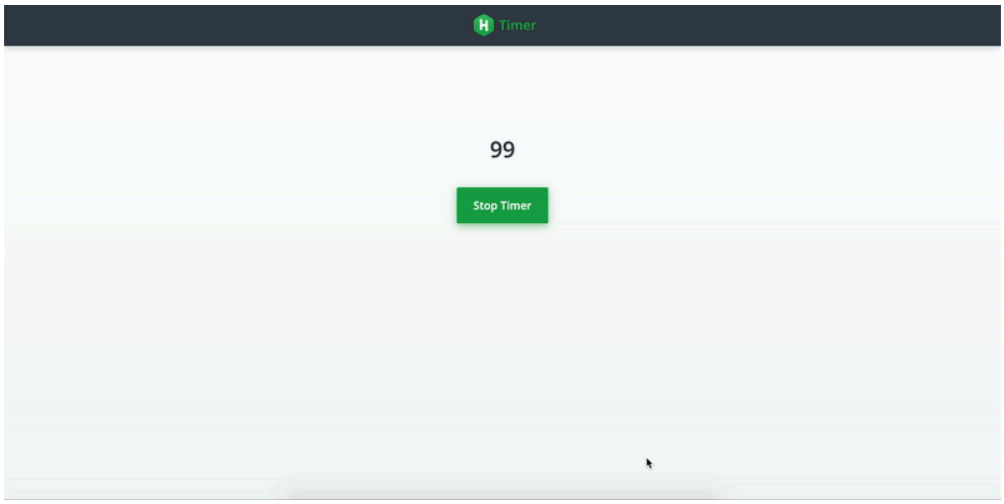
Question - 1
React(TypeScript): Timer Component

Create a timer component like the one shown.

Hide animation



The component has the following functionalities:
- The timer value decreases by 1 every second. For example, if the initial value is 100, after 1 second it becomes 99, etc.
- The value starts to decrease when the component is mounted.
- The initial value of the timer is set by a prop, *initial*.
- Once the counter reaches 0, it should stop.
- The 'Stop Timer' button stops the timer at its current value.

The following data-testid attributes are required in the component for the tests to pass:

Component	Attribute
Title	*app-title*
Timer value	*timer-value*
Stop timer button	*stop-button*

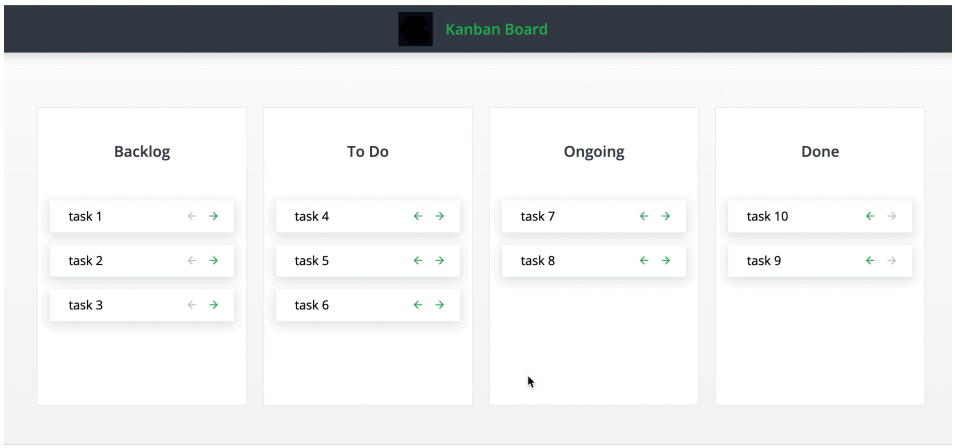
Please note that components have these *data-testid* attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.

Question - 2
React(TypeScript): Kanban Board

Kanban is a popular workflow used in task management, project management, issue tracking, and other similar purposes. The workflow is usually visualized using a Kanban Board.

Create a Kanban Board component with tasks, where each task consists of a name only, as shown below:

Hide animation



The component must have the following functionalities:

- The component board contains 4 stages of tasks in the sequence 'Backlog', 'To Do', 'Ongoing', and 'Done'.
- An array of tasks is passed as a prop to the component.
- In every individual stage, the tasks are rendered as a list <ul>, where each task is a single list item <li> that displays the name of the task.
- Each task list item has 2 icon buttons on the right:
    1. Back button: This moves the task to the previous stage in the sequence, if any. This button is disabled if the task is in the first stage.
    2. Forward button: This moves the task to the next stage in the sequence, if any. This button is disabled if the task is in the last stage.
- Each task has 2 properties:
    - name: The name of task. This is the unique identification for every task. [STRING]
    - stage: The stage of the task. [NUMBER] (0 represents the 'Backlog' stage, 1 represents the 'To Do' stage, 2 represents the 'Ongoing' stage, and 3 represents the 'Done' stage)

The following data-testid attributes are required in the component for the tests to pass:

- The <ul> for the 'Backlog' stage should have the data-testid attribute 'stage-0'.
- The <ul> for the 'To Do' stage should have the data-testid attribute 'stage-1'.
- The <ul> for the 'Ongoing' stage should have the data-testid attribute 'stage-2'.
- The <ul> for the 'Done' stage should have the data-testid attribute 'stage-3'.
- Every <li> task should follow these guidelines:
    1. The containing the name should have the data-testid attribute 'TASK_NAME-name', where TASK_NAME is the name of the task joined by a hyphen symbol. For example, for the task named 'task 0', it should be 'task-0-name'. For the task named 'abc', it should be 'abc-name'.
    2. The back button should have the data-testid attribute 'TASK_NAME-back', where TASK_NAME is the name of the task joined by a hyphen symbol. For example, for the task named 'task 0', it should be 'task-0-back'. For the task named 'abc', it should be 'abc-back'.
    3. The forward button should have the data-testid attribute 'TASK_NAME-forward', where TASK_NAME is the name of the task joined by a hyphen symbol. For example, for the task named 'task 0', it should be 'task-0-forward'. For the task named 'abc', it should be 'abc-forward'.

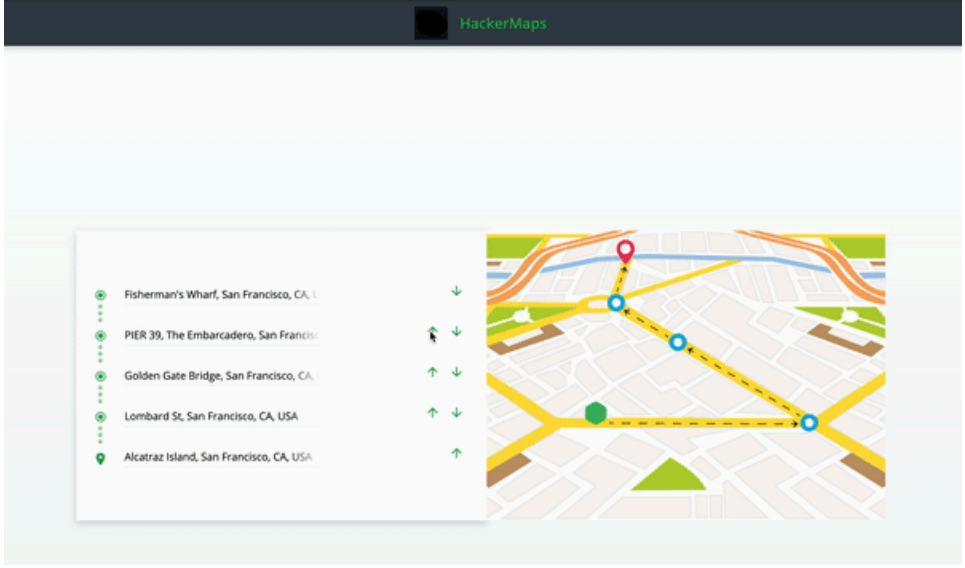
Please note that components have the above data-testid attributes for test cases and certain classes and ids for rendering purposes. It is advised not to change them.

Question - 3

React(TypeScript): HackerMaps

Create a basic navigation application, as shown below. Some core functionalities have already been implemented, but the application is not complete. Application requirements are given below, and the finished application must pass all of the unit tests.

Hide animation



The app has one component: the Navigation view. The list of locations to be displayed is already provided in the app.

The app should implement the following functionalities:

- The locations should be initially displayed in their respective <li> tags in the same order in which they are provided.
- Each location can have one or two icon buttons, depending upon its position in the list:
  - The first location should only have the Move Down icon button.
  - The last location should only have the Move Up icon button.
  - All the other locations should have both the Move Up and the Move Down buttons.
- Clicking on the Move Down button should move the location down by one position in the list.
- Similarly, clicking the Move Up button should move the location up by one position in the list.
- When a location is moved either up or down, it should exchange its position with the location positioned just above (if moving up) or below (if moving down).
- The list of locations is passed as props to the Navigation component.

The locations list is an array of strings, with each item representing a location in the list.

Note: The utility function isLast is provided to help with determining if the current location is the last item in the list. Also, the function getClasses is present in the template to aid in rendering. Please do not modify this function.

The following data-testid/class attributes are required in the component for the tests to pass:

- The parent container of the location list <ul> should have the data-testid attribute 'location-list'.
- Each location item in the list should have the data-testid attribute 'location-0', 'location-1', 'location-2', and so on.
- Each location name paragraph tag <p> should have the data-testid attribute 'location'.
- Each Move Up button should have the data-testid attribute 'up-button'.
- Each Move Down button should have the data-testid attribute 'down-button'.

Please note that the component has the above data-testid attributes for test cases and certain classes and ids for rendering purposes. It is advised not to change them.

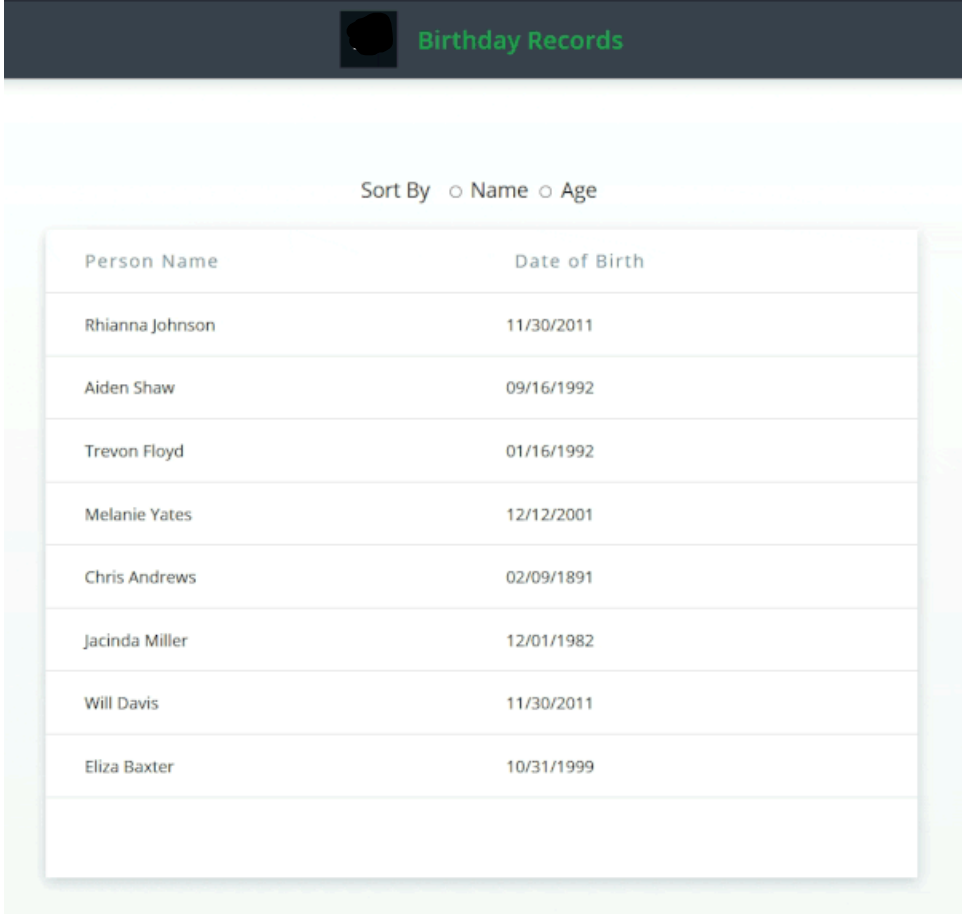
Question - 4

React(TypeScript): Birthday Records

Complete a Birthday Records component that sorts a list of people by name or by age.

Certain core React functionalities have already been implemented. Complete the React application as shown below in order to pass all the unit tests.

Hide animation



The component has the following functionalities:

- The application's initial state displays a table where the unsorted list of people records must be rendered.
- The radio buttons to sort by 'Name' or 'Age' are toggled when clicked, so that only one is selected at a time.
- Clicking the radio button to sort by 'Name' must sort the list by *ascending name*.
- Clicking the radio button to sort by 'Age' must sort the list by *descending age* (i.e., *ascending date*).

The following data-testid attributes are required in the component for the tests to pass:

- The radio button to sort by 'Name' should have the data-testid attribute 'name'.
- The radio button to sort by 'Age' should have the data-testid attribute 'age'.
- The table displaying the names and dates should have the data-testid attribute 'table'.

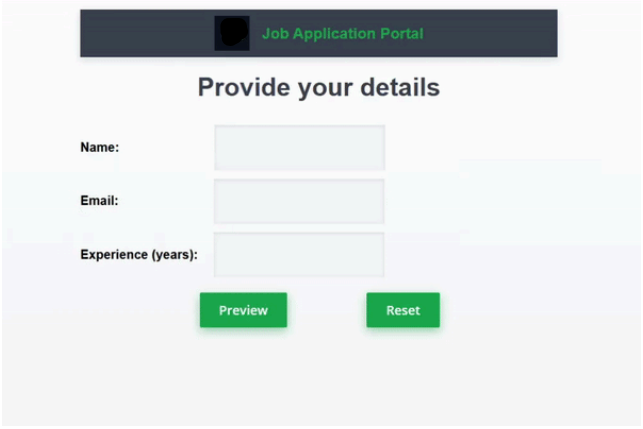
Please note that component has the above data-testid attributes for test cases and certain classes and ids for rendering purposes. It is advised not to change them.

Question - 5

React: Job Application Portal

In this application, the task is to build a Job Application Portal where users can fill out and submit their job applications. Edit the *App.js*, *JobApplicationForm.js*, *Preview.js* and *SuccessMessage.js* to ensure the application functions as described below:

Hide animation



Functionality Requirements:

1. Initial State of the project:
   - The form should have input fields for:
     - Name (required)
     - Email (required, valid email format) : Ex: test@email.com, is a valid email.
     - Experience (required, numeric input).
   - Input fields must be initially empty.
2. Preview Functionality:
   - After filling the form, users can click the Preview button to review their inputs.
   - The preview should display the entered name, email, and experience.
3. Clear Functionality:
   - In the preview, clicking the Clear button should take the user back to the form, with all the previously entered values cleared.
   - Validation:
     - Display the following error message if:
       - Name is empty: "Name is required."
       - Email is empty or invalid: "Enter a valid email address."
       - Experience is not a positive integer: "Experience must be a positive number."
     - The Preview button should not perform any operation if any field is empty or invalid.
   - Submission:
     - Upon clicking the Submit button in the preview, show a success message indicating that the application was successfully submitted.
   - Reset and Home Functionality:
     - A Reset button in the form and Home button in the success message:
       - Should clear all inputs and error messages.
       - Return the user to the initial state of the application.

Note:

The following data-testid attributes are required in the components for the test cases to pass, do not change/delete them:

Table of data-testid

data-testid	Description
app	container representing entire application
job-application-form	The form where users can enter their application details.
input-name	Input box to enter the applicant's name.
error-name	Displays the error message related to name.
input-email	Input box to enter the applicant's email address.
error-email	Displays the error message related to email.
input-experience	Input box to enter the applicant's years of experience.
error-experience	Displays the error message related to experience.
preview-button	Button to preview the application data.

reset-button	Button to reset the form or success message.
preview	The preview section displaying the entered details.
preview-name	The preview text displaying the entered name.
preview-email	The preview text displaying the entered email.
preview-experience	The preview text displaying the entered experience.
clear-button	Button in the preview to return to the form.
submit-button	Button to submit the application from the preview.
success-message	The message displayed upon successful submission of the form.