University of Maryland College Park
Department of Computer Science
CMSC335 Spring 2023
Exam #3

FIRSTNAME LASTNAME (PRINT IN UPPERCASE):

STUDENT ID (e.g., 123456789):

Instructions

- This exam is a closed-book, closed-notes exam with a duration of 75 minutes and 200 total points.
- You may lose credit if you do not follow the instructions below.
- **At this point, you must write your name and id at the top of this page and add your directory id (e.g., terps) at the end of odd-numbered pages.**
- Please use a pencil to answer the exam.
- **Do not remove the exam's staple or bend any of the pages, as doing so will interfere with the scanning process.**
- **Provide answers in the rectangular areas.** For multiple-choice questions, please fill in the bubble (do not circle). Make a note if you continue a problem on another page(s**).**
- For multiple-choice questions, you can assume only one answer unless stated otherwise.
- **Your code must be efficient and as short as possible.**
- **Provide your answer within the provided box.**
- You don't need to use meaningful variable names; however, we expect good indentation.
- You must stop writing once the time is up.

Grader Use Only

Problem #1 (Miscellaneous)	30	
Problem #2 (Fetch, LocalStorage, Forms)	80	
Problem #3 (Express)	90	
Total	200	

Problem #1 (Miscellaneous)

1. (3 pts) The following code fragment is part of a JavaScript program.

```
let timeInMilliSeconds = 0;
setTimeout(() => console.log("Hello"), timeInMilliSeconds);
```

When will the "Hello" message be printed?

   a. Always immediately (no wait time).
   b. It depends on how many tasks are on the Macrotask queue.
   c. It will take less than 2 seconds.
   d. It will take between 2 and 4 seconds.

2. (3 pts) In which Event loop queue are promise tasks placed?
   a. In the Microtask queue.
   b. In the Macrotask queue.
   c. In either the Macrotask or the Microtask queue.
   d. In the third queue, the GeneralTask queue.

3. (3 pts) The **fetch** API places tasks:
   a. In the Microtask queue.
   b. In the Macrotask queue.
   c. In either the Macrotask or the Microtask queue.
   d. In the third queue, the GeneralTask queue.

4. (3 pts) We can use Axios to generate:
   a. An HTTP request (only get requests).
   b. An HTTP request (only post requests).
   c. An HTTP request (they can be get or post requests).
   d. None of the above.

5. (3 pts) With the fetch API, we can:
   a. Only issue get requests.
   b. Issue both get and post requests.
   c. Only issue post requests.
   d. Not issue any get or post requests.

6. (3 pts) Which type of JavaScript modules system relies on require() to import modules?
   a. ECMAScript Harmony (ES6)
   b. CommonJS
   c. DeferredES1 modules
   d. None of the above.

7. (3 pts) Which **npm** command allows us to install any necessary modules?
   a. npm i
   b. npm init
   c. npm
   d. None of the above.

8. (3 pts) A browser sends a cookie back with every request of the server:
   a. Yes
   b. No

9. (3 pts) Asynchronous functions can use return to result the asynchronous computation.

   a. Yes
   b. No

10. (3 pts) The pseudocode for asynchronous functions below illustrates:

```
asyncFuncA((error1, result1) => {
    // callbackA code
    asyncFuncB((error2, result2) => {
        // callbackB code
        asyncFuncC((error3, result3) => {
            …
```

   a. How we use promises.
   b. How to incorrectly use promises.
   c. The pyramid of Doom problem associated with asynchronous functions calling other asynchronous functions.
   d. None of the above.

DirectoryId (e.g., terps):

Problem #2 (Fetch/LocalStorage/Forms)

Implement a JavaScript application that reads a JSON file with information about news and displays a table with the news. In addition, the program will allow us to save and retrieve news headlines from **localStorage**. When the user clicks the **getNews** button, the JSON file will be read (**using fetch**). The JSON file has an array of news objects, each with a **headline** and **story** attribute. The following is an example of the JSON file:

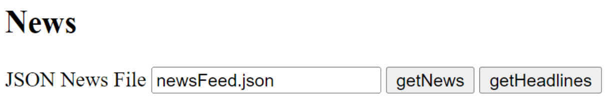
```
[{"headline": "Budget Approved",
  "story" : "After a long time the budget has been approved."
 },
 {"headline": "Terps Win",
  "story" : "The terps won the championship in Texas."
 },
 {"headline": "Pollen Season Arrived",
  "story" : "The pollen season is strong this season. Many students affected."
 }
]
```

The example below illustrates the program's labels, text field, buttons, and functionality. The table will be displayed in a div (using innerHTML). As your code generates the table to display, you should also store the headlines in a variable that you can store in localStorage. **You MUST read the JSON file only once.** Clicking on the **getHeadlines** button will retrieve the headlines from localStorage and display them in the div area (the same one used to display the table). The string with headlines should separate each headline with a vertical bar. Define newFeed.json as the default value for the text field.

For this problem, you can assume a valid JSON file and that we will provide a file name in the text field. You can add any functions you want. **Remember, your code must work for different JSON files (not just the one above).**

Example of running the application

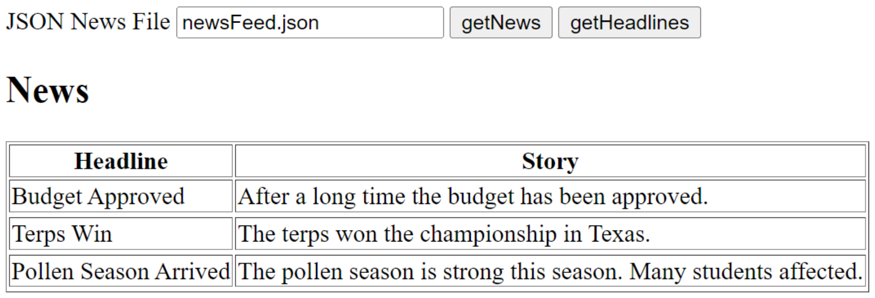
Step #1:



News

JSON News File newsFeed.json

Step #2 (after clicking on getNews)

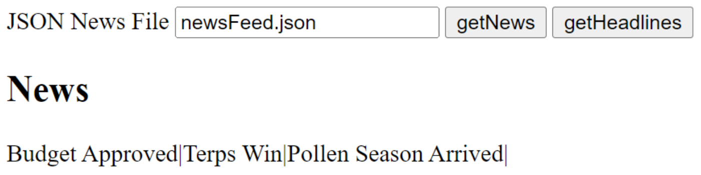


JSON News File newsFeed.json

News

Headline	Story
Budget Approved	After a long time the budget has been approved.
Terps Win	The terps won the championship in Texas.
Pollen Season Arrived	The pollen season is strong this season. Many students affected.

Step #3 (after clicking on getHeadlines)



JSON News File newsFeed.json

News

Budget Approved|Terps Win|Pollen Season Arrived|

Page for Previous Problem

DirectoryId (e.g., terps):

Problem #3 (Express)

Complete the routes/endpoints below. You can assume the code below precedes the routes you will define. A cheat sheet can be found on the last page (do not remove the page with it).

```
const express = require("express");
const app = express();
const path = require("path");
const portNumber = 5000;
const bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: false }));
app.set("views", path.resolve(__dirname, "templates"));
app.set("view engine", "ejs");
```

1. **terpmovies -** When users type the URL **http://localhost:5000/terpmovies,** the server will send the HTML snippet corresponding to the following web page where UMD is a link to www.umd.edu. You can use the <h2> tag for the header.

Welcome to TMT

Sponsored by UMD

```
app.get(
```

2. **location -** When users type the URL **http://localhost:5000/terpmovies/location**, the server will send JSON corresponding to an object with two attributes: street with a value of "123 Main St" and "zipCode" with a value of 20740.

```
app.get(
```

3. **shows -** When users type the URL **http://localhost:5000/terpmovies/shows?movie=cats,** the server will send HTML (in <em> tags) with the value "3 pm". When the URL is **http://localhost:5000/terpmovies/shows?movie=dogs**, the server will send HTML (in <em> tags) with "4 pm". For any other value of the **movie** query parameter, the server will send "Cannot find movie" (no need for <em>) and set the status code to 404. Notice that **cats** and **dogs** can appear in upper, lowercase (or combination) and should be recognized as if we typed **cats** or **dogs**.

```
app.get(
```

4. **reserve -** This endpoint will handle data from a form that uses post and has a field named "tickets". The URL used by the form is **http://localhost:5000/terpmovies/reserve**. The server will access the "tickets" value and multiply that value by 15, generating the total cost of that number of tickets. The server will then send a message that has the following format:

tickets reserved: <NUMBER_OF_TICKETS>, cost: <COST>

Where <NUMBER_OF_TICKETS> is the number of tickets the user sends and <COST> is the total cost. For example, if the user sends 4 using the form, the web page will display the following:

tickets reserved: 4, cost: $60

```
app.post(
```

5. **offers -** This endpoint allows the server to send information about ticket offers (specials) in different states and cities. For example, the URL **http://localhost:5000/terpmovies/offers/md/cp** will allow the web page to display: **Offers available for state md and city cp**. Currently, only offers are available for the **md** state and the **cp** city. Any other combination of state and city will generate a no offers message with the following format: "No offers for state <STATE> and city <CITY)" where <STATE> and <CITY> are the state and city values provided in the URL. The following are examples. Remember, the server must handle additional state and city combinations not just these ones.

http://localhost:5000/terpmovies/offers/md/greenbelt → No offers for state md and city greenbelt
http://localhost:5000/terpmovies/offers/virginia/cp → No offers for state virginia and city cp
http://localhost:5000/terpmovies/offers/ohio/palace → No offers for state ohio and city palace

```
app.get(
```

DirectoryId (e.g., terps):

CHEAT SHEET

```
app.get("/getHTML", (req, res) => {
  res.send("<h1>Returning HTML </h1>");
});

app.get("/getURLParameters", (req, res) => {
   res.send(`Received age: ${req.query.age} salary:
${req.query.salary}`);
});

app.get("/getJSON", (req, res) => {
  const student = { name: "Mary", age: 15 };
  res.json(student);
});

app.get("/getRender", (req, res) => {
   const variables = { semester: "Summer" };
   res.render("welcome", variables);
});
```

```
app.get("/getRouteParameters/:semester/CMSC:course",
(req, res) => {
   res.send(`Received semester: ${req.params.semester},
course: ${req.params.course}`);
});

app.get("/getRedirect", (req, res) => {
   res.redirect("http://www.cs.umd.edu");
});

app.get("/getResourceNotFound", (req, res) => {
  const student = { name: "Mary", age: 15 };
  const http_page_file_not_found = 404;
  res.status(http_page_file_not_found);
  res.send("Cannot find resource (e.g., web page or
file)");
});
```

```
app.post("/postSendingEmailAddress", (req, res) => {
  res.send(`<h2>Received via post email address ${req.body.email}</h2>`);
});

app.put("/putRequest", (req, res) => {
  res.send("<h2>Received put request</h2>");
});

app.delete("/deleteRequest", (req, res) => {
  res.send("<h2>Received delete request</h2>");
});
```