DIT Academic Exposure

1 Setting up Fullstack Web Application

1.1 Download and Setup Git

1. Go to https://git-scm.com/downloads and download the relevant version.

2. Follow the installation instruction (When asked set Visual Studio Code as default editor).

3. Configure your Git name and email by executing the following 2 commands (One at a time):

```
git config --global user.name "YOUR_NAME"
git config --global user.email "YOUR_GITHUB_EMAIL@example.com"
```

1.2 Clone repository

1. Run the following commands to clone the code onto your computer

```
cd Downloads
git clone https://github.com/ST0507-ades/acad-exposure
```

2. Open the cloned folder `acad-exposure` in your VSCode (For windows):

```
code acad-exposure
```

> MacBook users can simply use the "Open Folder" feature of VSCode

1.3 Setup .env

1. Create a new file `.env` on the root of the project with the following content

   ```
   DATABASE_URL=XXXX
   ```

   > We will replace `XXXX` with something else later.

1.4 Install dependencies

1. Open a new terminal (`CTRL/CMD +``)

2. Run the following command

   ```
   npm install
   ```

1.5 Setup & Populate Database

1. Go to https://www.elephantsql.com

2. Sign in with your GitHub account

3. Create a new instance

   1. Name: Anything you like
   2. Plan: Tiny Turtle (free)
   3. Tags: Leave blank
   4. Data Center: Any Tokyo or HongKong server is fine.
   5. Create instance

4. Go to the `Details` page of your database

5. Click on the `copy` icon beside the `URL` field (Note: Do not ctrl+c the text you see there, the password has been obscured with `****`).

6. In your `.env` file, replace `XXXX` with the `URL`. E.g.,

   ```
   DATABASE_URL=postgres://abcdef:abcdefghxyz12345@abc.db.elephantsql.com/abcde
   f
   ```

7. In the VSCode terminal, run the following command:

   ```
   npm run init-db
   ```

1.6 Running the server

1. Run the following command:

   ```
   npm start
   ```

1.7 Viewing the website on a browser

1. Go to the browser and enter `http://localhost:3000`

1.8 Testing the application

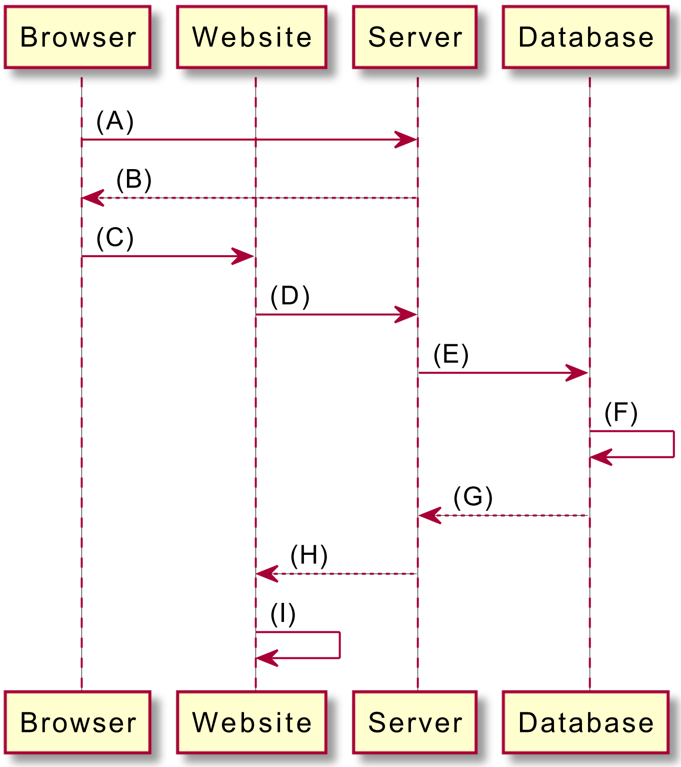
1. You should see a loading animation and subsequently some data being displayed

---

2 Understanding the components

1. Explain what happens when you run (e.g. what files are involved, what scripts are executed, etc...)
   1. `npm install` - How can you know what will be installed?
   2. `npm run setup-db` - Which file is executed? What is in the file? What does each part of the file do?
   3. `npm start` - While file is executed? Can you do it without running `npm start`?
2. How does the browser get the website?
   1. What files does the browser need to display the website?
   2. Where does the browser get these files from?
   3. How does it request for those files?
3. How does the website get the data?
   1. Does the HTML page come with the data?
   2. When does the page get the data?
   3. In what format is this data?
   4. What shape is this data in?
   5. Is the getting of data instantaneous?
   6. Does the website freeze while waiting for the data?
4. How does the server provide the data?
   1. Where in the server is the starting point of the processing of the request for all the data?
   2. Where does the server get the data from?
   3. How does the server request for the data?
   4. What does the server do after receiving the data?
   5. Is the getting of data instantaneous?
   6. How many request can the server process at a single point of time?
5. The entire process can be described as several steps. The following steps are in randomized order. Fill in the diagram with the correct steps.

1. Website process and display the
2. Server returns Website to Browser
3. Server returns data to Website
4. Server request for data from Database
5. Website request for data from Server
6. Browser displays the Website
7. Database returns data to Server
8. Database process request
9. Browser request for Website from Server



~~~

---

3 Making changes

We will modify the page such that when it scrolls to the bottom, the next page worth of data will be loaded.

1. Find the function in the Website that is executed when the page is scrolled to the bottom.

    ```js
    window.onscroll = function () {
        const isEndOfpage = ...
        if (isInitialFetched && isEndOfPage) {
            if (isFetching) return;
            isFetching = true;
            loadingSkeleton.classList.add('loading');
            ...
        }
~~~

~~~
    };
    ```

2. Add the following code in the function, in the part where it is executed only
when you've scrolled to the bottom (Which steps, A to I, does this code cover?)

    ```js
    fetch(`/data?offset=${offset}`) // Q1: Which step is this?
        .then(function (response) {
            // Q2: Which step is this?
            return response.json();
        })
        .then(function (json) {
            // Q3: Which step is this?
            offset = json.offset;
            data = json.data;
            for (let i = 0; i < data.length; i++) {
                const item = data[i];
                addItem(item);
            }
        })
        .catch(function (error) {
            alert(error.message);
        })
        .finally(() => {
            isFetching = false;
            loadingSkeleton.classList.remove('loading');
        });
    ```

3. Observe the website. What's the problem?
4. Find the function in the Server that handles that request (`GET /data`)

    ```js
    app.get('/data', (...) => {
        ...
    })
    ```

5. Modify that function accordingly (What is this change doing?)

    ```js
    app.get('/data', (req, res, next) => {
        const offset = req.query.offset || 0;
        ...
        const sql = `SELECT * FROM ..... LIMIT 10 OFFSET $1`
        const params = [offset];
        ...
            return res.json({
                offset: +offset + rows.length, // Why +offset?
                ...
            })
    })
    ```
~~~

```
6. Reset the server, observe the website.
```