Merge branch 'master' of...
qwer-1234-q authored 10 hours ago

Name	Last commit	Last update
backend	Updated Swagger Docs for '/job/feed'	1 day ago
frontend	2.1 Updated - Sylvia	10 hours ago
.gitignore	Ready for 23T1	1 week ago
README.md	Updated Swagger Docs for '/job/feed'	1 day ago
bonus.md	Ready for 23T1	1 week ago
progress.csv	Ready for 23T1	1 week ago
usability.md	Ready for 23T1	1 week ago

README.md

Assessment 3 - Vanilla JS: LurkForWork

0. Change Log

13/03/2023 - Updated the description of the '/job/feed' endpoint in Swagger docs from 'all jobs' -> 'next 5 jobs'.

1. Background & Motivation

Web-based applications are becoming the most common way to build a digital capability accessible to a mass audience. While there are modern tools that help us build these rapidly, it's important to understand the fundamental JavaScript-based technology and architectures that exist, both to gain a deeper understanding for when these skills may be needed, but also to simply understand the mechanics of fundamental JS. Even when working with a high level framework like ReactJS, understanding (in-concept) the code that it is transpiled to will ensure you're a more well rounded web-based engineer.

This assignment consists of building a **frontend** website in Vanilla JS (no ReactJS or other frameworks). This frontend will interact with a RESTful API HTTP backend that is built in JavaScript (NodeJS express server) and provided to you.

A theoretical background on how to interface with this API can be found the "promises & fetch" lecture.

The web-based application you build is required to be a single page app (SPA). Single page apps give websites an "app-like feeling", and are characterised by their use of a single full load of an initial HTML page, and then using AJAX/fetch to dynamically manipulate the DOM without ever requiring a full page reload. In this way, SPAs are generated, rendered, and updated using JavaScript. Because SPAs don't require a user to navigate away from a page to do anything, they retain a degree of user and application state. In short, this means you will only ever have `index.html` as your HTML page, and that any sense of "moving between pages" will just be modifications of the DOM.

2. The Task (Frontend)

Your task is to build a frontend for a UNSW rip-off version of the popular professional social networking tool LinkedIn. If you haven't used this application before, we would recommend creating your own LinkedIn profile - it's probably good for your career anyway!

UNSW's rip-off of LinkedIn is called "LurkForWork". However, you don't have to build the entire application. You only have to build the frontend. The backend is already built for you as an express server built in NodeJS (see section 3.2).

Instead of providing visuals of what the frontend (your task) should look like, we instead are providing you with a number of clear and short requirements about expected features and behaviours.

The requirements describe a series of **screens**. Screens can be popups/modals, or entire pages. The use of that language is so that you can choose how you want it to be displayed. A screen is essentially a certain state of your web-based application.

2.1. Milestone 1 - Registration & Login (9.4%)

This focuses on the basic user interface to register and log in to the site.

2.1.1. Login

- When the user isn't logged in, the site shall present a login form that contains:
  - an email field (text)
  - a password field (password)
  - submit button to login
- When the submit button is pressed, the form data should be sent to `POST /auth/login` to verify the credentials. If there is an error during login an appropriate error should appear on the screen.

2.1.2. Registration

- When the user isn't logged in, the login form shall provide a link/button that opens the register form. The register form will contain:
  - an email field (text)
  - a name field (text)
  - a password field (password)
  - a confirm password field (password) - not passed to the backend, but an error should be thrown on submit if it doesn't match the other password
  - submit button to register
- When the submit button is pressed, if the two passwords don't match the user should receive an error popup. If they do match, the form data should be sent to `POST /auth/register` to verify the credentials. If there is an error during registration an appropriate error should appear on the screen.

2.1.3. Error Popup

- Whenever the frontend or backend produces an error, there shall be an error popup on the screen with a message (either a message derived from the backend error response, or one meaningfully created on the frontend).
- This popup can be closed/removed/deleted by pressing an "x" or "close" button.

2.2. Milestone 2 - Basic Feed (14%)

Milestone 2 focuses on fetching feed data from the API. A feed and it's associated content should only be accessible to logged in users.

2.2.1. Basic Feed

The application should present a "feed" of user content on the home page derived `GET /job/feed`. Note that the feed will only return information from people that the logged in user is watching.

The jobs should be displayed in reverse chronological order (most recent jobs first).

Each job should display:

1. Who the job post was made by
2. When it was posted
   - If the job was posted today (in the last 24 hours), it should display how many hours and minutes ago it was posted
   - If the job was posted more than 24 hours ago, it should just display the date DD/MM/YYYY that it was posted
3. The job content itself. The job content includes the following:

- An image to describe the job (jpg in base64 format)
- A title for the new job (just as a string)
- A starting date for the job (just as a string)
- How many likes it has (or none)
- The job description text
- How many comments the job post has

2.3. Milestone 3 - Advanced Feed (9.4%)

Milestone 3 focuses on a richer UX and will require some backend interaction.

2.3.1. Show likes on a job

- Allow a user to see a list of all users who have liked a job. In terms of how it is displayed, consider your preferred user experience approach out of the following 3 options:
  - The list of names is visible on each job in the feed by default
  - The list of names is visible on a job in the feed if a show/hide toggle is clicked (hidden by default).
  - The list of names is visible in a popup, modal, or new screen, when a button/link is clicked on the feed.

2.3.2. Show comments on a job

- Allow a user to see a list of all the comments on the job. Each comment should contain at minimum the user's name and their comment. In terms of how it is displayed, consider your preferred user experience approach out of the following 3 options:
  - The list of names and comments are visible on each job in the feed by default
  - The list of names and comments are visible on a job in the feed if a show/hide toggle is clicked (hidden by default).
  - The list of names and comments are visible in a popup, modal, or new screen, when a button/link is clicked on the feed.

2.3.3. Liking a job

- A user can like a job on their feed and trigger a api request (`PUT /job/like`)
- For this milestone, it's OK if the like doesn't appear/update until the page is refreshed.

2.3.4. Feed Pagination

- Users can page between sets of results in the feed using the position token with (`GET /job/feed`).
- Note: You will automatically receive marks for this section if you end up implementing the infinite scroll alternative in a later milestone.

2.4. Milestone 4 - Other users & profiles (14%)

Milestone 4 focuses predominately on user profiles and how users interact with them.

2.4.1. Viewing others' profiles

- Let a user click on a user's name from a job, like, or comment, and be taken to a profile screen for that user.
- The profile screen should contain any information the backend provides for that particular user ID via (`GET /user`).
- The profile should also display all jobs made by that person. You are not required to show likes and/or comments for each job here.
- The profile should also display somewhere all other users this profile is watched by (information via `GET /user`). This should consist of a list of names (which for each name links to another profile), as well as a count somewhere on the page that shows the total number of users they are watched by.

2.4.2. Viewing your own profile

- Users can view their own profile as if they would any other user's profile
- A link to the users profile (via text or small icon) should be visible somewhere common on most screens (at the very least on the feed screen) when logged in.

2.4.3. Updating your profile

- Users can update their own personal profile via (`PUT /user`). This allows them to update their:
  - Email address
  - Password
  - Name
  - Image

2.4.4. Watching / Unwatching

- Watching on user profiles:
  - When a logged in user is visiting another user's profile page, a button should exist that allows them to "watch" the other user (via `PUT user/watch`).
  - If the logged in user already watches this person, an unwatch button should exist.
- Somewhere on the feed screen a button should also exist that prompts the enter to enter an email address in a popup. When entered, the email address is sent to `PUT /user/watch` to watch that particular user.

2.5. Milestone 5 - Adding & updating content (9.3%)

Milestone 5 focuses on addition and removing both content and comments.

2.5.1. Adding a job

- Users can upload and job new content from a modal, component, or seperate screen via (`POST /job`)
- How users open this component, modal, or separate screen can be found in a single or multiple places, and should be easily and clearly accessible.

2.5.2. Updating & deleting a job

- Let a user update a job they made or delete it via (`DELETE /job`) or (`PUT /job`).

2.5.3. Leaving comments

- Users can write comments on "jobs" via (`POST /job/comment`)

2.6. Milestone 6 - Challenge Components (`advanced`) (9.3%)

2.6.1. Infinite Scroll

- Instead of pagination, users an infinitely scroll through results. For infinite scroll to be properly implemented you need to progressively load jobs as you scroll.

2.6.2. Live Update

- If a user likes a job or comments on a job, the job's likes and comments should update without requiring a page reload/refresh. This should be done with some kind of polling.

Polling is very inefficient for browsers, but can often be used as it simplifies the technical needs on the server.

2.6.3. Push Notifications

- Users can receive push notifications when a user they watch posts a job. To know whether someone or not has posted a job, you must "poll" the server (i.e. intermittent requests, maybe every second, that check the state). You can implement this either via browser's built in notification APIs or through your own custom built notifications/popups. The notifications are not required to exist outside of the webpage.

No course assistance in lectures will be provided for this component, you should do your own research as to how to implement this. There are extensive resources online.

2.7. Milestone 7 - Very Challenge Components (`advanced *= 2`) (4.6%)

2.7.1. Static feed offline access

- Users can access the most recent feed they've loaded even without an internet connection.
- Cache information from the latest feed in local storage in case of outages.
- When the user tries to interact with the website at all in offline mode (e.g. comment, like) they should receive errors

No course assistance will be provided for this component, you should do your own research as to how to implement this.

2.7.2 Fragment based URL routing

Users can access different pages using URL fragments:

```
/#profile=1
/#feed
/#profile=4
```

No course assistance in lectures or on the forum will be provided for this component, you should do your own research as to how to implement this.

2.8. Bonus Marks (5%)

An extra 5% of the assignment can be attained via bonus marks, meaning a maximum mark of 105/100. Any bonus marks that extend your ass2 mark above 100% will bleed into other assignment marks, but cannot contribute outside of the 75% of the course that is allocated for assignment marks

Your bonus feature(s) can be anything. You just have to think of something that could make your web app stand out in some minor or major way. Simple examples would include just making sure that your user interface and user experience stands out amongst other students, maybe through some user testing.

You could also add extra features, such as some additional frontend form validations - the possibilities are limitless.

If you do implement a bonus feature, describe the feature and its details in `bonus.md` in the root directory of this repository.

3. Getting started

3.1. The Frontend

Stub code has been provided to help you get started in:

- `frontend/index.html`
- `frontend/styles/global.css`
- `frontend/src/helpers.js`
- `frontend/src/main.js`

You can modify or delete this stub code if you choose. It's simply here to potentially provide some help.

To work with your frontend code locally with the web server, you may have to run another web server to serve the frontend's static files.

To do this, run the following command once on your machine:

```
$ npm install --global http-server
```

Then whenever you want to start your server, run the following in your project's root folder:

```
$ npx http-server frontend -c 1 -p [port]
```

Where `[port]` is the port you want to run the server on (e.g. `8080`). Any number is fine.

This will start up a second HTTP server where if you navigate to `http://localhost:8000` (or whatever URL/port it provides) it will run your `index.html` without any CORs issues.

3.2. The Backend

You are prohibited from modifying the backend. No work needs to be done on the backend. It's provided to you simply to power your frontend.

The backend server exists in your individual repository. After you clone this repo, you must run `yarn install` in `backend` directory once.

To run the backend server, simply run `yarn start` in the `backend` directory. This will start the backend.

To view the API interface for the backend you can navigate to the base URL of the backend (e.g. `http://localhost:5005`). This will list all of the HTTP routes that you can interact with.

We have provided you with a very basic starting database containing two users and one public channel with messages. You can look in `backend/database.json` to see the contents.

Your backend is persistent in terms of data storage. That means the data will remain even after your express server process stops running. If you want to reset the data in the backend to the original starting state, you can run `yarn reset` in the backend directory. If you want to make a copy of the backend data (e.g. for a backup) then simply copy `database.json`. If you want to start with an empty database, you can run `yarn clear` in the backend directory.

Once the backend has started, you can view the API documentation by navigating to `http://localhost:[port]` in a web browser.

The port that the backend runs on (and that the frontend can use) is specified in `frontend/src/config.js`. You can change the port in this file. This file exists so that your frontend knows what port to use when talking to the backend.

Please note: If you manually update database.json you will need to restart your server.

Please note: You CANNOT modify the backend source code for bonus marks.

3.3. Taking the first steps

This is how we recommend you start the assignment:

1. Read the entire spec, including a thorough read of section 2 so you know what is ahead of you!
2. Try to load up the `index.html` on your browser with a simple "Hello world" text just to sanity check you know what page you're trying to load.
3. Plan out your UI by thinking about all of the key screens and what information they rely on
4. Try to load up the backend and verify you've got it working by making a simple API call to `/feed` (which should return you an empty list)
5. Good luck!

4. Constraints & Assumptions

4.1. Javascript

- You must implement this assignment in ES6-compliant Vanilla JavaScript. You cannot use ReactJS, JQuery, or other abstract frameworks. You can not, for example, use a popular Javascript framework such as Angular or React.
- You may **NOT** directly use external JavaScript. Do not use NPM except to install any other development libraries without prior approval from course authority.

4.2. CSS and other libraries

- You may use small amounts (< 10 lines) of general purpose code (not specific to the assignment) obtained from a site such as Stack Overflow or other publically available resources. You should clearly attribute the source of this code in a comment with it. You can not otherwise use code written by another person.
- You may include external CSS libraries in this assignment (with the `<link />` tag). You must attribute these sources (i.e. provide the URL/author in source code comments). For example, you are permitted to use the popular Bootstrap CSS framework. Some Bootstrap functionality relies on accompanying Javascript. You are permitted to include this Javascript. The Javascript accompanying Bootstrap requires the popular general purpose Javascrpt library jQuery. You are permitted to include **jQuery** so bootstrap can use it. However you are not permitted to use **jQuery** in the code you write for the assignment.

4.3. Browser Compatibility

You should ensure that your programs have been tested on one of the following two browsers:

- Locally, Google Chrome (various operating systems)
- On CSE machines, Chromium

4.4. Other Requirements

- The specification is intentionally vague to allow you to build frontend components however you think are visually appropriate. Their size, positioning, colour, layout, is in virtually all cases completely up to you. We require some basic criteria, but it's mainly dictating elements and behaviour.
- This is not a design assignment. You are expected to show common sense and critical thinking when it comes to basic user experience and visual layout, but you are not required to be creative to achieve full marks.
- Your web app must be a single page app. This means that there is only one initial browser load of content on one html page, and all subsequent dynamic changes to the page are based on Javascript DOM manipulation. If you do not build a single page app (e.g. using links to multiple HTML pages), you will receive a 50% penalty of your mark.

4.5. Static HTML, innerHTML, DOM manipulation

In this assignment, you are:

- Allowed to add static HTML/CSS to the stub website provided (i.e. you can put raw HTML/CSS as if it's a static page, even if you then later manipulate it with JavaScript).
- Allowed to build HTML elements and add CSS properties to the DOM via JavaScript. We expect this to be the most common way students build these pages.
- Are strictly **not** allowed to use the `innerHTML` property of nodes/tags to set the inner HTML of an element. This has security vulnerabilities and is in general not best practice. Either statically add the HTML/CSS and manipulate it with JavaScript, or generate and build nodes/elements in JavaScript (just like in lectures/tutes/labs), or both. But don't set inner HTML. The use of any `innerHTML` will result in a 50% penalty of your mark.

4.6. Async, Await, Promises

You are strictly **not** allowed to use the `async` and `await` syntax in this assignment. You must use ES6 Promises. The use of any `async` or `await` will result in a 50% penalty of your mark.

5. Marking Criteria

Your assignment will be hand-marked by tutor(s) in the course according to the criteria below.

Please note: When we test your UI we will use a pre-loaded database JSON that already has jobs and users and watches added to it.

Criteria	Weighting	Description
Compliance to task requirements	70%	• Each milestone specified a particular % of overall assignment (summing up to 70%). Implement those components as required to receive the marks. • You **MUST** update the `progress.csv` file in the root folder of this repository as you complete things partially or fully. The valid values are "NO", "PARTIAL", and "YES". Updating this is necessary so that your tutor knows what to focus on and what to avoid - giving them the best understanding of your work and provide you with marks you have earned.
Mobile Responsiveness	15%	• Your application is usable for desktop sizes generally, tablet sizes generally, and mobile sizes generally (down to 400px wide, 700px high).
Code Style	10%	• Your code is clean, well commented, with well-named variables, and is well laid out.
Usability & Accessibility	5%	• Your application is usable and easy to navigate. No obvious usability issues or confusing layouts/flows. • Your application follows standard accessibility guidelines, such as use of alt tags, and colours that aren't inaccessible. • Describe any attempts you've made to improve the usability/accessibility in `usability.md`
(Bonus Marks) Extra Features	5%	• Implementation of extra features that are not included in the spec. • Extra features should be non-trivial, have a clear justification for existing, and show either a form of technical, product, or creative flare. • Any extra features written down in `BONUS.md` in the project folder • Any bonus marks that extend your ass3 mark above 100% will bleed into other assignment marks, but cannot contribute outside of the 80% of the course that is allocated for assignment marks • **Expectations placed on solo groups will be half of that of pairs to achieve the same mark.**

6. Originality of Work

The work you submit must be your own work. Submission of work partially or completely derived from any other person or jointly written with any other person is not permitted.

The penalties for such an offence may include negative marks, automatic failure of the course and possibly other academic discipline. Assignment submissions will be examined both automatically and manually for such submissions.

Relevant scholarship authorities will be informed if students holding scholarships are involved in an incident of plagiarism or other misconduct.

Do not provide or show your assignment work to any other person — apart from the teaching staff of COMP6080.

If you knowingly provide or show your assignment work to another person for any reason, and work derived from it is submitted, you may be penalized, even if the work was submitted without your knowledge or consent. This may apply even if your work is submitted by a third party unknown to you.

Every time you make commits or pushes on this repository, you are acknowledging that the work you submit is your own work (as described above).

As per the course outline, "Pairs will be required to contribute regularly to gitlab and in reasonably equal contributions as we still assess contributions individually (there is no blanket group mark assigned). Failure to do so may result in a loss of marks."

If you are not working in a pair, we still strongly encourage you to commit small amounts of code regularly. This will assist you in managing any allegations of plagiarism that you feel are not correct.

Note you will not be penalized if your work has the potential to be taken without your consent or knowledge.

8. Submission

This assignment is due *Monday 27th of March, 10am*.

To submit your assignment, you must complete the following two steps in order:

- Ensure you've pushed all of your code to your gitlab master branch. You can check if you've done this properly by seeing what code is on the gitlab site on your master branch.

This will submit the latest commit on master as your submission.

You do NOT have to run a submit command for this assignment (we are experimenting with not using the command).

It is your responsibiltiy to ensure that your code can run successfully when cloned fresh from Gitlab.

For pairs, only one team member needs to submit.

8. Late Submission Policy

No late submission are accepted.