Lab 1: Basic web pages: How much do you know?

Motivation

It has been assumed that you have some knowledge of:

- GitHub
- HTML
- CSS
- JavaScript
- Front- and back- end development
- Node-based server-side code (or some other server code)

The first lab ensures this (and also makes sure everything is working as expected). You can complete Task 1 and Task 2 out of order if you prefer to work on your back end first.

Task 1: A stylish front end

On Moodle, you will find an .html file. It's a "website", but it's barely functional and completely unstyled. You should be able to fix this.

You can use Visual Studio Code for editing code and Google Chrome for viewing it. Other IDEs and browsers are available at your discretion. You should just be able to double click on the .html file and it will open in the default browser. **THIS IS NOT SERVING IT.**

For GitHub, select the way of interacting with it that you find most comfortable. That might be via CLI or through your IDE, or you might use the web interface. You must use small, atomic commits.

Your frontend tasks:

1. Create a GitHub repository, give it a ReadMe.md. Put the html file in there as a first commit. Pay attention to folder structure in your project. It's ok to have it in the top level for now but you might want it in a sub-folder eventually.
2. Style the website using CSS (and HTML if you want to change it). It must be **styled consistently** but it does not need to be beautiful. You can use Bootstrap if you like. Commit everything to GitHub when you're happy, update your ReadMe if you need to.
3. Add a picture somewhere (see below). The picture should go in your GitHub, too.
4. Consider the JavaScript implementation – do you want to refactor the code? Do you want your JavaScript in with your HTML like that? Change it if you prefer. Commit to GitHub when you're happy.
5. Add functionality (JavaScript) to round the overall price to the nearest £50 or £100. You could also change the functionality so that there is a default salary option (find out the average salaries for web developers if you like). Commit to GitHub when you're happy, update your ReadMe if you need to.

Task 2: A functional back end

You now have a website (or a single page application). Time to serve it up (locally) using node.js as a server.
Using node.js and express (and LLMs to help if you wish), write a server to serve your (static) web page. It should run on port 8080 or on port 8000. The main file could be index.js (but it's ok if you give it a different name like main.js or server.js).

You will have a package.json file, which should very nearly write itself as you work (using *npm init* and *npm install* with the *-s* option). Add all the necessary things to GitHub as you go. Realistically, you will need express (npm package).

Do not add node_modules to your GitHub. These are covered by npm and package.json.

You can run node.js (and npm) via PowerShell or via a terminal on Visual Studio Code.

You should add some sort of response for a 404 error to your server.

Task 3: Shifting some functionality around

Open up the developer tools in your browser and see how much of the functionality of your website is completely visible to anybody. You might not want this. **Think about why**. Find a way to shift this functionality to the back-end. You can put almost all the JavaScript from the front end into the back end.
Note: You may want ajax on the front end to take the inputs and turn them into a url and call and respond to it from your page. You will want api routes on the back end that the front end can call (they go in before the static express code).

At the end

You should have a GitHub with a functioning web system which includes a front page and a back end api. You should be able to spin up your server using node and you should be able to access your web page using a browser.

Aaaaaah, it's the Windows Firewall!

You can run npm and node from a Visual Studio Code Terminal (View->Terminal – it is actually PowerShell). But, RGU is very security conscious, so when you do that, Windows Firewall is going to ask you if you want to make changes to the local device. You don't have access to admin accounts on the RGU computers, so you will have to cancel it away. It will still work on **localhost** though, so you can still see your website on a browser on the computer.

The thing that will not work (because of the firewall) is that you will not be able to access your website from another machine. We will cover this next week when we start using Linux virtual machines.

Additional Materials

If you know how to do stuff, that's great. If you're keen to utilise (or just try) ChatGPT or Co-pilot, that's also great – but remember that if they generate the code, you should use them to do some of the debugging, too. If you want some advice, look here.

One possible answer

If you want to see one possible answer and how it was constructed, look here:
https://github.com/PamJohnston/PriceCalculator
This is also available on Moodle.

Git on the command line

You do not have to use git on the command line, but if you master git on the command line, you will have access to the full git functionality and you will not have to figure out how to integrate it into every new IDE you learn. https://docs.gitlab.com/ee/gitlab-basics/start-using-git.html This gives you the instructions.

How Pam uses GitHub
Usually, I create my new repository on GitHub's web interface, then use:
git clone <url of the github>
to pull down the file structure.

Add files/folders individually using:
git add <file name>

Check what you've updated with:
git status

Commit (frequently) using:
git commit -a -m "This is my short but detailed comment"

Push to the remote repository using:
git push

To push to your remote repository, you will need to be logged in. You can use gh to login – it's GitHub's new CLI. Otherwise, you will also need to set up an access token (if you haven't already). To generate a **personal access token** via the GitHub web pages, you need to go to Account Settings, then to Developer Settings and then to personal access tokens. Follow the instructions to generate a personal access token. It's probably best to use the "fine-grained" one and set it up to access only this repository. You only need repo access for now.

When it comes time to push, it will want your git user name (email address) and password (ahem, **personal access token**), so have those ready.
This tells you how to set it up so you don't need your login credentials very often:
https://www.edgoad.com/2021/02/using-personal-access-tokens-with-git-and-github.html
This also talks about caching your github credentials: https://docs.github.com/en/get-started/getting-started-with-git/caching-your-github-credentials-in-git

Scriddie's answer (not the top answer) here:
https://stackoverflow.com/questions/18935539/authenticate-with-github-using-a-token
And this medium article: https://pythononpow.medium.com/howto-use-github-tokens-on-the-command-line-995422c06f00 tells you to add your personal access token to the github remote settings for a particular local repository. **Use this one with care!!!!** npm init takes the default github name from this particular setting and puts it into packages.json. If you then commit your packages.json file to GitHub, then your token can end up in a public repository. This is bad because GitHub immediately knows and revokes your personal access token.

Styling the front page

There is nothing wrong with Bootstrap:
https://www.w3schools.com/bootstrap/bootstrap_get_started.asp
There is also nothing wrong with leaving full styling until you've got it running on a server. At the very least, add some bootstrap classes to your html so that it's ready.

A simple back-end

You could use express.static to simply serve up a folder. There's an example here:
https://github.com/rsp/node-express-static-example