CS312 Spring 2024 – Midterm 1 Solution

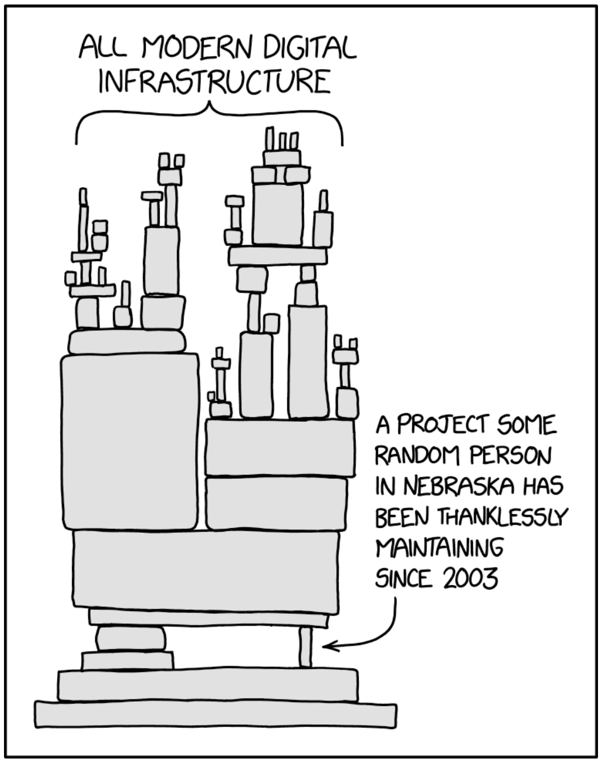
Name: ______________________________

Date: ________________ **Start time:** ______________ **End time:** ______________

Honor Code:

Signature: ____________________________

This exam is open course web page, open Ed, 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.** 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. You may use extra sheets of paper, stapled to your exam, if you need more room, as long as the problem number is clearly labeled and your name is on the paper. If you attached extra sheets indicate on your main exam paper to look for the extra sheets for that problem.


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

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

Question 1. User stories

You are developing a web application for managing a library. When interviewing stakeholders, multiple respondents described wanting to reserve books online for subsequent pickup. Write two I.N.V.E.S.T. user stories for this feature, one from the perspective of a library patron, the other from the perspective of a library staff member. Your user stories will be evaluated on format and quality.

(a) Library patron:

Solution:
As a library patron,
I want to reserve a book online
So that I can ensure its available when I visit the library.

(b) Library staff member:

Solution:
As an library staff member,
I want to obtain a list of new or pending reservations
So that I can efficiently prepare materials for pickup by patrons.

Question 2. Javascript

Assume the function `any_n(promises, n)` returns a promise that resolves when all, or at least `n`, of the promises in the array `promises` have resolved, whichever happens first. The promise returned by `any_n` will resolve with an array of the fulfilled values of `promises`. The function `wait(sec)` returns a promise that resolves after `sec` have elapsed.

```
function do(time) {
  return wait(time).then(() => {
    console.log(time)
  });
}

do(3);
any_n([do(1),do(2),do(4)], 2).then(()=>{
  console.log(5)
});
```

```
async function do(time) {
  await wait(time);
  console.log(time);
}

await do(3)
await any_n([do(1),do(2),do(4)], 2);
console.log(5);
```

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:

```
1
2
5
3
4
```

Solution:

```
3
1
2
5
4
```

Question 3. Testing

You are developing a React component named `PasswordGenerator` for generating a random password. The component has a select box for specifying common allowed character sets, e.g., “letters and numbers”, a numeric input for specifying the length, a “Create” button to create a new password, and a text field to display the generated password. Using the skeleton below, implement **pseudo-code** for a F.I.R.S.T. unit test to verify that each time the user clicks the “Create” button a new correctly formatted password is generated. 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.

```
describe("Password generator", () => {
  beforeEach(() => {
```

Solution:

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

Solution:

```
  });
  test("Generates a new correctly formatted password (letters/numbers) on each click", () => {
```

Solution:

```
Render the PasswordGenerator component
Find and select the "letters and numbers" option in the character input
Find and set the length input to be 12
Find "Create" button and simulate click.
Find the password display field and assert it contains a string of length 12
  consisting of only letters and numbers.
Save the current generated password as a variable
Find "Create" button and simulate click.
Find the password display field and assert it contains a string of length 12,
  consisting of only letters and numbers, which is different from the previous
  password.
```

A satisfactory tests would assert the password is the correct length, contains the specified characters and that clicking “Create” twice generates two different passwords. The design does not imply that this component expects a callback as a prop, instead as indicated, the generated password is displayed in a text field.

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

Question 4. Scenarios

In your application, the user profile page has a password field with an associated “Update” button to update the user’s password. If the user updates their password (to a new password), the user is logged out, redirected to the login page, and a confirmation message is shown in a “flash” (i.e., a message that is shown for a short period of time as a banner at the top of the page). Write a Gherkin-style test scenario for updating a password. You do not need to provide the implementation details of the tests, just describe the scenario for the test.

Solution:

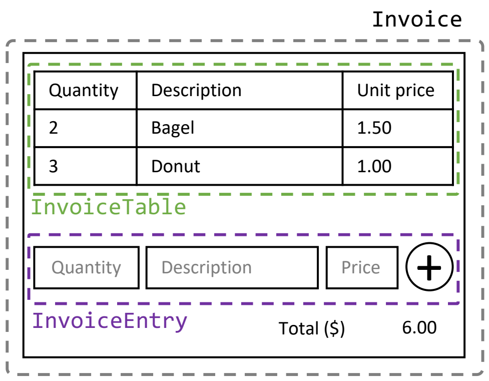
We want to make sure that the user can update their password, and is signed out and redirected to the login page with the confirmation message upon doing so. To setup the scenario, the user would need to be loggen in and on the user profile page. The scenario would be:

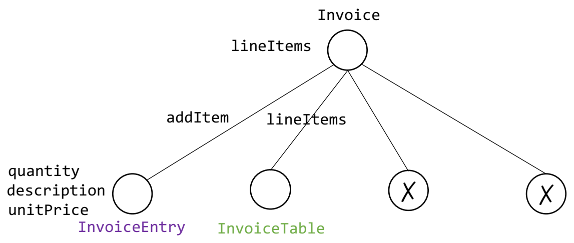
```
Given the user has logged in with username and password "user1" and "password1"
And the user is on the user profile page
When the user enters the new password "password2" in the update field
And clicks the "Update" button
Then the user should be logged out
And the user should be redirected to the login page
And a confirmation message "Password updated successfully" should be displayed
  in a flash message
```

Question 5. React

You are implementing the invoice creator shown below with React. Entering a integer quantity, string description and decimal unit price and clicking the "+" should add an entry to the invoice and update the total at the bottom (as the sum of the quantity times the unit price for all entries). Outline and label the wireframe (below, left) with a possible set of components. Label the tree (below, right) 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 `Invoice` 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 invoice entries as an array of objects named `lineItems`. Since that state is needed by the both form and the table, we locate it in parent `Invoice` component, and pass it as a prop to `InvoiceTable`. `InvoiceEntry` implements a form with controlled components, and thus has state for the quantity, description, and unit price. Clicking "+" invokes a callback provided as a prop to add an entry to the invoice. The total can be derived from the invoice entries and thus should be implemented as a separate piece of state (to ensure a single source of truth).

Question 6. REST

For each of the following pages in a NextJS-based online store, 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 component would interact with. An example is provided below.

Page	Page URL	API HTTP verb and URL
Add a new article to Simplepedia	**/edit**	**POST /api/articles**
Change the price for a product	**/products/1/edit**	**PUT /products/1**
View products sorted in ascending order of price	**/products?sort=price&asc**	**GET /products?sort=price&asc**
View a specific user's past orders	**/users/1/orders**	**GET /users/1/orders**

Question 7. Data modeling

Assume you are developing a web application for supporting student clubs at a college, e.g., membership lists, calendars, announcements, etc. 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 a student, I want to view a consolidated a list of announcements for all the clubs I am a member of, so that I can stay informed about all club activities.

Solution: The minimum models would be `User`, `Club` and `Announcement`. The `Announcement` stores the content of the announcement for a club. A `User` is linked to a `Club` through a join table, e.g., `Membership`. Answers with and without the join table as a separate model were accepted.

(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.

Student and Club
- ○ One-to-One
- ○ One-to-Many
- [✓ **Many-to-Many**]
- ○ No relation

Club and Announcement
- ○ One-to-One
- [✓ **One-to-Many**]
- ○ Many-to-Many
- ○ No relation

Solution: Since a student can be a member of multiple clubs and a club can have multiple members, the relationship between Student and Club is many-to-many. A club can have multiple announcements, but an announcement is only associated with a single club, so the relationship between Club and Announcement is one-to-many.

(c) In a normalized schema designed for a relational database (RDBMS), what schema would be needed to support club membership. Assume club members can have different roles, e.g., “member”, “president”, etc. You do not need to provide SQL, just the attributes, their types, the primary key, and any foreign key constraints.

Solution: Students and Clubs have a Many-to-Many relationship, which would be implemented with a join table, e.g., `Membership`. The `Membership` table would have a foreign key to the integer `id` attribute in the `User` table, a foreign key to the integer `id` attribute in the `Club` table, and a string attribute for the role (or possible a foreign key to a roles table, both versions were accepted). If we assume a student can only have one role in a club then the primary key can be a composite of the user and club foreign keys. If a student could have multiple roles, we would need a separate primary key, e.g., an auto-incrementing integer, or the primary key would need to be a composite of the user, club, and role.

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) Each team member uses a separate "personal" branch throughout the sprint, e.g., `mlinderman_sprint1`, to implement their tasks before merging with `main` at end of the sprint.

◯ Consistent [✓ **Not consistent**]

Solution: Our best practices are to use short-lived features branches for each task, not a long lived branch for the entire sprint. This action could create large, difficult merges with many conflicts. Using a single opaquely named branch would make it difficult to find the code for a specific task.

(b) Assign a responsible developer for all the tasks in the sprint backlog during the sprint planning meeting.

◯ Consistent [✓ **Not consistent**]

Solution: We are unlikely to perfectly predict how long a task will take, so "pre-assigning" all the tasks at the beginning of the sprint, even if guided by the story points, is likely to result some team members idle and others over-committed. Instead we should assign tasks as they are ready to be worked on, i.e., some tasks may not be assigned at the beginning of the sprint.

(c) Update the `main` branch and rebase a newly created feature branch before pushing that feature branch to GitHub for the first time.

[✓ **Consistent**] ◯ Not consistent

Solution: The problem was intended to describe updating the `main` branch in the *local* copy of the repository before rebasing the feature branch. Since the description was ambiguous answers that interpreted the problem as updating the `main` branch on GitHub directly were accepted.

Rebasing our feature against the latest `main` branch ensures that our feature branch is up-to-date with the latest changes in the project, and thus can merged "cleanly" back into `main` (e.g., as part of a pull request). Since the feature is new and not previously pushed to GitHub, we are not concerned with rewriting history as part of the rebase operation. Using `rebase` in this context will create a more linear history than using `merge`.