Front-End Academy

JavaScript Capstone

"Stay Organized"

Project Description

You will build a website that allows users to manage a list of the things they need to accomplish. For example:

```
Category: "Personal Task"
Description: "Finish studying for ENG 211 exam"
Deadline: "2022-04-15"
Priority: "Medium"
```

Setting up the REST API

The capstone project is supported by a Node.js REST API that you can clone from:

```
git clone
https://github.com/DevelopIntelligenceBoulder/stay-organized-workshop-express-s
erver
```

After you clone this repository to your local computer, change directories (cd) into the newly cloned project folder and Install the projects dependencies with NPM

```
cd stay-organized-workshop-express-server
npm install
```

Start the local server using the command:

```
npm start
```

Expected Output

```
App listening at port 8083
```

You can verify the server is working as expected by using a browser to access http://localhost:8083/api/users

REST API Endpoints

The supported endpoints are shown below. Please note you will need to prepend them with http://localhost:8083/ before sending the request to the API

```
GET api/users
GET api/categories
GET api/todos
GET api/todos/1                          (other other todo id)
GET api/todos/byuser/1                   (or other user id)
GET api/username_available/sallie71      (or other user id)
GET api/users/gamer04                    (or other user id)
POST api/todos
PUT api/todos/2                          (or other todo id)
POST api/users
```

You can use a browser to test the GET endpoints. VS Code's REST Client shown on Thursday to can be used to test the POST and PUT endpoints. Read the link provided!

The site already has some users registered. You can see them by looking at the server's `data/users.json` file. It resembles:

```
[
  {
    "id":1,
    "name": "Ian Auston",
    "username": "gamer04",
    "password": "gamer04!"
  },
  ...
]
```

IMPORTANT: Professional systems do not store passwords in "clear text" (the way they were typed in). When developers store passwords in clear text and an unauthorized person gains access to the server, they quickly learn the passwords to all your users. Good systems store a "hashed" (modified) version of the password. That, however, is beyond the scope of this academy.

Page Requirements

Because of the short week (Summer Fridays!), you will only need to provide the ability for users to VIEW or ADD ToDo tasks.

Your website should include the pages below

- A home page named `index.html` that highlights our "Stay Organized" web site.

- A page that allows you to view all of the ToDo tasks for a user by picking the user from a dropdown. The page will be named `user-todos.html`
- A page that allows the user to add a ToDo task named `new-todo.html`
- **(Bonus)** A page named `new_user.html` that allows a new user to register for our services

There isn't any security on your web site. One user can see the tasks of another user, or even assign a ToDo task to them. Don't let this worry you. Just imagine this is the site you use to manage your family!

Details about `user-todos.html`

This page allows the user to view all of the ToDo tasks for a specific registered user by picking that user from a dropdown.

When the page loads, it will display a dropdown listing all of the registered users. You can find that information by sending a GET request to the REST API endpoint `api/users`. Note that passwords are NOT returned in the response.

When a registered user is selected from the dropdown list, the page should fetch and display that user's ToDo tasks. The REST API allows you to find the ToDo Tasks of a single user by sending a GET request to `api/todos/byuser/1` where 1 is the `id` of the user whose ToDos you want.

```
 [
  {
    "id": 1,
    "userid": 5,
    "category": "Personal Task",
    "description": "Finish studying for ENG 211 exam",
    "deadline": "2020-11-15",
    "priority": "Medium",
    "completed": false
  },
  ...
]
```

Display the ToDo tasks in some reasonable format.

IMPLEMENTATION HINTS

- When you create their dropdown list, set the text of each option to the user's `name` and the value of each option to that user's `id`. You must do this because you need the user's `id` to retrieve their ToDo tasks.

BONUS IDEA: Display a small ✓ or X image rather than true or false for the completed value

BONUS IDEA: Rather than displaying each ToDo task in a table with a long row of many data fields, display the only the description and deadline and provide a "See Details" hyperlink to a `todo_details.html` to view all of the details for that ToDo task. ***NOTE: This bonus will take time -- DO NOT attempt it until you have the `new-todo.html` page working.***

Details about `new-todo.html`

This page allows a user to enter a new ToDo task. You must collect the following information for a ToDo task:

userid	- a number that identifies the user (see notes below)
category	- a type of ToDo that comes from a dropdown displaying values retrieved from the API by calling `/api/categories`
description	- a potentially long string describing the task. Use a `<textarea>` to let user enter a long description.
deadline	- a date deadline (when you test, use the format YYYY-MM-DD but you don't have to enforce it via code)
priority	- an urgency that comes from a dropdown with hard-coded the options "Low", "Medium" or "High"

You should use a dropdown list to allow the user to select the person to whom the ToDo task is assigned. Like the `todos.html` page, you must get the list of registered users from the API by sending GET request to `api/users`. When you create the dropdown, remember to set the text of each option to the user's `name` and the value of each option to that user's `id`.

To make the "add ToDo" feature work, you must send a POST request to `api/todos`. The POST request must send the data in the form. The names for the data sent must ***exactly match*** what is shown in `todos.json` EXCEPT you will not send a ToDo `id` or `completed` value. They will be auto-generated.

OTHER IMPLEMENTATION HINTS:

- You can get the list of categories to load into a dropdown list by calling your API using `api/categories`. NOTE: Although categories have an `id`, don't place it in the `<option>` element's `value`. When you POST the ToDo item to the API, you need to send the actual category text (ex: "Errand"), not the id.

- Hard-code the options for priority dropdown. Use "Low", "Medium" and "High".

- The format of the deadline matches the format generated by an HTML5 input field with `type="date"`.

- You will NOT prompt the user for a completed value. The POST method on the server will automatically assign new ToDo tasks a completed value of false.

(Bonus) Details about `new_user.html`

The site already has some users registered. If you choose to implement the register page, you will collect 3 pieces of information from the user:

name
username
password

Under the hood, the `users.json` file resembles:

```
[
  {
    "id":1,
    "name": "Ian Auston",
    "username": "gamer04",
    "password": "gamer04!"
  },
  ...
]
```

To add a user, you must send a POST request to `api/users`. The field names for the user data must match what is shown in `users.json` EXCEPT you will not send an `id`. It will be auto-generated.

IMPLEMENTATION HINTS:

- Provide two password fields. Make sure they contain the same text before POSTing your new user data to the API. If the passwords don't match, display a message to the user.

- When adding the user, check to make sure the API call didn't return a 403. If it did, you tried to add a user and the username was already in use. If it returns an error, display a meaningful message to the user.

- Fun bonus: Before trying to add the user, check to see if the user name they selected is available by calling the API using `api/username_available/FARMGIRL1` where FARMGIRL1 is the username you are checking. The API method does a case insensitive

search on existing user names and returns the single word "YES" or "NO" indicating whether the name is available.

What Makes a Good Capstone?

You should:

- build a consistent look-and-feel throughout the site with intuitive navigation
- implement at least the required pages
- have a responsive user interface

You should adhere to best practices such as:

- have a good directory structures (ex: css, images and scripts folders)
- include Bootstrap from a CDN
- have good file naming conventions (ex: lowercase file names with no spaces)
- have well- formatted HTML, CSS and JavaScript (indentions, blank lines, etc)
- use good names for your HTML elements and JavaScript variables/functions
- use HTML, CSS and JavaScript comments effectively

Make sure that:

- you use validators to ensure you have no HTML or CSS errors!
- you use a ESLint tool if you can find one to ensure you've written good JavaScript!
- there are no JavaScript errors at run time (check the Console tab in the browser)

Build a **PUBLIC** GitHub Repo for your code.

- Use an appropriate branch structure and have a commit history with meaningful comments
- Include a README.md file that describes your project and includes screen shots!
- Make sure it is on the first page of your GitHub projects!

Class Demonstrations

On the last day of the capstone week, you will present your capstone to your peers, managers, and mentors. The ITC staff will provide more details.

Typically, you will be expected to:

- Show off your website and the pages within it
- Show one interesting piece of JavaScript you wrote
- Answer questions from the audience if time permits