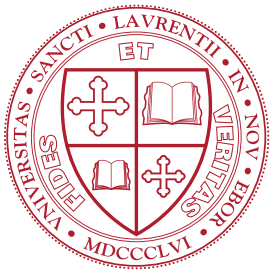

St. Lawrence University
Math, Computer Science & Statistics
CS 332: Web Programming
Fall 2023

Practice Exam 2

Wednesday, November 29, 2023
2:30pm – 4:00pm (90 minutes)

Username: _____ **Student ID:** _____

Name: _____

Honor Pledge:
"I have neither given nor received unauthorized aid on this examination,
nor have I concealed any violations of the Honor Code."

Signature: _____

Instructions:

- The exam is closed book and notes except for one 8.5"x11" sheet of notes (both sides). All electronic device use is prohibited during the exam (calculators, phones, laptops, etc.).
- Print your name, student ID number, and Username **LEGIBLY**. Sign the Honor Pledge. Your exam will not be graded without your signature.
- Record your **USERNAME** at the top of each odd-numbered page in the space provided. This will allow us to identify your exam if pages become separated during grading.
- Please write your answers in the space provided on the exam, and clearly mark your solutions. You may use the backs of the exam pages as scratch paper. Please do not use any additional scratch paper.
- Partial credit will be awarded for partial solutions. If you leave a non-extra-credit portion of the exam blank, **you will receive one-third of the points for that small portion (rounded down)**. If you wish to make an answer to a sub-question blank, draw an "X" through the portion you wish to be blank. **DO NOT** cross out portions of the exam until you near the end because this mark cannot be undone.
- Partial credit will be awarded for partial solutions.
- **DO NOT** detach any pages from this booklet.
- There should be 8 pages in this packet; if you are missing any pages, please let the instructor know.

---

1. **Asynchronous Programming** [X Points]

A) You can access a JSON file containing a property (field) named `student_count` using a GET request to the URL `/student/statistics.json`. Write a JavaScript function, `getStudent Count`, that *asynchronously* accesses this JSON data and returns a Promise producing the value associated with `student_count`.

B) When does a Promise object from a `fetch` request get rejected? Describe an example scenario that would lead to this rejected promise.

C) For this part of the question consider the following function, which generates Promise objects that resolve after a provided number of milliseconds.

```
function wait(ms) {
  return new Promise( resolve => setTimeout(resolve, ms));
}
```

Assuming all timing in JavaScript is correct and that console output takes 0 seconds, what is output of the following code?

```
wait(1000).then( () => {
  console.log("1 s passed");
  return wait(10);
}).then( () => console.log("10 ms passed"));
wait(0).then( () => console.log("0 s passed"));
wait(500).then( () => {
  console.log(".5 s passed");
  return wait(20);
}).then( () => console.log("20 ms passed"));
console.log("done");
```

D) Convert the code from the previous sub-question to use `async` and `await` instead of `then`. Your code should produce the same output as the previous sub-question.

2. **SQL Queries** [X Points]

Consider the following document definition for data stored in the `zips` collection inside the `class_examples` database in a MongoDB instance.

```
{
  "zipcode": <String>,
  "city": <String>,
  "state": <String>,
  "latitude": <Number>,
  "longitude": <Number>,
  "pop": <Number>
}
```

Fill in the blanks in the following JavaScript-based queries to perform the specified actions on the `zips` collection. Write a **single word or number** in each blank.

```
const db = client.db(class_examples);
```

A) Create a variable to represent the `zips` collection in the `class_examples` database stored in a MongoDB instance.

```
const zips = db._____("zips");
```

B) Get all ZIP codes with a population greater than 100,000.

```
const result = await zips.find({

    _____ : {

        _____ : 100000

    }
});
```

C) Query how many documents there are in the `zips` collection.

```
const rows = await zips._____({});
```

D) Add a ZIP code document to the collection and print the ID given by MongoDB to the document. The data is stored in the variable `newZip`.

```
const result = zips._____(newZip);

_____.log(result._____);
```

E) Find the minimum ZIP code in each state.

```
const smallest = await zips._____([
    {
        $group: {
            _id: "_____",
            zip: {
                _____: "_____"
            }
        }
    }
]);
```

F) Find all of the ZIP codes in Virginia that are north of latitude 38 in descending order.

```
const result = zips._____({
    _____: "_____",
    _____: {
        _____ : _____
    }
}).____________({
    _____: _____
});
```

3. **RESTful Interfaces** [X Points]

A) Explain the design principles of RESTful interfaces.

4. **Authentication and Passwords** [X Points]

You see a post on Piazza from an anonymous classmate asking the following question:

> For my implementation of the backend of SlugFest, I would like to be able to email users their passwords if they forget them. I can't figure out how to make the `argon2` hashing function give me back the original password. Should I just store the original password in my database, or is there a better hash function that I can use?

Write a response to your classmate that addresses each aspect of their question. Be sure to consider security, performance, and best practices in your answer. Part of your answer should include a suggested approach to solving the problem the student is experiencing.

5. Express and Node.js [X Points]

A) Write an excpert of code that adds a new route to an Express application, `app`, that will respond to GET requests to the URL `/calculate_change/:value` where `:value` is a number provided as an *express parameter*. The route should return a JSON object with the following fields:

- `twenties`: the number of $20 bills needed to make change for `:value`
- `tens`: the number of $10 bills needed to make change for `:value`
- `fives`: the number of $5 bills needed to make change for `:value`
- `ones`: the number of $1 bills needed to make change for `:value`
- `quarters`: the number of $0.25 coins needed to make change for `:value`
- `dimes`: the number of $0.10 coins needed to make change for `:value`
- `nickels`: the number of $0.05 coins needed to make change for `:value`
- `pennies`: the number of $0.01 coins needed to make change for `:value`

B) Explain the purpose of an Express application run with Node.js. How does it relate to the overall architecture of a web application?

C) What is the purpose of the `package.json` file in a Node.js package? What is a Node.js package?

D) How can Node.js code be split across several source code files? How do you make code available from one file to another? How do you access code from another file?

E) Explain the breakdown of the directory structure of the ToDoApp package. What is the roll of the code in the `models`, `routes`, and `public` directories, respectively? What is the advantage of structuring code this way?

F) What are server-side options for maintaining state between HTTP requests?