UNIVERSITEIT VAN PRETORIA
Department of Information Science

IMY 220
Advanced Markup Languages II
Examination

Examiner:		27 November 2020
Internal:	Mr ID Bosman	Marks: 48
		Time: 24 hours
External:	Mr YL Wong	

This exam will run from 08:00 on 27 November to 08:00 on 28 November. Download the test files from ClickUP, which contain the folder structure you must adhere to when submitting your files. You will need to submit a zip file with all your code in the relevant folders on ClickUP.

NB: download and test your zip file after uploading. If the file is corrupted or contains the wrong files, I cannot accept a re-submission.

Rename the "position_surname" folder to your position and surname. Include the following statement as a text file inside the renamed "position_surname" folder of your submission (with *name and surname* replaced with your name and surname):
The University of Pretoria commits itself to produce academic work of integrity. I, [name and surname], 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 otherforms of assessment. I am aware that no student or any other person may assist or attemptto assist another student, or obtain help, or attempt to obtain help from another student orany other person during tests, assessments, assignments, examinations and/or any otherforms of assessment.

Note: wherever you are unable to provide the required functionality in the specified manner, such as not using loops, it is recommended that you focus on providing the functionality. You will receive more marks for working code that does not meet the specifications than for code that does not work at all.

Section 1 **[10]**

HTML & PHP

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 results.php.

1.2 If data has been sent, as per question 1.1, save all the data to appropriately named session variables. (2)

1.3 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)

1.4 As long as the user has logged in with the form in *index.php* and has not logged out (which is described in question 1.3), 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

1.5 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*, *results.php* must only display a level-1 heading with an appropriate message and nothing else, for example: (2)

Section 2 [20]

JavaScript, ES6, jQuery, AJAX

Use only the files provided in the "S2" 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(s), or *$.each* function(s) 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.

Complete the following questions in <u>script.js</u>.

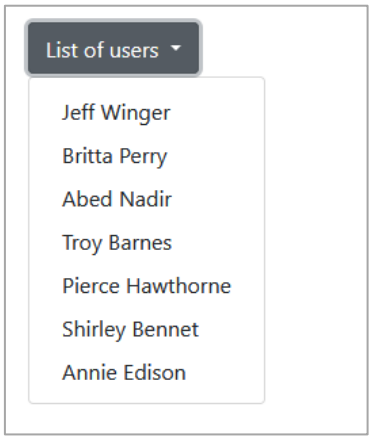
2.1 Write a function called *getUserDetails* which receives two parameters: the name of a file and an array of user IDs. The function will be used to read a JSON file and return an array of user details from this file (you may assume that the format will always be similar to that found in *users.json*). If an array of user IDs is supplied in the function parameters, it must only return the details for the users whose IDs match those given. If no array of user IDs are supplied to the function, it must simply return the details for all users listed in the file name given. You must use a default parameter to accomplish this. (4)

2.2 Write a function called *getFriendList* which receives two parameters: the name of a file and a single user ID. The function will be used to read a JSON file and return an array containing the list of user IDs that correspond with that of the given user (you may assume that the format will always be similar to that found in *friends.json*). (4)

2.3 Write a function called *createUserRow* which receives a JS object containing the name and surname of a user and returns a table row with those details. You must use the syntax recommended in the notes for creating jQuery elements. (3)

2.4 Using these three functions, create the following functionality: (3)

a) When the page loads, populate the dropdown list with the names of all the users in *users.json*. (You do not need to call an additional function for creating the HTML elements.) This must look as follows (when clicked on):



b) You must also use a data-attribute to save the user ID of each user listed in the dropdown list.

2.5 When clicking on any of the dropdown list items, their list of friends (as found in *friends.json*) must be displayed in a table. This must work as follows: (4)

a) The user ID of the user must be saved to a variable.

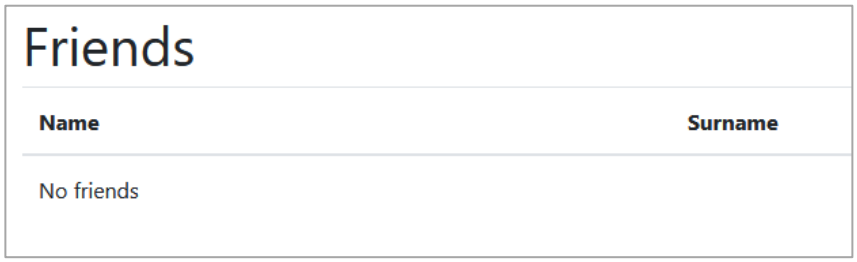
b) You must use the *getFriendList* and *getUserDetails* functions to get these details. Note that this will require you to use sequential/chained asynchronous calls since you must use the results of one function when calling the other. You may not use async/await functionality to implement this and you may not manage the array data in this function call (the correct array must be returned from *getUserDetails*).

c) Use the *createUserRow* function to create and return the table rows and then append them to the table. For example, when clicking on the second name in the list, the results must be as follows:

Friends

Name	Surname
Abed	Nadir
Troy	Barnes
Shirley	Bennet
Annie	Edison

d) If a user has no friends in their friends array in *friends.json,* only a row must be added with an appropriate message, for example: (1)



e) If a user has themselves in their friend list, their name should not be displayed. (1)

f) Only the details for the last user clicked on must be shown at any given time.

Section 3 [18]

NodeJS + ReactJS

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(s), or *$.each* function(s) 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 for being 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.

You will use the data found inside *index.js* for the core functionality of this section. You can use this variable declaration to pass as a prop to the *AcademicRecord* component but you may not alter this JS object in any way. You will also be required to add to this file to render the necessary components.

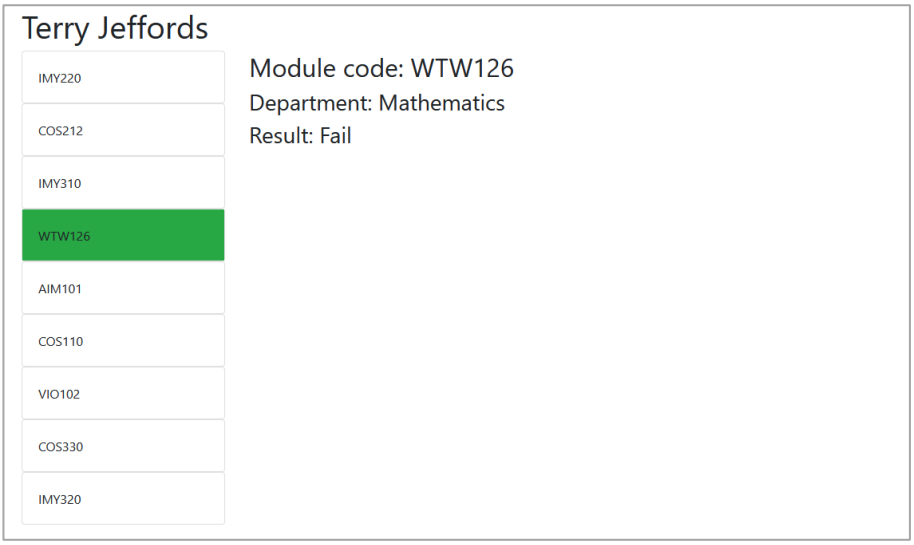
3.1 Define a React component called *AcademicRecord* which takes an object as its prop (which will always be of the same format as that found inside *index.js*). The component must display the name and surname of the student in a level-1 heading and display the following in a Bootstrap grid: (4)

- A list of *ModuleCard* components that take up ¼ of the container width. You will need to loop through the array of modules to create this.
- A *ModuleInformation* component that takes up ¾ of the container width.

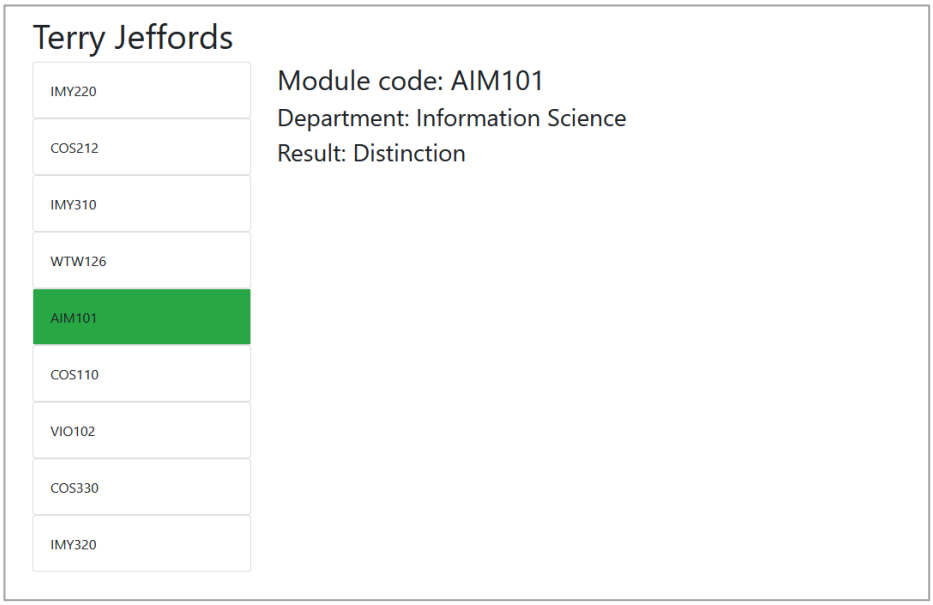
3.2 Define a React component called *ModuleCard* which renders a Bootstrap card containing the module code for a single module. For the purpose of rendering the card, you must only pass the module code as the component's prop (although you will need to pass other props as well). (4)

3.3 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). (4)

3.4 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-color classes to highlight the selected module, i.e. the one that was last clicked on. For example, when clicking on WTW126 (4)

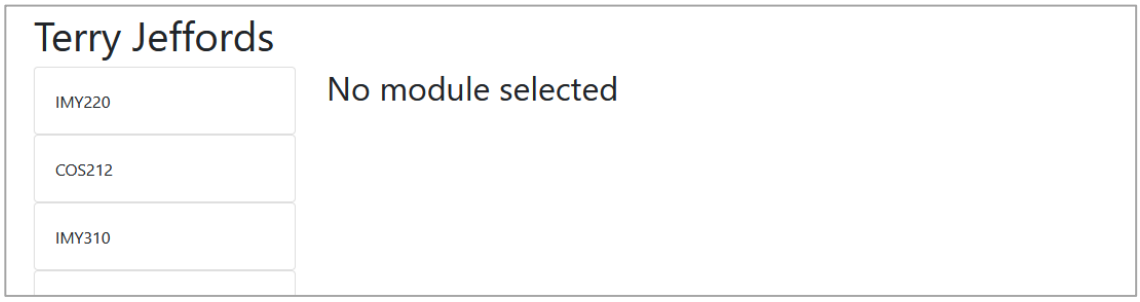


And then on AIM101



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.

3.5 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
```