CSE 154: Web Programming Summer 2019

Exam 2 Key

1. Short Answers (10pts)

1. Callbacks vs. Promises. (1pt)

Provide one advantage of using promisified versions of callback-based functions like `fs.readFile`. For full credit, you must clearly identify the advantage between the two options, and may refer to other promisified functions we've covered in your justification.

Possible answers include identifying promisified functions as being easy to use with async/await avoid callback pyramids, shadowing of callback function parameters, making asynchronous code easier to read and more maintainable, etc.

2. Regex. Circle the strings which match the regex pattern shown below: (1.5pt)

`/^[A-Z][a-z]+.[a-z]{3}$/`

- `Helloworld.foo`
- `A.biz`
- `Apple@com`
- `Abc.d3`

3. Node/SQL Connection. Which of the following statements are true about **`MySQL`** databases and the **`promise-mysql`** functions? Circle all true statements. (2.5pt)

- A database can contain more than one table.
- If an error occurs in **`mysql.createConnection`**, the returned db object will still be defined.
- If an error occurs in **`db.query`**, the db object will still be defined.
- When we have a db object used in an Express endpoint function, we must **always** call `db.end()` **before** using **`res.send()`**
- When we have a db object used in an Express endpoint function, we must **always** call `db.end()` **after** using **`res.send()`**

4. SQL TABLE Relationships. Which of the following statements are true about **`FOREIGN KEY`**s and **`PRIMARY KEY`**s? Circle all true statements. (3pt)

- Every table must have at least one **`PRIMARY KEY`**.
- A table can have more than one **`FOREIGN KEY`**.
- **`PRIMARY KEY`**s must be **`INT`**s
- FOREIGN KEYs must reference either a PRIMARY KEY or a UNIQUE column in another table.
- A PRIMARY KEY referenced by a FOREIGN KEY can have the same column name.
- A PRIMARY KEY referenced by a FOREIGN KEY can have a different column name.

5. Asynchronous Program Execution.

Solution: c.

2. (Node.js Web Service) Books on a Budget (16pts)

```
// Part A Solution:
app.get("/courses", async (req, res) => {
  res.type("text");
  try {
    let files = await readdir("data");
    let result = "";
    for (let i = 0; i < files.length; i++) {
      let courseName = getCourseName(files[i]);
      result += courseName + "\n";
    }
    res.send(result.trim());
  } catch (err) {
    res.status(500).send(SERVER_ERROR_MSG);
  }
});

// Part B Solution:
app.get("/books/:course", async (req, res) => {
  let classname = req.params.course;
  try {
    let data = await readFile("data/" + classname + ".txt", "utf-8");
    let result = {};
    let lines = data.split("\n");
    result.name = lines[0];
    let minPrice;
    let minSeller;
    for (let i = 1; i < lines.length; i++) {
      let lineParts = lines[i].split(":");
      let seller = lineParts[0];
      let price = parseFloat(lineParts[1]);
      if (!minPrice || parseFloat(price) < minPrice) {
        minPrice = price;
        minSeller = seller;
      }
    }
    result.price = minPrice;
    result.seller = minSeller;
    res.json(result);
  } catch (err) {
    if (err.name === "ENOENT") {
      res.status(400).send("No data found for course.");
    } else {
      res.status(500).send(SERVER_ERROR_MSG);
    }
  }
});
```

3. **(Client-side JS with AJAX)** Books on a Budget (12pts)

```
"use strict";
(function() {

  window.addEventListener(“load”, init);

  // Write your solution here, starting with init function.
  function init() {
    fetch("/courses")
      .then(checkStatus)
      .then(resp => resp.text())
      .then(populateClasses)
      .catch(displayError);
  }

  function populateClasses(resp) {
    resp = resp.trim().split("\n");
    let courses = id("courses");
    for (let courseName of resp) {
      let btn = gen("button");
      btn.textContent = courseName;
      btn.addEventListener("click", getCheapestBook);
      courses.appendChild(btn);
    }
  }

  function getCheapestBook() {
    let classname = this.textContent;
    fetch("/books/" + classname)
      .then(checkStatus)
      .then(resp => resp.json())
      .then(populateInfo)
      .catch(displayError);
  }

  function populateInfo(resp) {
    id("book-info").classList.remove("hidden");
    id("book-name").textContent = resp.name;
    id("seller").textContent = resp.seller;
    id("price").textContent = resp.price;
  }

  function displayError() {
    id("book-info").classList.add("hidden");
    id("courses").classList.add("hidden");
    id("error-info").classList.remove("hidden");
  }
})();
```

4. (SQL) Who needs .txt-books anyways? (12pts)

a (3pts).

```
SELECT DISTINCT course
FROM requirements
ORDER BY course;
```

b (3pts).

```
DELETE FROM offers WHERE price <= 0.1;
```

c. (6pts)

One WHERE solution is provided below:

```
SELECT name, seller, price, course
FROM books b, offers o, requirements r
WHERE b.id = o.book_id AND b.id = r.book_id
ORDER BY course, price DESC;
```

One equivalent JOIN solution is provided below:

```
SELECT name, seller, price, course
FROM books b
JOIN offers o ON b.id = o.book_id
JOIN requirements r ON b.id = r.book_id
ORDER BY course, price DESC;
```

There was a contradiction in the expected results and exam specification on price ordering. We accepted both ASC and DESC orderings.