitHub repository**:

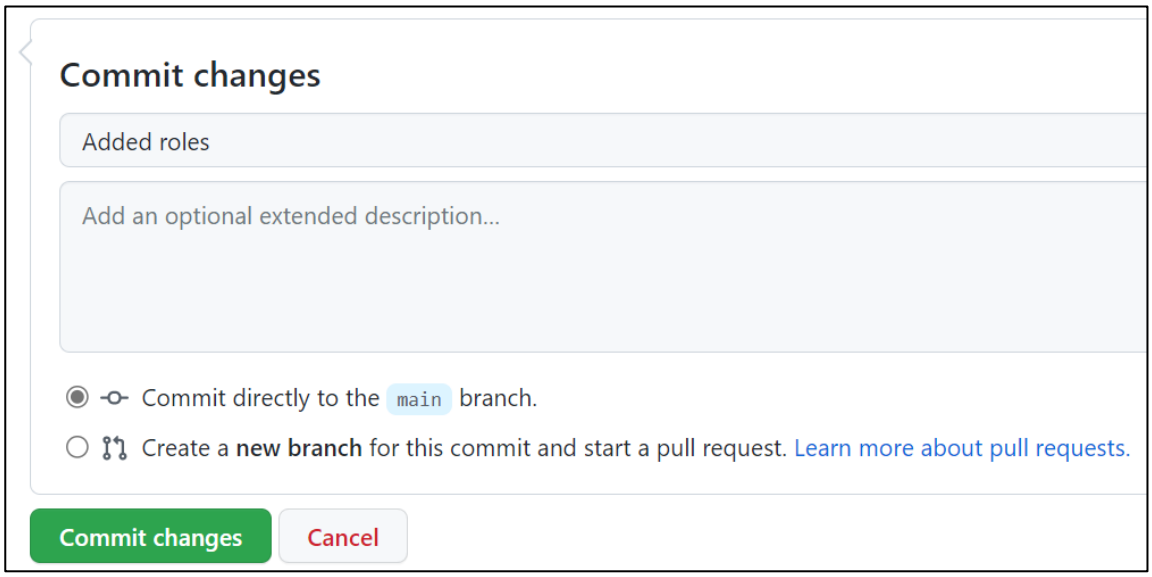
```
git clone https://github.com/<team-leader-username>/Towns
```

5. Edit the Project Description

This step should be executed only **after each team member have already cloned the project** locally.

The **team leader** makes **changes** in **`README.md`** file from the GitHub's project Web site, to describe with a text which team member which role will take, e.g.,

```
# Roles

- {Name1} takes the role "Editor"
- {Name2} takes the role "Shuffler"
- {Name3} takes the role "Styler"
```



6. Implement Project Functionalities

Each **team member implements** different **functionality locally** (each **member** has its **own instructions**).

- **Member #1**: **Editor**'s functionality (**add town**)
- **Member #2**: **Shuffler**'s functionality (**shuffle towns**)
- **Member #3**: **Styler**'s functionality (**improve styling and CSS**)

NOTE: If you are working alone, don't forget that you should fulfil the three roles **simultaneously**.

Editor

The **Editor** should already have cloned the forked repo.

Step 1: New Town Textbox + Button

Now, in **`towns.html`**, the editor should add a textbox + button for creating a new town:

```
<div>
    <input type="text" id="townNameForAdd" />
    <button id="btnAdd">Add</button>
</div>
```

Step 2: New Town JavaScript Code

In **`towns.js`**, the editor has to add a new function for adding a new town:

```
function addTown() {
      let townName = $('#townNameForAdd').val();
      $('#townNameForAdd').val('');
      $('#towns').append($('<option>').text(townName));
      $('#result').text(townName + " added.");
}
```

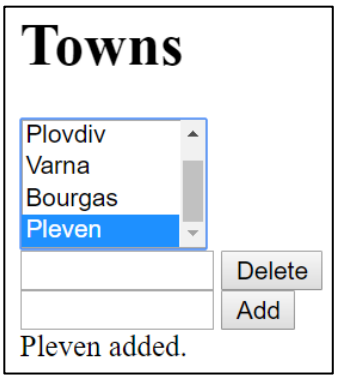
Also, the editor should add a code to **attach an event handler** to invoke the new function when the **[Add]** button is pressed. In the start of the JS code, he adds the code marked in **blue** below:

```
$(document).ready(function() {
      …
      $('#btnAdd').click(addTown);
```

```
});
```

Step 3: Test the Project functionality

Editor now should **test the project functionality** to see whether the styling and effects work correctly, as well as whether the entire project works as expected:



Step 4: Commit All Changes in the Local Repo

Editor adds and **commits** in Git all local changes:

```
git commit -a -m "Implemented functionality to add a new town"
```

Step 5: Push the Local Commits to GitHub

Editor pushes all the changes to Git:

```
git push
```

Step 6: Resolve Any Conflicts

```
git pull
```

Editor should edit all files and **fixes** the code in order to **merge** all concurrent **changes correctly**.

Then, Editor **adds** and **commits** in **Git** the **merged files**:

```
git commit -a -m "Implemented functionality to add a new town + merged the
conflicting files"
```

Finally, **Editor should push all his changes** again to Git:

```
git push
```

Shuffler

The **Shuffler** should already have cloned the forked repo.

Step 1: New Town Textbox + Button

Now, in **`towns.html`**, the editor should add a textbox + button for shuffling the towns:

```
<div>
    <button id="btnShuffle">Shuffle</button>
</div>
```

Step 2: New Town JavaScript Code

In **towns.js**, the editor has to add a new function for shuffling the towns:

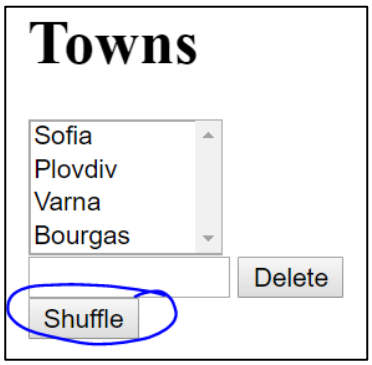
```
function shuffleTowns() {
      let towns = $('#towns option').toArray();
      $('#towns').empty();
      shuffleArray(towns);
      $('#towns').append(towns);
      $('#result').text("Towns shuffled.");

      function shuffleArray(array) {
            for (var i = array.length - 1; i > 0; i--) {
                  var j = Math.floor(Math.random() * (i + 1));
                  var oldElement = array[i];
                  array[i] = array[j];
                  array[j] = oldElement;
            }
      }
}
```

Also, the editor should add a code to **attach an event handler** to invoke the new function when the **[Add]** button is pressed. In the start of the JS code, he adds the code marked in **blue** below:

```
$(document).ready(function() {
      $('#btnShuffle').click(shuffleTowns);
});
```

Step 3: Test the Project functionality

Shuffler now should **test the project functionality** to see whether the styling and effects work correctly, as well as whether the entire project works as expected:



Step 4: Commit All Changes in the Local Repo

Shuffler adds and **commits** in Git all local changes:

```
git commit -a -m "Implemented functionality to shuffle the existing towns"
```

Step 5: Push the Local Commits to GitHub

Shuffler pushes all the changes to Git:

```
git push
```

Step 6: Resolve Any Conflicts

```
git pull
```

Editor should edit all files and **fixes** the code in order to **merge** all concurrent **changes correctly**.

Then, **Shuffler adds** and **commits** in **Git** the **merged files**:

```
git commit -a -m "Implemented functionality to shuffle the existing towns + merged
the conflicting files"
```

Finally, **Shuffler should push all his changes** again to Git:

```
git push
```

Styler

The **Styler** should already have cloned the forked repo.

Step 1: Improve the HTML Structure

Styler wants to style the site to look better but the **HTML structure** does not allow writing CSS, so they modifie **`towns.html`** and introduces a new way to structure the content as a sequence of **articles** holding **headers** and other elements after the header:

```
<article>
      <header>Towns</header>
      <select id="towns" size="4">
            <option>Sofia</option>
            <option>Plovdiv</option>
            <option>Varna</option>
            <option>Bourgas</option>
      </select>
</article>

<article>
    <header>Delete Existing Town</header>
    <input type="text" id="townName" />
    <button id="btnDelete">Delete</button>
</article>
```

Step 2: Write the CSS Code

Styler now rewrites the entire **`towns.css`** file from scratch:

```
@import url('https://fonts.googleapis.com/css?family=Rubik');

body {
      font-family: 'Rubik', sans-serif;
}

* {
      box-sizing: content-box;
}

article {
      background: #CCC;
      width: 180px;
      padding: 10px;
```

```
	margin: 10px;
	display: inline-block;
	vertical-align: top;
}

article>header {
	background: #5F5F5F;
	color: white;
	margin: 0px 0px 10px 0px;
	padding: 4px 6px;
}

article>header>h1 {
  margin: 0px;
}

article>select {
	width: 178px;
}

article>input {
	width: 176px;
}

article>button {
	display: block;
	margin: 10px auto 0px auto;
	border: none;
	border-radius: 3px;
	padding: 5px 15px;
	background: green;
	color: white;
	font-weight: bold;
}

article>button:hover {
	box-shadow: 0px 0px 10px white;
	cursor: pointer;
}

button#btnDelete {
	background: red;
}

#result {
	display: none;
	width: 50%;
	margin: 10px auto;
	padding: 10px 15px;
	background: #DDD;
	border-radius: 5px;
	border: 1px solid #777;
}
```

The new CSS code assumes the HTML uses **articles** with **headers** for its major areas. It displays the articles in a nice-looking way. The CSS also hides by default the result info box and assumes it will be shown by the JS code later.

Step 3: Add "Auto Hide" Effect for the Info Messages

After a button is clicked (e.g. **[Delete]**), the result of the performed action is shown into an info box (**#result**). Styler modifies this behavior, so that the info box is by default hidden, then it displays a message for 3 seconds, then it disappears. First, he adds a JS library in **towns.html** to enable animation effects with jQuery:

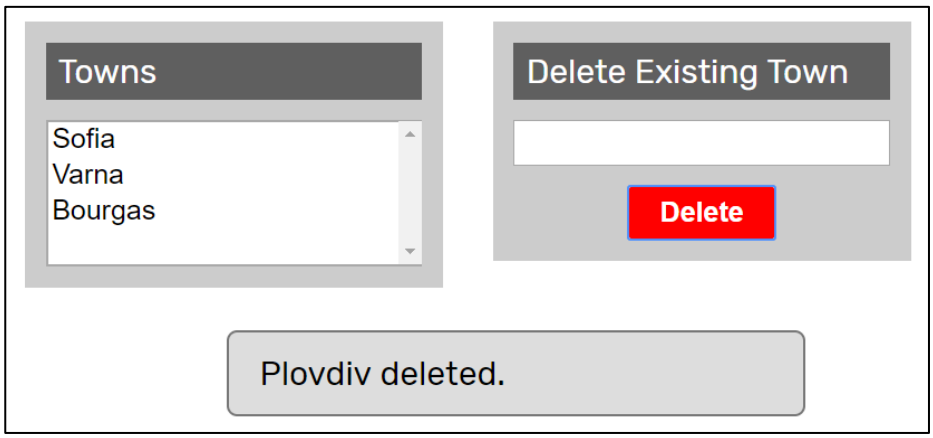
```
<script src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.5/jquery-ui.min.js"></script>
```

Next, he modifies the **deleteTown()** function in **towns.js** to display messages through a new function **showMessage(msg)**:

```
function deleteTown() {
    let townName = $('#townName').val();
    $('#townName').val('');
    let removed = false;
    for (let option of $('#towns option')) {
        if (option.textContent == townName) {
            removed = true;
            option.remove();
        }
    }
    if (removed)
        showMessage(townName + " deleted.");
    else
        showMessage(townName + " not found.");
}


function showMessage(msg) {
    $('#result').text(msg).css("display", "block");
    setTimeout(function () {
        $('#result').hide('blind', {}, 500);
    }, 3000);
}
```

Step 4: Test the Project Functionality

Styler now **tests the project functionality** to see whether the styling and effects work correctly, as well as whether the entire project works as expected:



Step 5: Commit All Changes in the Local Repo

Styler adds and **commits** in Git all local changes:

```
git commit -a -m "Implemented functionality to add a new town"
```

Step 6: Push the Local Commits to GitHub

Styler pushes all his changes to Git:

```
git push
```

Step 7: Resolve Any Conflicts

In case of **conflict** Styler **pulls**, **merges**, then **pushes** again:

```
git pull
```

Styler edits all files and fixes the code in order to merge all concurrent changes correctly.

Then, **Styler adds** and **commits in Git** the **merged files**:

```
git commit -a -m "Improved the UI: added CSS styles + HTML structure + JS effects"
```

Finally, **Styler pushes all his changes** again to Git:

```
git push
```