CS 498RK: The Art of Web Programming
Midterm

Read *ALL* the instructions before you begin the exam.

General Information

- This is a **take-home open-book** exam.
- This exam is worth 20% of your total grade in the course. You should attempt to answer every question, as partial credit will be awarded for incomplete work approaching a correct solution. However, you should not perform a brain dump on any problem in the hopes that something you write down will be awarded points. Extraneous information (especially incorrect extraneous information) will be heavily penalized.

Timeline

- This midterm became available for view on **10/20/2021 at 2:00 PM CT**.
- You have **48 hours** to submit your responses.
- The deadline to submit your responses is on **10/22/2021 at 2:00 PM CT**.

Academic Integrity Policy

- **This is an individual assignment; any type of collaboration is strictly prohibited.**
- You are encouraged to use any and all resources available. However, you must clearly cite any contributing source. Failure to cite any contributing source will be considered cheating regardless of the reason for the omission. Likewise, verbatim duplication of any source will always be considered plagiarism.
- We take issues of academic integrity very seriously, and you will fail the course if you're caught cheating on an exam.

What to Upload

- You will need to upload a `.zip` archive that contains your responses to all of the questions.
- First, create a directory and change its name to your NetId. For each question, create a numbered directory that corresponds to the question number. Each question will ask for specific files that need to be included as your response. Here is what your final directory structure should look like:

```
<your netid>
├── 1
│   └── [your files for q1]
├── 2
│   └── [your files for q2]
└── 3
    └── [your files for q3]
```

FAQ

- *How should I cite the resources I used?*
  You can create a `references.txt` file under each question directory to indicate the resources you used. For each reference, please include a short explanation of what you actually used. Do not give a bare list of URLs.
- *I was not able to / forgot to submit the midterm on time. What should I do?*
  We wont accept any late submissions for the midterm barring extreme circumstances.
- *Can I use my late days for the midterm?*
  No.
- *Can I submit multiple times?*
  Yes, we will only grade your latest submission.
- *I already submitted the midterm but I want to change one of my answers. Do I need to resubmit all my answers?*
  Yes, your submission should contain all your answers.
- *I have other files related to the question. Should I upload them as well?*
  You should only upload what the question asks you. Your answers should always be contained within the files required by the question.

BY UPLOADING A SUBMISSION FOR THIS MIDTERM YOU CERTIFY THAT YOU UNDERSTAND AND AGREE TO ABIDE BY THE ACADEMIC INTEGRITY POLICY.

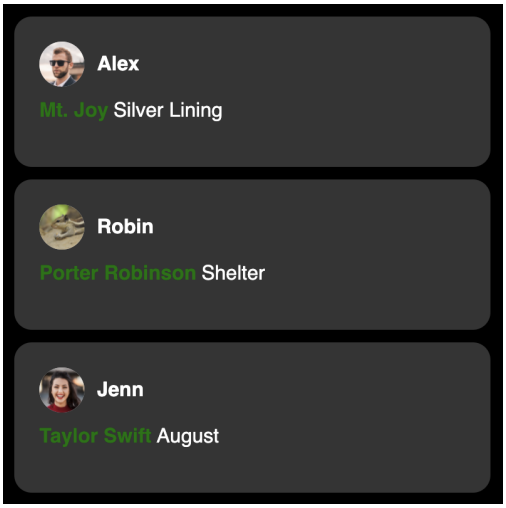
IF YOU HAVE ANY QUESTIONS ABOUT WHAT CONSTITUTES ACCEPTABLE BEHAVIOR, PLEASE OPEN A PRIVATE QUESTION ON PIAZZA AND ASK THE COURSE STAFF.

#	1	2	3	4	5	Total
Score						
Possible	20	25	30	15	10	100

1 [20 pts] Laying Out Listenify

Your good friend Alex pitches you on his new billion dollar app idea, a music streaming service called "Listenify". Alex has written the code for a React component that will list out your friends and the songs that they are currently playing. He also sketched a mockup for the component, but asks you for CSS help.

Mockup



Requirements

- **Your solution must use Flexbox to achieve the desired layout.**
- The CSS should style the component to match the mockup as close as possible.
- The name of the friend and the name of the artist needs to be bold.
- The brand color is green, so make the name of the current artist green to signal to the user that it is clickable.
- The cards and the profile picture need to be rounded.
- The theme of the UI must be dark.
- This list needs to be displayed vertically like in the provided mockup.

React Component

```
const FriendsList = props => {
  return (
    <div className="parent">
      {props.friends.map(friend => (
        <div className="card">
          <div className="profile">
            <img className="photo" src={friend.photoUrl} alt={friend.name} />
            <span className="text name">{friend.name}</span>
          </div>
          <div className="now-playing">
            <span className="text artist">
              {friend.currentArtist}
            </span>
            <span className="text song">
              {friend.currentSong}
            </span>
          </div>
        </div>
      ))}
    </div>
  )
}
```

What to Submit

Submit the following file(s):

- `listenify.css`, containing your CSS for achieving the layout.

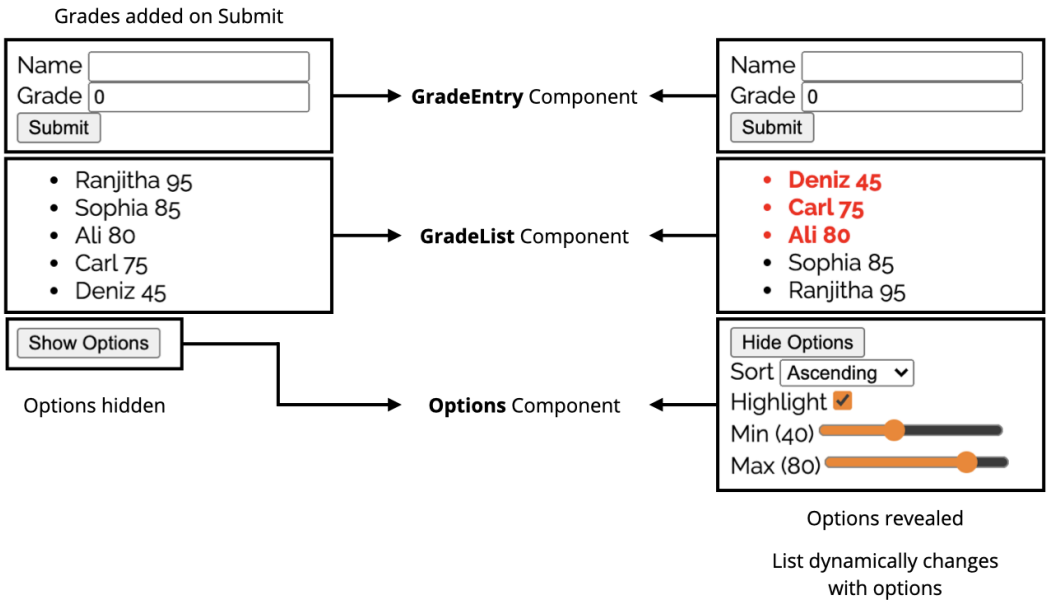
2 [25 pts] Reactive Gradebook

In one of your other classes, your professor announces that the online gradebook they use for keeping track of student grades has shut down. They ask if anyone knows how to implement a system to record the grades and offer extra credit. Since in CS498RK, you learned how to create React projects, you are confident in your ability to implement a new Gradebook in React. As you really need the extra credit, you volunteer and your professor provides the following functional requirements along with a mockup.

Requirements

- Users can enter a new grade record by entering grade information (name and grade out of 100) and clicking the submit button.
- The grade list should be sorted by grades (descending) by default.
- Users can show and hide the options menu which includes controls for sorting and highlighting.
- Users can change the list sort order between ascending and descending.
- Users can highlight list items that are in a specified range. This can be done by toggling a "highlight" option and setting the min and max values.

Mockup



You will need to implement the following React components:

- `Gradebook`: The top-level component that uses GradeEntry, GradeList, and Options components.
- `GradeEntry`: A component that has two input fields with labels that allow entering grades and a submit button to save the information.
- `GradeList`: A component that lists all the entered grades and changes dynamically with respect to the options.
- `Options`: A component that allows setting display options for the Gradebook. It needs to have the following controls: Sort, Highlight, Min, Max.

Clarifications

- **Your solution must use React.**
- You are not allowed to use other external libraries.
- Your Gradebook does **not** need to look exactly like the provided images. It just needs to satisfy the functional requirements.
- You do not need to use any styles.
- You will need to set up a new React project to work on your responses.
- You do not have to implement grade removal or edit.
- You can use Hooks.

What to Submit

Submit the following file(s):

- `Gradebook.js`, the top-level component,
- `GradeEntry.js`, the component for entering grades,
- `GradeList.js`, the component for listing grades,
- `Options.js`, the component for adjusting options.

3 [30 pts] Variations on Memoization

Memoization is the process of building a function that is capable of remembering its previously computed answers. This can markedly increase performance by avoiding needless complex computations that have already been performed once.

In this question, we'll explore different ways of implementing memoization in JavaScript using the functional aspects of the language.

(a) **[3 pts]** Write a JavaScript function that computes prime numbers. Given a value, *isPrime()* determines whether that value is a prime number or not. The function should run in $\Theta(k)$ time, and don't forget your friend, the modulo operator (%).

(b) **[7 pts]** Now let's memoize *isPrime()*. Implement memoization by leveraging the fact that **JavaScript functions can store state**.

The JavaScript function *memoizedIsPrime()* should store previously computed results in its *values* property. If the answer has previously been computed and stored in *values*, it can be returned directly. If it hasn't been computed, then perform the calculations needed to determine if the value is prime. Finally, store the result in *values* and return it. (Remember to initialize *values* at the beginning of the function if it hasn't already been initialized.)

(c) **[10 pts]** The previous part assumed that we had access to the function that we wanted to memoize, but what if that's not the case? Write the JavaScript function *memoized* that can be used to memoize any function.

The *Function.prototype* declaration makes *memoized()* a method of all functions. Just like *apply()* and *call()*, *memoized()* can now be called on any function. The assert statements below show how to invoke *memoized()* on *isPrime()*.

```
assert(isPrime.memoized(5), "The function works; 5 is prime.");
assert(isPrime._values[5], "The answer has been cached.");
```

(d) **[10 pts]** However, requiring the caller of the *isPrime()* function to call it through its **memoized** method can be annoying. It turns out that closures can be used to write a new function that memoizes all of its function calls automatically.

Write **memoize()**, which wraps the original function with the **memoized()** method applied, such that it will always return the memoized version of the original function.

```
var isPrime = (function(num) {
...
}).memoize();
assert(isPrime(17),"17 is prime");
```

What to Submit

Submit the following file(s):

- `isPrime.js`, which provides the `isPrime` function you implemented as a `default export`,
- `memoizedIsPrime.js`, which provides the `memoizedIsPrime` function you implemented as a `default export`,
- `memoized.js`, which adds the `memoized` function you implemented to `Function.prototype`,
- `memoize.js`, which adds the `memoize` function you implemented to `Function.prototype`,

4 [15 pts] Spotify Schemas

Suppose you are a backend engineer at Spotify tasked with designing the MongoDB schemas for their app. You design a database with three collections: `songs`, `albums`, and `users`.

(a) **[5 pts]** In your schema design, each `album` document maintains a list of ObjectID references to the songs that comprise the album.

Here is an sample `song` document:

```
> db.songs.findOne()
{
    _id : ObjectID("AAAA")
    name : "Code Monkey",
    artist : "Jonathan Coulton",
    year: "2006"
    duration: 318000
}
```

Here is a sample `album` document:

```
> db.albums.findOne()
{
    _id : ObjectID("BBBB")
    name : "Thing a Week III",
    artist : "Jonathan Coulton",
    year: "2006",
    songs : [
      ObjectID("AAAA"),
      ObjectID("F17C"),
      ObjectID("D2AA"),
      // etc
    ]
}
```

Copy the MongoDB query code below to a file called `a.txt` and complete it for extracting an array of all the song titles for the album "Thriller," by "Michael Jackson." `album_songs` should contain output like:

```
[{"name": "Baby Be Mine"}, {"name": "The Girl Is Mine"},...]
```

MongoDB query code:

```
> var db = connect("localhost:27017/myDatabase");
> var album =
> var album_songs =
```

(b) [5 pts] Your boss points out that two-way referencing between the `songs` and `albums` collections would be a better schema design. Do you agree with him? Why or why not?

(c) [5 pts] Your friend, who is a frontend engineer, says he needs a way to correlate songs with listeners' demographic information such as location, age, and gender. He suggests that you augment the `song` document schema to include a list of ObjectID references to users who listen to the songs. He claims this strategy would allow him to perform application-level joins to understand the relationship between song and user data. Is this a good strategy? Why or why not?

What to Submit

Submit the following file(s):

- `a.txt`, containing completed MongoDB query code for part (a),
- `b.txt`, containing your answer to part (b),
- `c.txt`, containing your answer to part (c).

5 [10 pts] Asyncing Feeling

This question uses the `Fetch API`:

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

Assume all HTTP requests are valid.

(a) **[5 pts]** Which of the following is a possible output of this code?

```
function baz() {
  console.log("baz");
}

let gotUsers = false;
fetch("/api/users").then(function foo() {
  console.log("foo");
  gotUsers = true;
});

fetch("/api/tasks").then(function bar() {
  console.log("bar");
  while (!gotUsers) {
    // loop
  }
});

baz();
```

A.
```
baz
bar
```

B.
```
foo
bar
baz
```

C.
```
baz
bar
foo
```

D. Both A and B

E. Both B and C

(b) **[5 pts]** Explain the reasoning for your choice.

What to Submit

Submit the following file(s):

- `a.txt`, containing your choice,
- `b.txt`, containing your reasoning for your choice for part (a).