UNIVERSITY OF PRETORIA
Department of Information Science
Multimedia

IMY 220: Advanced Markup Languages II
Examination

Examiners:		25 November 2022
Internal	Mr ID Bosman	Total marks: 48
		Duration: 24 hours
External	Mr YL Wong	

This exam will run from 12:00 on 25 November to 12:00 on 26 November. Download the test files which contain the folder structure you must adhere to when submitting your files from ClickUP. You will need to submit a zip file with all your code in the relevant folders on ClickUP.

Please include the following declaration statement in your final submission before answering any questions:

The University of Pretoria commits itself to producing academic work of integrity. I affirm that I am aware of and have read the Rules and Policies of the University, more specifically the Disciplinary Procedure and the Tests and Examinations Rules, which prohibit any unethical, dishonest or improper conduct during tests, assignments, examinations and/or any other forms of assessment. I am aware that no student or any other person may assist or attempt to assist another student, or obtain help, or attempt to obtain help from another student or any other person during tests, assessments, assignments, examinations and/or any other forms of assessment.

Along with your final submission, include a signed copy of the plagiarism form "declaration of originality" found alongside this examination.

Section 1: HTML, PHP, & MySQLi **[10]**

Use only the files provided in the "S1" folder to complete this section.

Complete the following questions in index.php.

1.1 Add the correct attributes in the form so that it meets the following requirements: (2)

a. When the form is submitted, the form data should be sent to *results.php*.
b. The data being sent through must be visible in the URL.

For the purposes of the current section, entering data and submitting the form constitutes a login.

Complete the following questions in <u>results.php</u>.

1.2 If data has been sent, as per question 1.1, save all the data to appropriately named session variables. (1)

1.3 As long as the user has logged in with the form in *index.php* and has not logged out (which is described in question 1.4), the data with which they have logged in must be displayed inside a paragraph element. In other words, after a user has entered their details and submitted the form in *index.php* as well as if they visit the page by entering only the URL (without the appended data) after having logged in, their data must be displayed, which must be in the following format: (3)

The following details have been entered:
First name: Morty
Last name: Smith
Email: morty.smith@gmail.com
Birthday: 2000-02-11

1.4 The page must also display a button underneath the user details, which will "log out" a user. Clicking this button must redirect them to *index.php* and clear the session variables set for question 1.2. You will need to use a form to accomplish this and may use additional form elements as long as they do not affect the visual result of the webpage. (2)

1.5 If a user is not logged in, in other words, either they have not yet entered and submitted form data in *index.php* or they have logged out from *results.php*, the results *results.php* page must only display a level-1 heading with an appropriate message and nothing else, for example: (2)

You are not logged in

Section 2: **Bootstrap and jQuery** [20]

Use only the files provided in the "S2" folder to complete this section.

This section requires you to write functionality using jQuery. You must, wherever appropriate, make use of the following ES6 syntax:

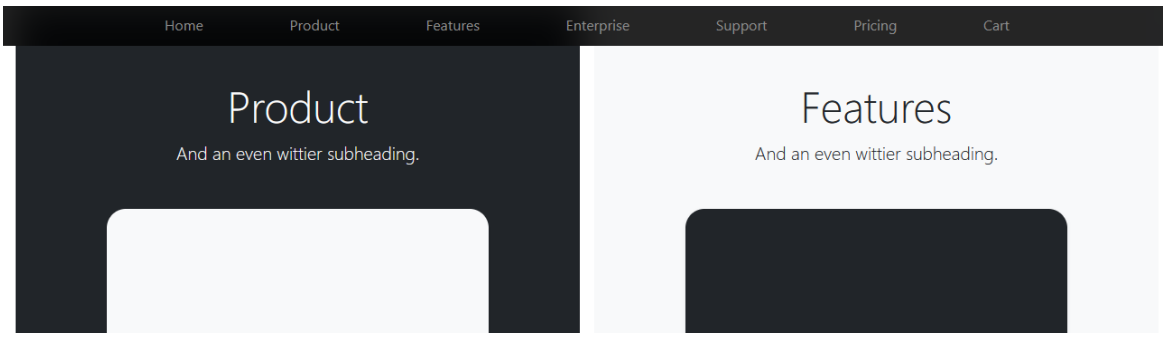
- Arrow functions
- Appropriate variable declarations

All functions must be written as function expressions. Where appropriate, you must use the recommended jQuery approach for adding event handlers.

Download the file *index.html*. This file contains the basic contents of a webpage (slightly modified from: https://getbootstrap.com/docs/5.2/examples/product/). You will need to add your own CSS and JS in the files provided. You will also need to add your own HTML, but you may not remove existing HTML from this file.

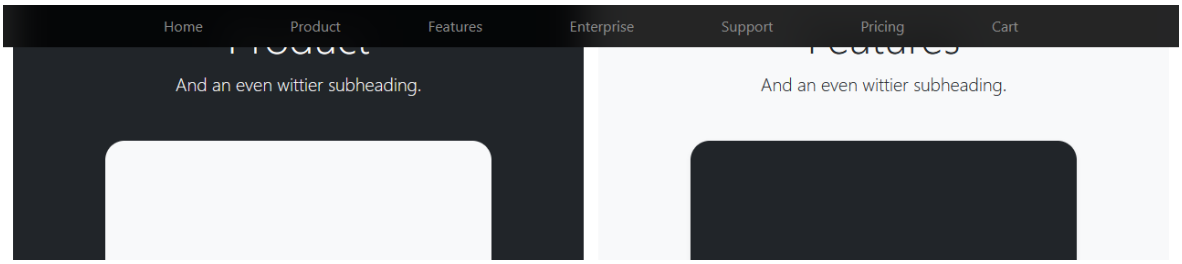
The solution for this section is provided in the file *indexComplete.html*. Do not attempt to use the code from this file to create your own solution. If your JS code looks anything like the code in this file, your solution for Section 2 will not be marked.

2.1 Create functionality for scrolling the page to the appropriate heading when clicking on links in the navigation bar. You must use the Bootstrap Scrollspy functionality for this. For example, when clicking on Product in the navigation bar, the page should scroll to the Product heading as follows: (3)

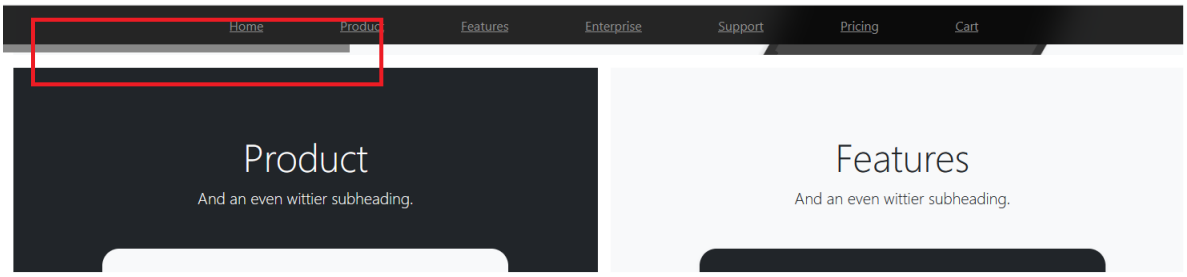


2.2 The scrolling should happen over time (not instantaneously jump to the heading on the page). (2)

2.3 The page should scroll so that the heading is still visible and is not cut off by the navigation bar, e.g., the following would be incorrect: (3)



2.4 Create a "progress bar" that indicates the scrolling position on the page. In other words, as the user scrolls down, the progress bar should fill more of the width of the page and should be exactly the width of the page (not shorter, not longer) when the user has scrolled all the way to the bottom. You will need to write new HTML, CSS, and jQuery code for this to work. (5)



(Refer to *indexComplete.html* for a demonstration of this functionality.)

2.5 The progress bar should always show the correct scroll amount. In other words, the user should never have to perform any additional actions (like refreshing the page) for the progress bar to be correct. (4)

2.6 The progress bar should also stay correct if the user resizes the page. (3)

Section 3: NodeJS, ReactJS, and MongoDB [18]

Use only the files provided in the "S3" folder to complete this section.

This section requires you to write functionality in JavaScript. You must, wherever appropriate, make use of the following ES6 syntax:

- Arrow functions
- Appropriate variable declarations
- Template strings
- Object destructuring

All functions must be written as function expressions. Furthermore, you are not allowed to use any loops, *forEach* function, or *$.each* function to implement array-based functionality. You must make use of the other JS array functions as discussed in class. Array-based functionality must work for any number of array items.

Install all the required modules required to be able to transpile and bundle JSX into valid JS and serve a webpage that renders JSX components. You must use the recommended directory structure and core file names (server.js, index.js, bundle.js, etc.) as discussed in the notes and serve the webpage to port 3000. All React components must be declared inside their own files; naming rules apply. Do **not** include your node_modules folder in your final submission.

You will use the data found inside *academicRecord.json* for the core functionality of this section.

3.1 Create a new collection in your own MongoDB database named academicRecord. Insert a new document inside this collection with the data from *academicRecord.json*.
Note: The content of this file is not valid JSON, you will have to modify it to make it valid before adding it as a new MongoDB document.

3.2 Define functionality that reads the contents of this document inside your Node server code. Use a projection to only retrieve the *name*, *surname*, and *results*. Use *socket.io* to send the data to be used inside your React components. (4)

3.3 Define a React component called *AcademicRecord* which takes an object as its prop (which will always be of the same format as that found inside *academicRecord.json*). The component must display the name and surname of the student in a level-1 heading and display the following in a Bootstrap grid: (2)

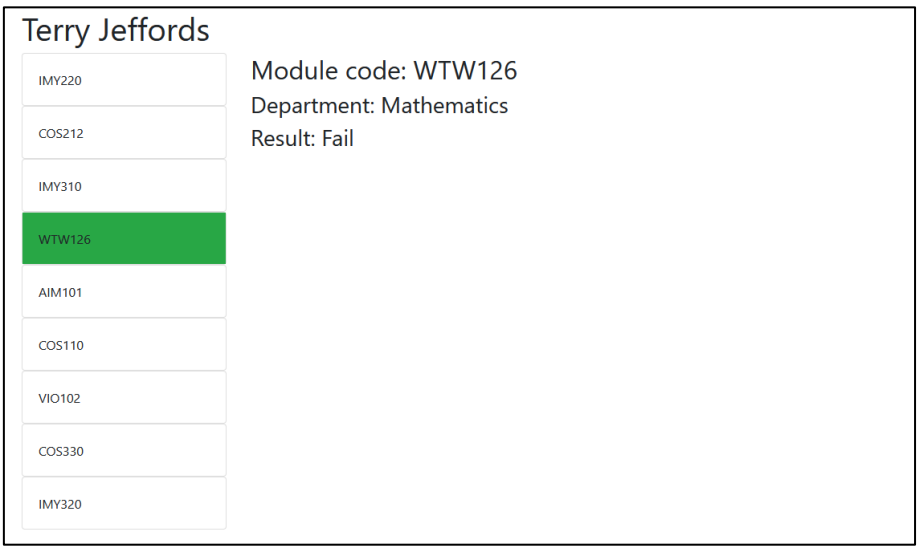
- A list of *ModuleCard* components (see 3.4) that take up ¼ of the container width. You will need to loop through the array of modules to create this.

- A *ModuleInformation (see 3.5)* component that takes up ¾ of the container width.

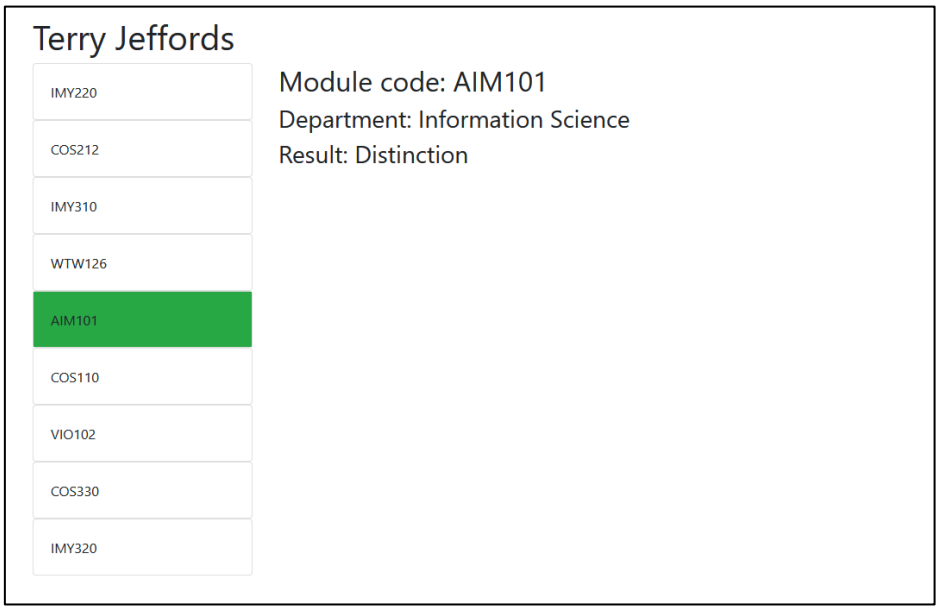
3.4 Define a React component called *ModuleCard* which renders a Bootstrap card containing the module code for a single module. For the purpose of rendering the card, you must only pass the module code as the component's prop (although you will need to pass other props as well). (2)

3.5 Define a React component called *ModuleInformation* which receives a JS object containing a module's information (as found in *index.js) and renders its information. Instead of giving the mark, however, you need to state whether the grade is a distinction (≥75), pass (≥50 and <75) or fail (<50).* (3)

3.6 Add functionality inside the components so that when one of the module cards is clicked on, its information is shown on the right. Also use Bootstrap's background-colour classes to highlight the selected module, i.e. the one that was last clicked on. (5)
For example, when clicking on WTW126:

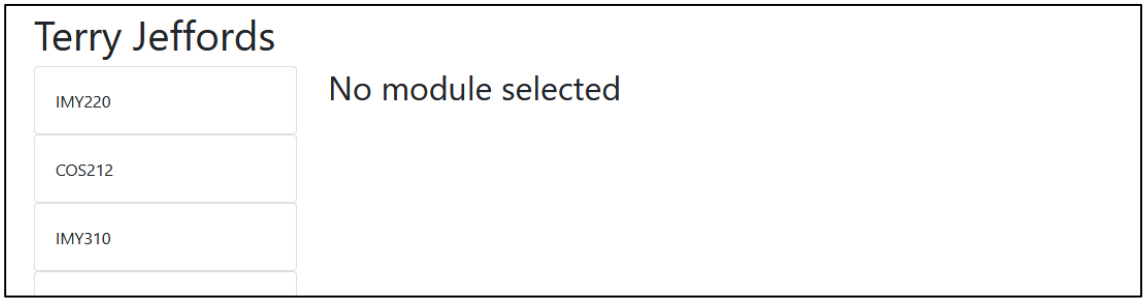


And then on AIM101:



To do this, you will need to add an event handler inside the *ModuleCard* component and you will need to keep track of the currently-selected module inside the state of *AcademicRecord* and pass this as a prop to other components.

3.7 When the page loads and no module is selected, an appropriate message should be displayed: (2)



The page must always be error-free, although you may ignore this error:

```
GET http://localhost:3000/favicon.ico
```

Warning: Do **not** include your node_modules folder in your final submission.
Note: wherever you are unable to provide the required functionality in the specified manner, such as not loading the data from MongoDB, it is recommended that you focus on still providing the required functionality. You will receive more marks for working code that does not meet the specifications than for code that does not work at all.