CSE5WDC Assignment

You have been hired by a client to develop a new web service, “Neverwrote”, for managing virtual notes - “Never say 'I never wrote that down' ever again!”. Having heard of the latest and greatest in web technologies, your client specifies that the backend shall run on a Node.js server, and that the frontend shall be built with React. “That’s strange, those are exactly the same things that I learnt in CSE5WDC” you think to yourself as you boot up your computer and get ready to start work.

Plagiarism

This is assignment must be completed individually, not in groups. You are welcome to discuss problems with fellow students, but the work that you submit must be your own. Don't try to plagiarise, it's extremely obvious and will only serve to get you into trouble. That's not any fun for you or for us.

Due Date

This assignment is due on Wednesday, 21 October 2020, 5:00 PM. Penalties are applied to late assignments (accepted up to 5 days after the due date only). See the departmental Student Handbook for details.

Submission Details

- Zip up everything in your project directory into one file.
- Submit your zipped file via LMS

Your final submission must run in a clean virtual machine after invoking `docker-compose up`, as this is the environment that we will be using to mark your assignment.

Demo Implementation

I will show the demo again in the next lecture to help you understand about what an implementation of Neverwrote might look like. However, be aware that everyone in the class will be able to see what you write, so be sensible and don't put up personal information or inappropriate content.

Objectives

- To combine what you've learnt in labs into a complete web application
- To provide you with a reference web application for future projects

Getting started

There is a template for the assignment project available for you on LMS at https://lms.latrobe.edu.au/mod/resource/view.php?id=4329541. Download this zip file and use it as a starting point for your assignment. Be sure to commit and push regularly to avoid losing any of your work as you develop.

You will find more specific information about the project structure in the README.md file.

Part 1 – Backend API (50 marks)

Task 1.1 (50 marks)

Create the backend database and REST API for the Neverwrote application.

Specifications

The database shall have two tables, one for notebooks called **Notebooks** and one for notes called **Notes**.

- Records from the **Notebooks** table shall contain the attribute **title** (string).
- Records from the **Notes** table shall contain the attributes **title** (string) and **content** (text).
- All tables have a primary key column called **id**.
- Foreign keys shall follow the camel-case naming convention (eg a foreign key to the Notebooks table would be called notebookId). Specify the foreign key name explicitly in associations, as in Lab 07 (there we did foreignKey: 'postId').
- Each note shall belong to one notebook, but a single notebook can have many notes.
- Two Sequelize models shall be defined – **Notebook** and **Note**. These models shall have the correct associations.
- The backend shall expose the following REST API:

HTTP action	**Description**
GET /notebooks	Returns a list of all notebooks (does not include notes). <u>This endpoint has been created for you already.</u>
GET /notebooks/:notebookId/notes	Returns a list of all notes for a particular notebook.
POST /notebooks	Creates a new notebook using the posted data. Returns the new notebook.
GET /notebooks/:notebookId	Returns a single notebook by ID.
DELETE /notebooks/:notebookId	Deletes a single notebook by ID. All of the notebook's notes shall be deleted also. Returns an empty object, {}.
PUT /notebooks/:notebookId	Updates the attributes of a particular notebook. Returns the updated notebook.
GET /notes	Returns a list of all notes.
POST /notes	Creates a new note using the posted data. The notebookId attribute shall specify which notebook it belongs to. Returns the new note.
GET /notes/:noteId	Returns a single note by ID.
DELETE /notes/:noteId	Deletes a single note by ID. Returns an empty object, {}.
PUT /notes/:noteId	Updates the attributes of a particular note. Returns the updated note.

- The tests shall pass. This is a very helpful tool, as you can verify that you are on track as you complete this part of the assignment. You can run the tests with:

```
docker-compose run --rm api jasmine
```

    - You shall not modify the tests in the spec/ directory
- The API shall access the database (and accept values from the HTTP request where appropriate). The REST API shall not use hard-coded values for notes or notebooks.

Part 2 – Frontend interface (40 marks)

You will find “Lab 08 – Frontend” to be particularly useful as a reference for this part. I have also prepared a stripped-back React/Redux code example which you may find useful to look at for ideas:

https://jsfiddle.net/dismal_denizen/zq23cheL/.

Don’t forget that you need to have Gulp running (docker-compose run --rm frontend gulp) for your frontend code to be compiled as you work.

Task 2.1 (18 marks)

Create a read-only web interface for the Neverwrote application which displays notebooks and notes.

Specifications

- There shall be a view which lists the titles of all of the notebooks.
- Clicking on a notebook shall display a list of all of the titles of notes within that notebook.
- Clicking on a note shall show the full content of that note.

Task 2.2 (20 marks)

Allow users to create and delete notes and notebooks via the web interface.

Specifications

- Users shall be able to create and delete notes and notebooks.
- The state of the application shall be maintained in a Redux store.
- All changes to the Redux store shall be made via dispatched actions.
- Modifications to notebooks and notes shall be persisted in the backend database via the REST API. Therefore changes should survive a page refresh.
- You shall use the MarkdownEditor component provided for you as the way to edit notes. This is not much different from using a normal text field. Example usage: <MarkdownEditor value={this.state.noteContent} onChange={onChangeContent} />
- You shall use two Redux reducers: one for the notebooks and one for the notes.

Task 2.3 (2 marks)

Make the interface pretty and nice to use. The Internet is a great place to find inspiration for styles and small snippets of CSS which you can incorporate into your design.

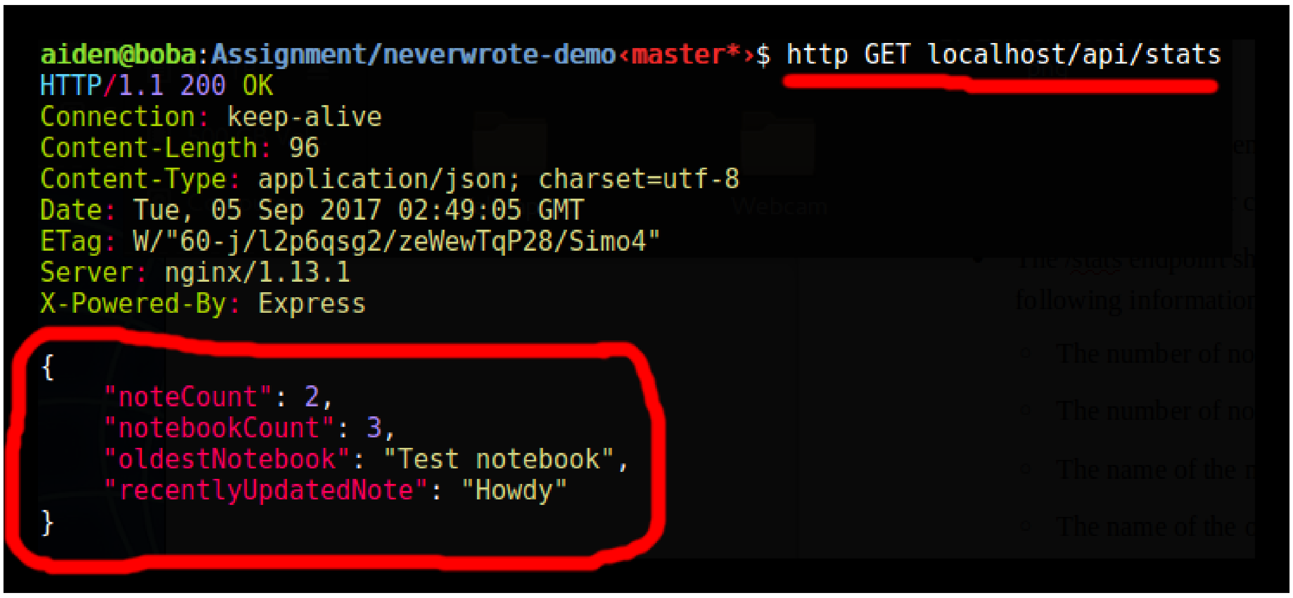
Bootstrap v3.3.6 is already included as a dependency of the project, and defines a bunch of nice-looking CSS classes which you can use out-of-the-box. Check out the documentation at http://bootstrapdocs.com/v3.3.6/docs/ to see what's available (in particular the CSS and Components pages).

Part 3 – Summary statistics (10 marks)

Task 3.1 (4 marks)

Create a new controller in the backend and set up the routes so that the controller is responsible for the /stats URL path. Add a single action to the controller which responds with JSON indicating some key statistics about the notebooks and notes stored in the database. Use your Sequelize models to perform the necessary database queries.

Here is an example of what the API response might look like:



```
aiden@boba:Assignment/neverwrote-demo‹master*›$ http GET localhost/api/stats
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 96
Content-Type: application/json; charset=utf-8
Date: Tue, 05 Sep 2017 02:49:05 GMT
ETag: W/"60-j/l2p6qsg2/zeWewTqP28/Simo4"
Server: nginx/1.13.1
X-Powered-By: Express

{
    "noteCount": 2,
    "notebookCount": 3,
    "oldestNotebook": "Test notebook",
    "recentlyUpdatedNote": "Howdy"
}
```

Specifications

- The new /stats API endpoint shall be made available at http://localhost/api/stats.
- The stats controller code shall be placed in a new file, src/controllers/stats.js.
- The /stats endpoint shall respond to GET requests with a JSON object containing the following information:
    - The number of notebooks in the database.
    - The number of notes in the database.
    - The name of the most recently updated note.
    - The name of the oldest notebook by creation time.

Task 3.2 (4 marks)

Modify the frontend to display summary statistics on the Neverwrote home page. The statistics themselves shall be retrieved using the /stats API endpoint implemented in Task 3.1.

Specifications

- A new reducer shall be added for managing statistics in the frontend application store.
- A new "Statistics" React component shall be created which is connected to the application store and displays the summary statistics.
- The Statistics component shall be displayed on the Neverwrote home page.
- Stats shall be loaded into the application store the first time that the Statistics component is initialised.
    - It is **not required** that the statistics update as notes/notebooks are added, removed, or modified.

Task 3.3 (2 marks)

Add a refresh button which updates the displayed statistics when clicked.

For example, if the statistics show a count of 3 notebooks, and then you add another one, clicking the refresh button should update the displayed statistics to indicate that there are now 4 notebooks.

Specifications

- A refresh button shall be visible in the statistics component.
- Clicking the refresh button shall update the stats by accessing the /stats API endpoint.

Part 3 – Bonus task (up to 20 bonus marks)

These bonus tasks are an opportunity for you to earn back marks which you may lose in the other parts of the assignment.

Implement a search system for the frontend interface.

- 10 marks will be awarded for search functionality within a notebook (filtering based on title and content). This can be purely client-side (in the browser).
- The rest of the marks will be awarded for a separate ability to search across all notes and notebooks based on title and content. This shall be implemented via a new API endpoint (so the searching happens in the backend).