CS312 Fall 2024 – Midterm 1 Solution

Name: _____

Date: _____ **Start time:** _____ **End time:** _____

Honor Code:

Signature: _____

This exam is open course web page, open Ed (reading, not posting), open notes, open slides, open your assignment solutions and open calculator, but closed everything else (e.g., consulting with others, searching online, using generative AI are not permitted). **You have 2 hours in a single sitting to complete the exam.** When you are done, scan the exam and upload it to gradescope. An additional 30 minutes have been provided to give you time to print and scan your exam. Read the problem descriptions carefully and write your answers clearly and *legibly* in the space provided. Circle or otherwise indicate your answer if it might not be easily identified. If you need more room you may use extra sheets of paper as long as the problem number is clearly labeled and your name is on the paper. These must also be scanned. If you attached extra sheets indicate on your main exam paper to look for the extra sheets for that problem.

Learning Target	**Assessment**
1	
2	
3	
4	
5	
6	
7	
8	

Question 1. User stories

You work for the technology arm of a large city's public transportation department. When interviewing stakeholders, multiple individuals described a feature where bus riders could input their desired stop on the app when they board the bus to allow for future notifications for both the rider and the driver. Write two I.N.V.E.S.T. user stories, one from the perspective of a bus rider, the other from the perspective of a bus driver. Your user stories will be evaluated on format and quality.

(a) Stakeholder 1: Bus Rider

Solution: As a rider of the bus,
I want to input my stop when I board and receive a notification when we are arriving,
So that I do not miss my stop.

(b) Stakeholder 2: Bus Driver

Solution: As a bus driver,
I want to know where we will need to stop,
So that I do not have to pull over at the last minute.

Question 2. Javascript

Assume the function `wait(sec)` returns a promise that resolves after `sec` seconds have elapsed. Assume that every other operation is instantaneous (e.g., takes 0 milliseconds). Remember that `new Date()` returns the number that represents milliseconds since the midnight at the beginning of January 1, 1970, UTC.

```
1  let elapsed = (start, secs) => {
2      return wait(secs).then(() => {
3          console.log(new Date() - start);
4      });
5  }
6
7  const start = new Date();
8  elapsed(start, 3).then(() => {
9      elapsed(start, 4);
10 });
11 console.log("this too!");
```

```
1  const elapsed = async (start, secs) => {
2      await wait(secs);
3      console.log(new Date() - start);
4  };
5
6  const start = new Date();
7  elapsed(start, 3);
8  await elapsed(start, 4);
9  console.log("this too!");
```

Consider the two code snippets above. Write the expected output for left-side code below on the left. If the right-side code produces the same result indicate below, otherwise provide the expected output below on the right.

◯ Both snippets produce the same output

Solution:

```
this too!
3000
7000
```

Solution:

```
3000
4000
this too!
```

Question 3. Testing

You are developing a React component named `Menu` for the dining hall, which takes an array of `menuItems` as a prop. The menu has a checkbox `Veg Only` (which starts unchecked) that filters out all options with meat and shows the menu with only vegetarian options. Checking the box should hide items with meat, and unchecking the box should display those items. Using the skeleton below, implement **pseudo-code** for a F.I.R.S.T. unit test for the behavior of the checkbox as described above. You do **not** need to provide executable Javascript, instead describe the steps of your test as pseudo-code. For example, one of the steps in your pseudo-code might be:

Assert mock function was not called

You may or may not need all of the functions below. You only need to include pseudo-code in bodies of the functions relevant to your answer. You may use the specified `menuItems` array in your test if desired.

```
describe("Menu", () => {
  const menuItems = [
    {name: "cheese pizza", hasMeat: false},
    {name: "pepperoni pizza", hasMeat: true}
  ];
  beforeEach(() => {
```

Solution:

```
  });
  afterEach(() => {
```

Solution:

```
  });
  test("Displays/hides options with meat based on the Veg Only checkbox", () => {
```

Solution:

```
Render the Menu component with menuItems as a prop
Assert that "cheese pizza" is displayed
Assert that "pepperoni pizza" is displayed
Find and click the "Veg Only" checkbox
Assert that "cheese pizza" is displayed
Assert that "pepperoni pizza" is not displayed
Find and click the "Veg Only" checkbox
Assert that "cheese pizza" is displayed
Assert that "pepperoni pizza" is displayed
```

A satisfactory test would start by checking the initial state (both pizzas are shown). Then, the checkbox should be clicked, and only the cheese pizza should appear. When the checkbox is clicked again, both pizzas will appear.

```
  });
});
```

Question 4. Scenarios

Imagine you are implementing a shopping cart for an e-commerce site. There is a `Quantity` component associated with each item; + and - buttons are used to control the quantity, which is also specified numerically on the screen. The - button is disabled when the quantity is 1. Write a Gherkin-style test scenario for increasing and decreasing the quantity that covers the disabling of the - button. You do not need to provide the implementation details of the tests, just describe the scenario for the test.

Solution:

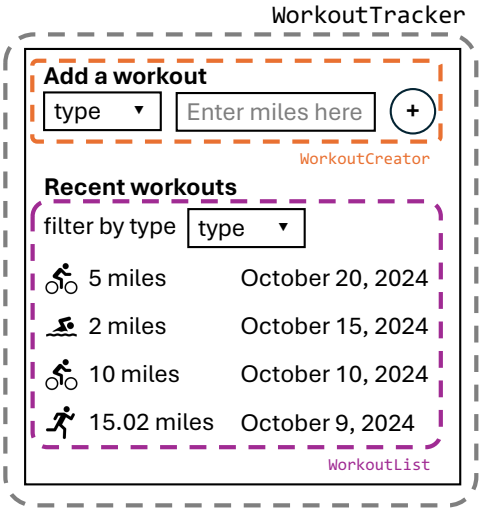
We want to ensure that the button is enabled when the user goes from a quantity of 1 to 2, and disabled when they go back from 2 to 1.

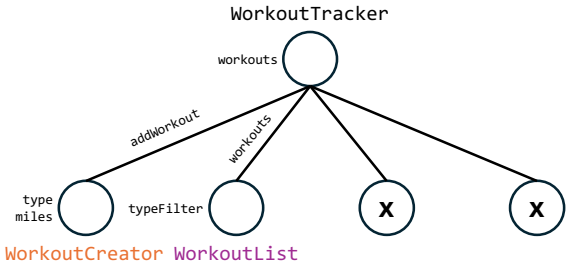
```
Given a shopping cart with one notebook in it,
Then the quantity should be 1,
And the - button should be disabled,
When I click on the + button,
Then the quantity should be 2,
And the - button should be enabled,
When I click on the - button,
Then the quantity should be 1,
And the - button should be disabled.
```

Question 5. React

You are implementing the workout tracker shown below with React. Entering a workout type in the dropdown and decimal number of miles then clicking the "+" should add a workout to the tracker and update the recent workouts. Selecting a workout type under "recent workouts" should filter the workouts to show only biking, swimming, or running workouts. Outline and label the wireframe (below, top) with a possible set of components. Label the tree (below, bottom) with components to show the hierarchy. Label the tree nodes with state implemented in that component and label the tree edges with props passed to each component (similar to the figure in programming assignment 2). Repeated components can be labeled once in tree. The top-level component `WorkoutTracker` is labeled for you. Any implementation reflecting good React practices will be accepted. You may not need all the nodes in the tree or may need to add nodes depending on your design; cross out any unused nodes. Your component, state and prop names should be sufficiently descriptive that their role is clear.

Solution:





An explanation is not required for full credit, but is provided here for clarity. We maintain the workouts as an array of objects named `workouts`. Since that state is needed by the both form and the list, we locate it in parent `WorkoutTracker` component, and pass it as a prop to `FilterableWorkoutList`. `WorkoutCreator` implements a form with controlled components, and thus has state for the type of workout and number of miles. Clicking "+" invokes a callback provided as a prop to add a workout to the list. The `typeFilter` state is only used in `FilterableWorkoutList`, so it is located in that component. A component for each individual workout is optional.

Question 6. REST

For each of the following pages in a NextJS-based petsitting application, provide an appropriate RESTful front-end (browser) URL for that page and, where relevant, an appropriate RESTful server API endpoint (HTTP verb and URL) that the component would interact with. An example is provided below.

Page	Page URL	API HTTP verb and URL
View all articles on simplepedia	**/articles**	**GET /api/articles**
View an owner's profile	**/owners/1**	**GET /api/owners/1**
Update a pet's profile	**/owners/1/pets/1/edit**	**PUT /api/owners/1/pets/1**
Create a new sitter profile	**/sitters/new**	**POST /api/sitters**

Question 7. Data modeling

You are developing a wedding guest list management website. You will be using a relational database to store the data for this application.

(a) Identify the *minimum* set of models you would define in your server backend to implement the following user story:

As an engaged couple, we want to organize our guests into households, so that we can send one save-the-date to each address.

Solution: The minimum models would be `Couple`, `GuestHousehold`, and `Guest`.

(b) Which of the following best describe the relations between the following pairs of entities. Select one answer for each pair, then briefly explain your answers.

`Couple` and `GuestHousehold`

- ◯ One-to-One
- √ **One-to-Many**
- ◯ Many-to-Many
- ◯ No relation

`GuestHousehold` and `Guest`

- ◯ One-to-One
- √ **One-to-Many**
- ◯ Many-to-Many
- ◯ No relation

Solution: A couple will invite multiple guests, so a one-to-many relationship is the most appropriate; if well-motivated, many-to-many may be accepted (e.g., if addresses are stored in the server for use by future couples), but this is obviously a privacy violation. Each household might be made up of one or more guests, but each guest should be part of a single household.

(c) In a normalized schema designed for a relational database (RDBMS), what schema would be needed to implement the user story in part a? You do not need to provide SQL, just the attributes, their types, the primary key, and any foreign key constraints. Ensure that *address* is included in your schema so that you can send the cards; it is OK if you store it as a single string rather than individual fields. Your schema should be designed to minimize repetition of addresses.

Solution:

`Couple` would have an integer primary key and two strings each representing the couple names

`GuestHousehold` would have a foreign key to `Couple` and an integer primary key. Address should also be stored with `GuestHousehold`, as all guests in the household should have the same address.

`Guest` would have an integer primary key, a name (string), and a foreign key to `GuestHousehold`

Question 8. Development processes

For each of the following, indicate whether the action would be consistent with the best practices for software development as described in class or not consistent. Here "consistent" is defined as consistent with good development practices generally, not that it was required as part of our class. Briefly explain your answer.

(a) Create a PR for every change, even if you're creating a new file that won't cause merge conflicts.

√ **Consistent** ◯ Not consistent

> **Solution:** When you create a PR, you'll be alerted to possible merge conflicts, but you will also start CI tests and have the opportunity to receive manual feedback from your teammates. All changes, no matter how small, should be accompanied by a PR.

(b) Write exactly three tests before writing your function.

◯ Consistent √ **Not consistent**

> **Solution:** In TDD, you should *iteratively* write tests for your function, then write the code to make the test pass. You do not need to write multiple tests prior to writing any of the code for your function, and there isn't a magic number of tests that should be written for each function.

(c) Break down a user story into multiple smaller stories if it does not meet the **V** criteria in I.N.V.E.S.T.

◯ Consistent √ **Not consistent**

> **Solution:** If a user story is too large to complete in one sprint, you should break it into multiple user stories that are each INVESTable. However, this would involve not meeting the S criteria (small). If a feature is not valuable, you shouldn't be building it!