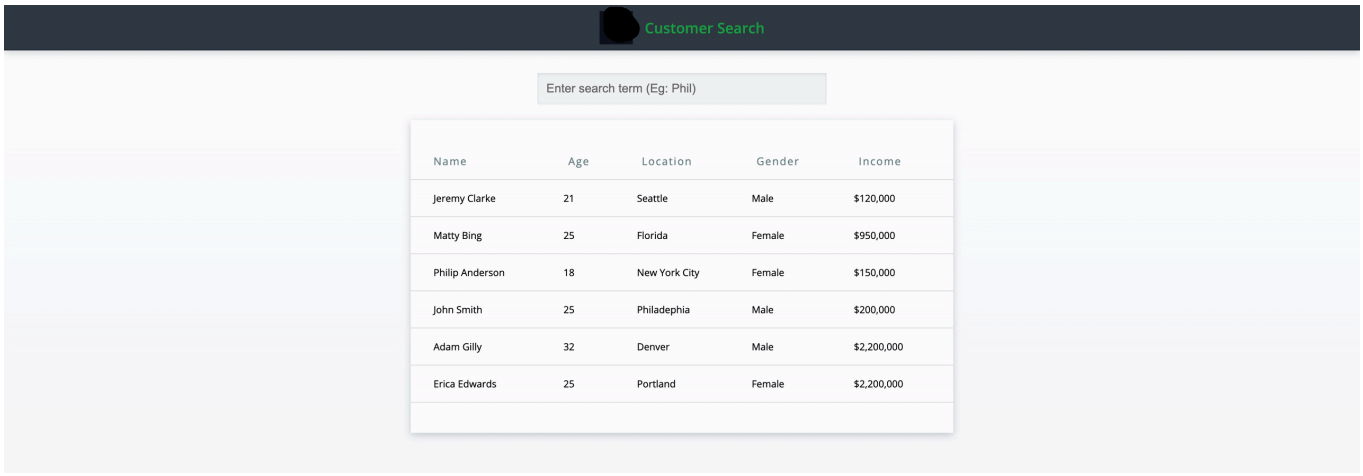
Question - 1

React: Customer Search

Complete the component as shown to pass all the test cases. Certain core React functionalities are already implemented.

Hide animation



The list of available customers and their details is imported as `List` in the SearchCustomer component. Use the list to render them as shown.

The data in List file is in this form.

```
{
   name: "Jeremy Clarke",
   age: 21,
   location: "Seattle",
   gender: "Male",
   income: "$120,000"
}
```

The application has 2 components:

- `SearchCustomer` - contains and search box and CustomerList component.
- `CustomerList` - displays the details of the customer based on the search term in a table.

The component must have the following functionalities:

- Initially, the input field must be empty. Whenever the input is empty, all the customers passed in the input to the component must be rendered in the list.
- The type of input in the input box should be `text`.
- As soon as the search term is typed in the input, search for customer records with any field that starts with the search term. For example, if the search term is "Phil", then records with the name "Philip Anderson" and the location "Philadelphia" should be displayed in the results.
- The filtered list should preserve the order customers are given in the input to the component.
- If the search term returns no customers, do not render the table. Instead, display the message "No Results Found!".
- The search results should be case-sensitive.

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

Attribute	Component

search-input	Search Input Box
searched-customers	List of searched customers
no-results	div when no customer matches the search term

Note:

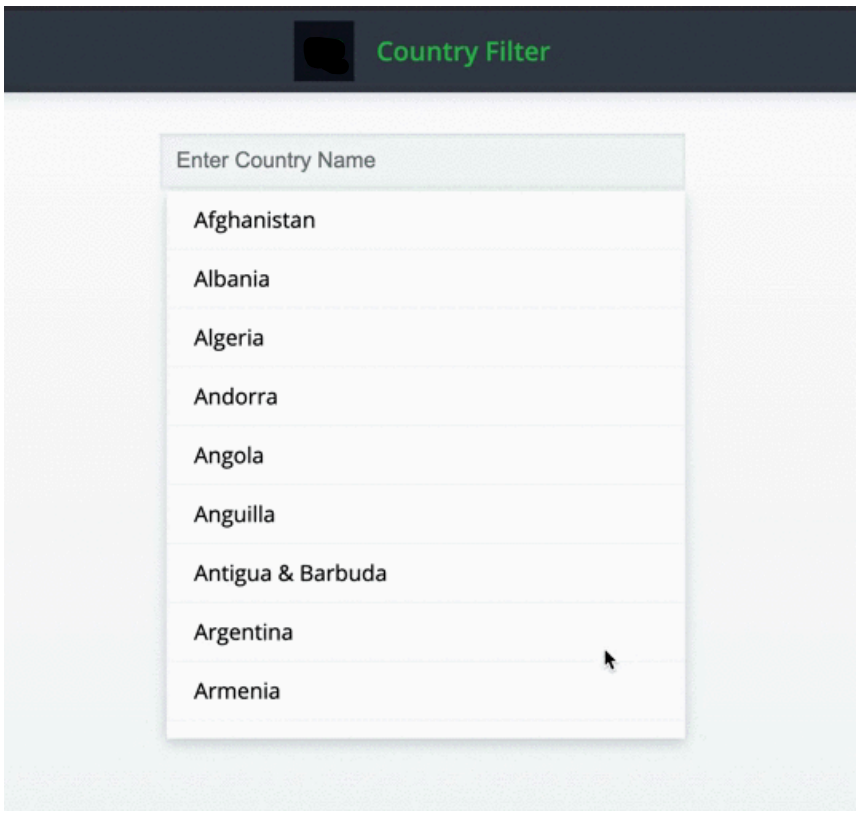
- Components have data-testid attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The files that should be modified by the candidate are `src/components/CustomerList.js` and `src/components/SearchCustomer.js`. They are open by default in the system editor.
- Avoid making changes to other files in the project structure.

Question - 2

React: Country Filter

Given a partially completed React application with the HTML template built and ready, implement a filter that searches and displays matching countries in a list.

Hide animation



The application has 2 components:

1. The `Search` view, which has the input box where the user can type to filter countries.
2. The `CountryList` view, which renders the list of countries that are filtered.

The entire list of countries to display is stored in a variable named `response` inside the file `src/response.js`.

The app should implement the following functionalities:

- The initial view should render an empty input box with all countries appearing in the same order as in the `response` variable.
- When a character is typed, each country containing the set of typed characters should be filtered and displayed.
- The filtering should be case insensitive.
- When there is no character typed in the search box, it should show all countries.

Note:

- Get the list of countries to be displayed from the `response.js` file.
- The order in which each country appears should follow the same order in the `response.js` file, even after filtering.

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

- The country filter field <input> tag should have the data-testid attribute 'filterInput'.
- The countries wrapper <ul> tag should have the data-testid attribute 'countryList'.

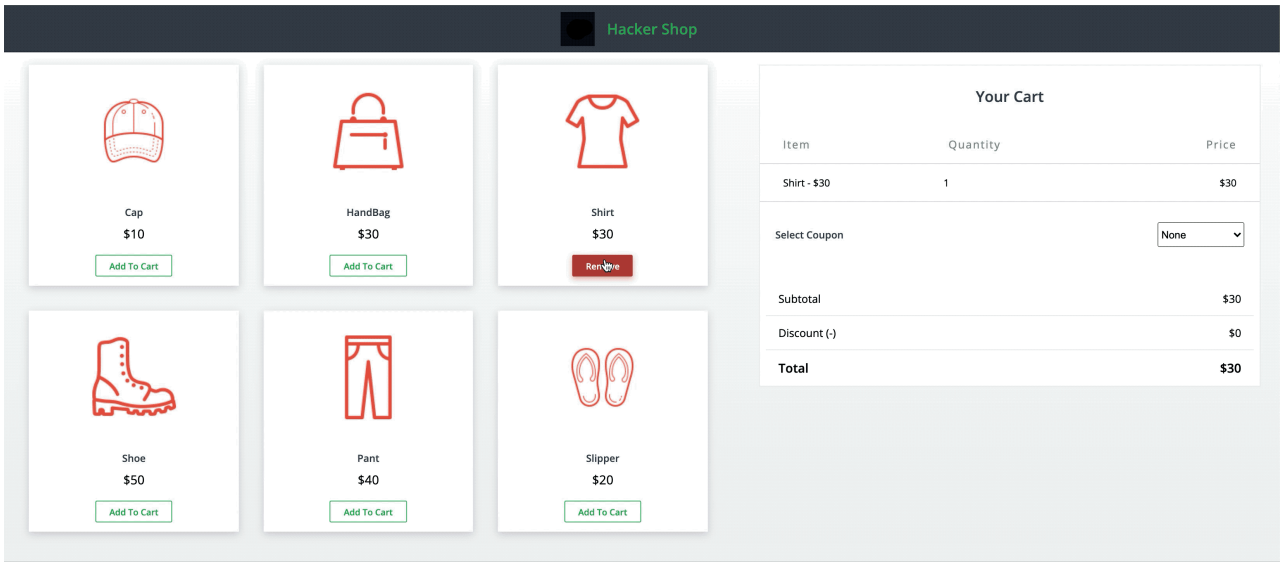
Please note that the component has these data-testid attributes for test cases. They should not be changed.

Question - 3

React: HackerShop Checkout

Create a basic shopping 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 two separate views/components: the Product Listing component and the Cart component. The list of products to be displayed is already provided in the app.

The app should implement the following functionalities:

- Clicking on each 'Add To Cart' button should add the item to the shopping cart. When an item is added to the cart:
  - The 'Add To Cart' button should be removed from view, and the 'Remove From Cart' button should be displayed.
  - An entry should be added to the table in the Cart component.
- Clicking on each 'Remove' button should remove the item from the cart and display 'Add to Cart' for the product item.
- The Cart component should have the following functionalities:
  - Display all the items in the cart in a table.
  - Display the cart's subtotal, discount value, and total price.
  - The cart has a 'Select Coupon' input. On selecting a coupon from this input, an appropriate discount is applied and the total price is calculated and displayed. (Subtotal - Discount = Total Price)
- Items should be displayed in the Cart component in the order they are added to the cart.
- The list of products and the cart object are passed as props to the Product Listing component and the Cart component respectively.

Each product object contains the following properties:

- name: Name of the product. [STRING]
- price: Price of the product. [NUMBER]

- id: Unique ID of the product. (Auto Generated) [NUMBER]
- image: The image URL of the product. [STRING]
- cartQuantity: The quantity of the item in the cart. The default value should be 0. [NUMBER]

Each item in the cart, CartItem, has the following properties:

- id: The ID of the product added to the cart. [NUMBER]
- item: The heading property of the product. [STRING]
- quantity: The quantity of the item in the cart. [NUMBER]
- price: The total price of the item in the cart. (quantity x product.price) [NUMBER]

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

- Each product item in the Product Listing component should have the class 'product-item'.
- Each 'Add to Cart' button should have the data-testid attribute 'btn-item-add'.
- Each 'Remove' button should have the data-testid attribute 'btn-item-remove'.
- The table rows <tr> in the Cart component, corresponding to items in the cart, should have the data-testid attribute of 'cart-item-0', 'cart-item-1', and so on.
- The table data <td>, containing the name of the item in the cart, should have the data-testid attribute 'cart-item-name'.
- The table data <td>, containing the quantity of the item in the cart, should have the data-testid attribute 'cart-item-quantity'.
- The table data <td>, containing the price of the item in the cart, should have the data-testid attribute 'cart-item-price'.
- The 'Select Coupon' input should have the data-testid attribute 'cart-coupon'
- The cart's 'Subtotal' value container should have the data-testid 'cart-subtotal'.
- The cart's 'Discount' value container should have the data-testid 'cart-discount'.
- The cart's 'Total Price' value container should have the data-testid 'cart-total'.

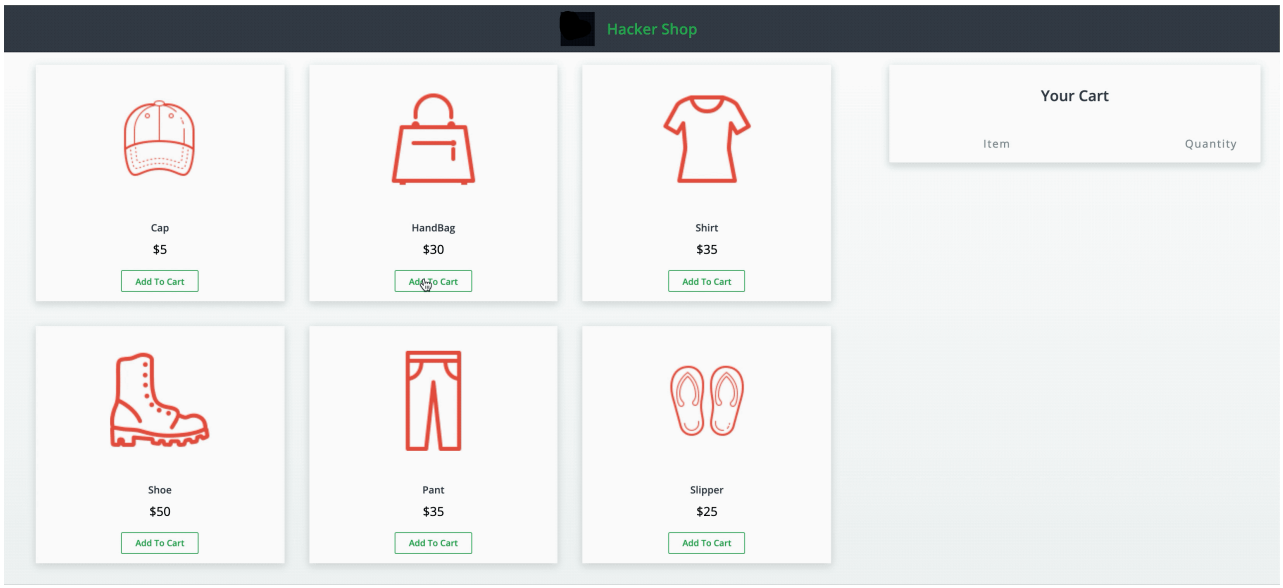
Please note that the component has the above data-testid attributes for test cases and certain classes and ids for rendering purposes. It is advised not to change them.

Question - 4

React: HackerShop Shopping Cart

Create a basic shopping 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 two separate views/components: the Product Listing component and the Cart component. The list of products to be displayed is already provided in the app.

The app should implement the following functionalities:

- Clicking on each 'Add To Cart' button should add the item to the shopping cart. The listing in the Product Listing component should be updated to show the 'Increase/Decrease Quantity' button and the quantity of the item in the cart.
- Clicking on each 'Increase Quantity' button should increase the quantity of the item in the cart.
- Clicking on each 'Decrease Quantity' button should do the following:
  - When the cart quantity of the item is 1: This should remove the item from the cart, hide the 'Increase/Decrease Quantity' button, and show the 'Add to Cart' button.
  - When the cart quantity is greater than 1: The quantity of the item in the cart should be decreased.
- On every quantity update operation, the text for the quantity of the item should be updated both in the Product Listing component as well as in the corresponding entry in the Cart component.
- Items should be displayed in the Cart component in the order they are added to the cart.
- The list of products and the cart object are passed as props to the Product Listing component and Cart component respectively.

Each product object contains the following properties:

- name: Name of the product. [STRING]
- price: Price of the product. [NUMBER]
- id: Unique ID of the product. (Auto Generated) [NUMBER]
- image: The image URL of the product. [STRING]
- cartQuantity: The quantity of the item in the cart. The default value should be 0. [NUMBER]

Each item in the cart, CartItem, has the following properties:

- id: The ID of the product added to the cart. [NUMBER]
- item: The name of the product added to the cart. [STRING]
- quantity: The quantity of the item in the cart [NUMBER]

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

- Each product item in the Product Listing component should have the data-testid attribute of 'product-item-0', 'product-item-1', and so on.
- Each 'Add to Cart' button should have the data-testid attribute 'btn-item-add'.
- Each 'Increase Quantity' button should have the data-testid attribute 'btn-quantity-add'.
- Each 'Decrease Quantity' button should have the data-testid attribute 'btn-quantity-subtract'.
- Each input to display the cart quantity in the Product Listing component should have the data-testid attribute 'cart-quantity'.
- The table rows <tr> in the Cart component, corresponding to items in the cart, should have the data-testid attribute as 'cart-item-0', 'cart-item-1', and so on.
- The table data <td>, containing the name of the item in the cart, should have the data-testid attribute 'cart-item-name'.
- The table data <td>, containing the quantity of the item in the cart, should have the data-testid attribute 'cart-item-quantity'.

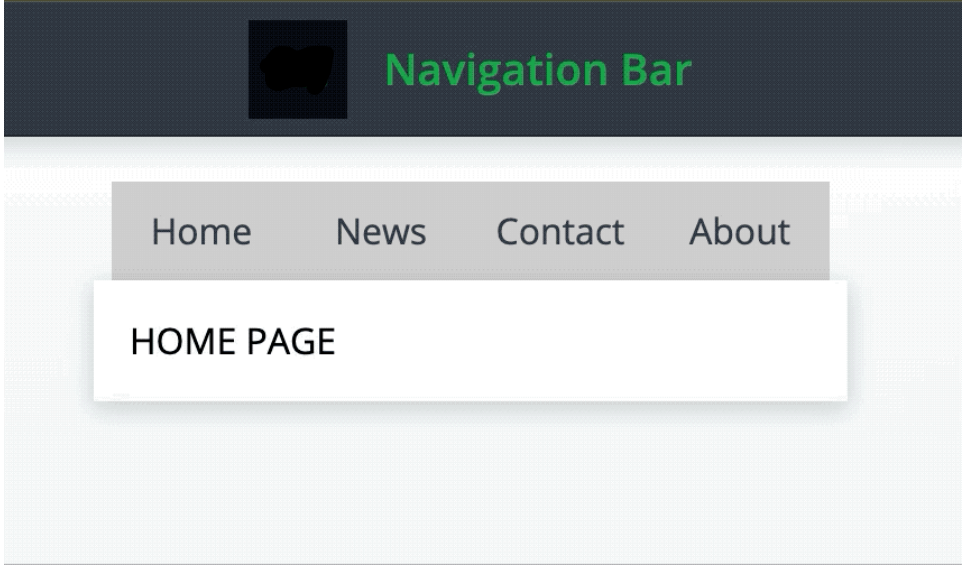
Please note that the component has the above data-testid attributes for test cases and certain classes and ids for rendering purposes. It is advised not to change them.

Question - 5

React: Navigation Bar Component

Create a navigation bar component as shown below.

Hide animation



The component has the following functionalities:

- There are 4 tabs: Home, News, Contact, and About.
- The default selected tab is the 'Home' tab.
- Clicking on a tab renders the relevant content.
    - Clicking on the Home tab renders content 'HOME PAGE'.
    - Clicking on the News tab renders content 'NEWS PAGE'.
    - Clicking on the Contact tab renders content 'CONTACT PAGE'.
    - Clicking on the About tab renders content 'ABOUT PAGE'.
- Since the default selected tab is the 'Home' tab, the default displayed content is 'HOME PAGE'.

Please note that components have some classes and ids for rendering purposes. It also has the following *data-testid* attributes for test cases.

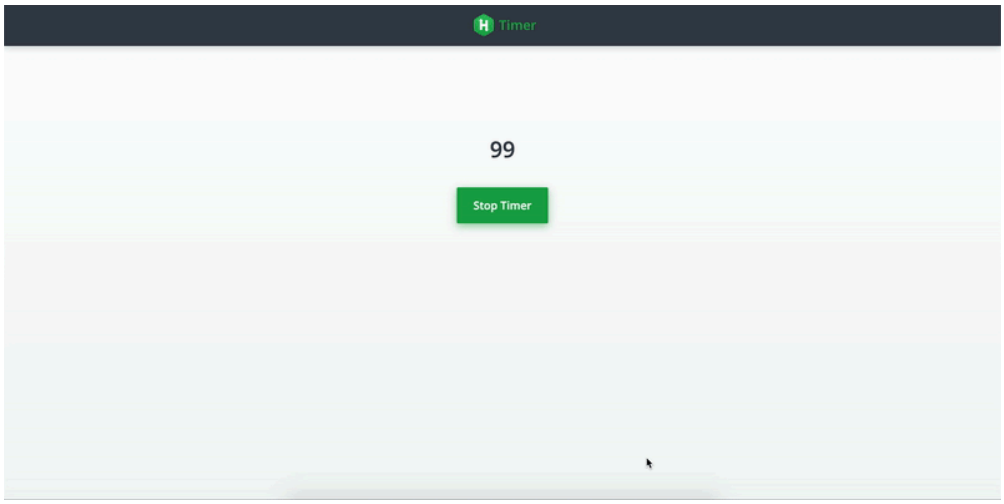
- The parents of 4 tabs have data-testid attribute 'navigation-tabs'.
- The content sections have data-testid attribute 'tab-content'.

It is not advised to change any of these classes, ids, or data-testid attributes.

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*

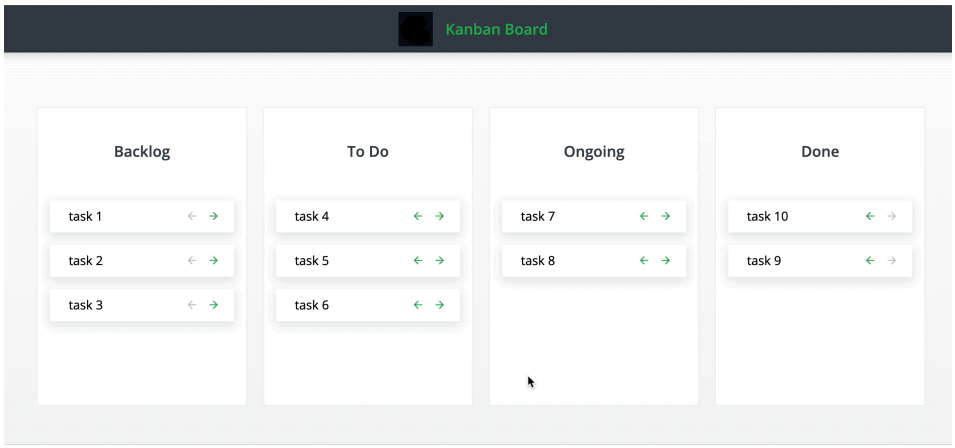
Please note that components have these *data-testid* attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.

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'.

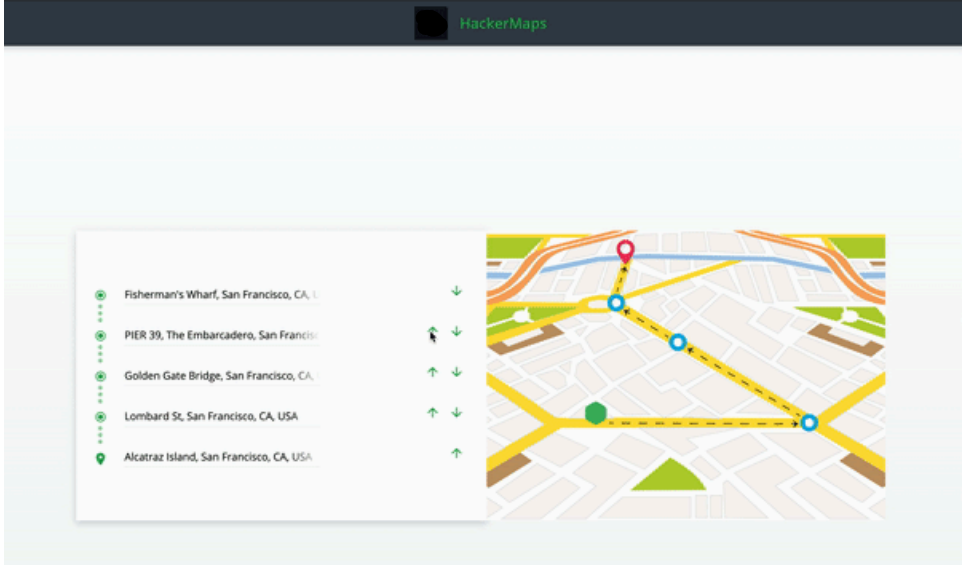
Please note that components have the above data-testid attributes for test cases and certain classes and ids for rendering purposes. It is advised not to change them.

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.

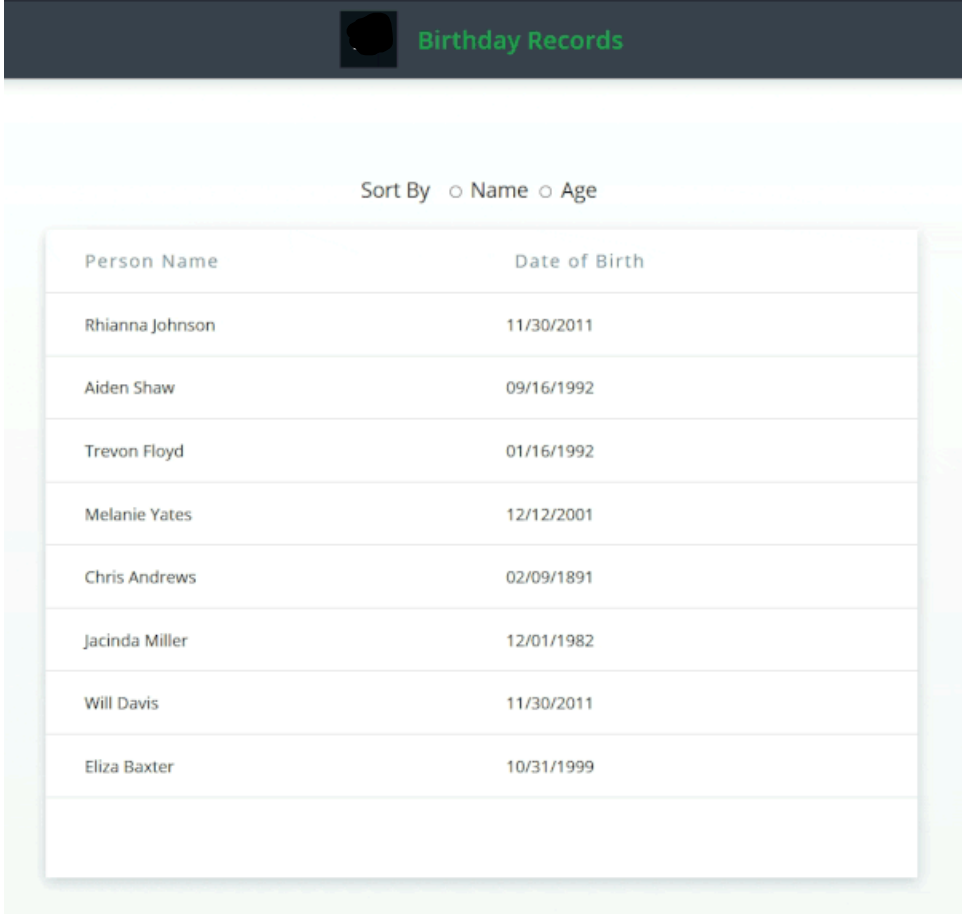
Question - 4

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'.

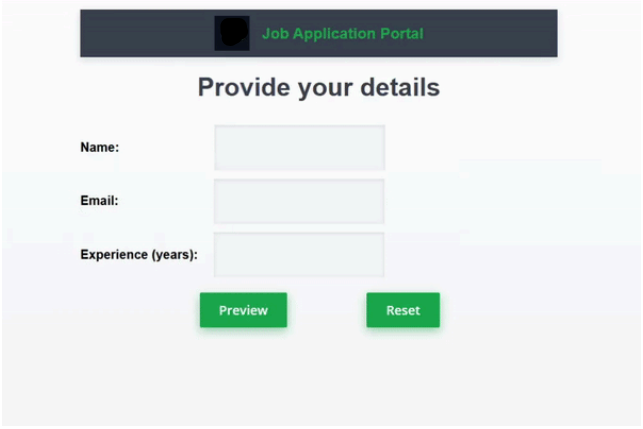
Please note that component has the above data-testid attributes for test cases and certain classes and ids for rendering purposes. It is advised not to change them.

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:

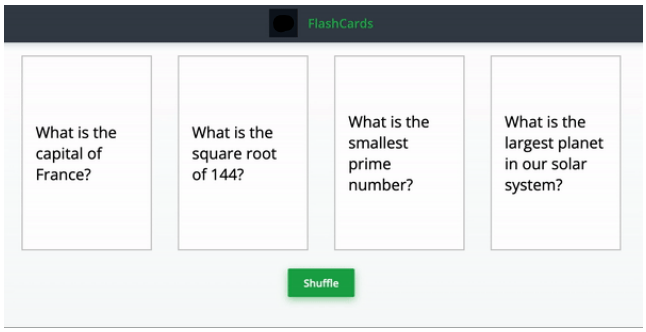
Table of data-testid

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.

Question - 1
React (Typescript): Flashcards

Hide animation



Using Typescript and React, design a flashcard app that displays a series of flashcards with questions on the front and answers on the back. Certain core React functionalities are already implemented.

The application has two components: *FlashCardDeck.tsx* and *FlashCard.tsx* where the functionalities should be implemented.

The component must have the following functionalities:
- Display a series of flashcards with questions on the front and answers on the back.
- Clicking a flashcard should flip it to reveal the other side.
- Update the *isFlipped* constant to a state variable in *FlashCard/index.tsx*:
  - When *isFlipped* is true
    - *flipped* is appended to divs with the class name *flashcard-content*.
    - the answer is shown.
  - When *isFlipped* is false
    - '' is appended to divs with the class name *flashcard-content*.
    - the question is shown.
- Clicking the *Shuffle* button should reorder the flashcards.
  - Ensure the shuffled order of the flashcards is different from the original order present in *src/data/cards-data.ts*.
- All the types are defined under file *src/types/FlashCard.ts*

Create a type for *FlashCard* in *src/types/FlashCard.ts* with the following properties:
- *id,* a *number*
- *question*, a *string*
- *answer*, a *string*

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

Attribute	Component
flashcard-deck	the main FlashCardApp component
flashcard-container-{card.id}	an individual FlashCard
flashcard-question-{card.id}	the question text in a FlashCard
flashcard-answer-{card.id}	the answer text in FlashCard

shuffle-button	a button to shuffle the flashcards

Note:
- Components have *data-testid* attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The files that should be modified by the candidate are *src/components/FlashCardDeck.tsx*, *src/components/FlashCard.tsx*, and *src/types/FlashCard.ts*.

Question - 2
React: Image Preview

Complete a React image preview application as shown below to pass all the unit tests. Certain core React functionalities are already implemented.

Show animation

The application has two components:
- The ImagePreview component that allows users to view and hide images by clicking on them.
- The HiddenImageDiv component that should be rendered when an image is to be hidden,

The image data is passed to the component as "images" to render in the component.

The application has the following functionalities:
- The ImagePreview component renders the following conditionally :
  - If the *visible* attribute is true, then the image with the *src* and *alt* attributes passed in the image data.
  - If the *visible* attribute is false, then the *HiddenImageDiv* component
- Clicking an image should hide it from the DOM, and the *HiddenImageDiv* should be visible in its place.
- Clicking the *HiddenImageDiv* component should render the original image again.
- On clicking the *ShowAll* button, all the images should be visible, and any *HiddenImageDiv* component should be hidden.
- On clicking the *HideAll* button, all the images should be hidden, and only the *HiddenImageDiv* components should be visible.

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

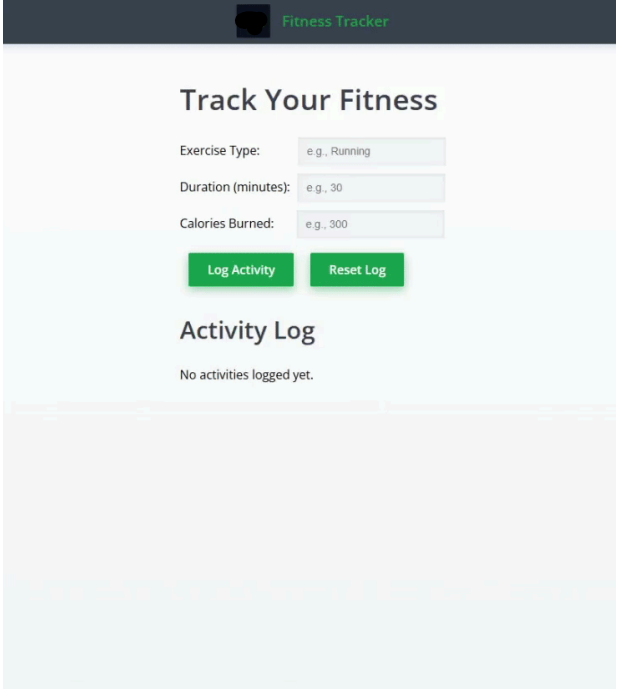
Attribute	Component
images-div	Images display div
show-all-btn	Show all images button
hide-all-btn	Hide all images button

Note:
- Components have *data-testid* attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The only file that should be modified by the candidate is the src/components/ImagePreview.js.

Question - 3
React: FitTrack Pro

In this application, the task is to create a fitness tracker that allows users to log their daily activities, including exercise type, duration, and calories burned. Edit the *FitnessTracker.js*, *LogForm.js,* and *LogList.js* within the project structure to ensure the application functions as described below:

Hide animation



Functionality Requirements:

1. Initial State of the project:
   - The exercise-type, duration, and calories-burned input fields should be empty initially.
   - The exercise-type input field should be of type text.
   - The duration and calories-burned input fields should be of type number.
   - The *Log Activity* button should not add any log until all fields have valid inputs.
2. Logging functionality:
   - The *Log Activity* button should add a log only when all fields have valid values.
   - The duration and calories burned should be non-zero and positive numbers.
   - On clicking the *Log Activity* button:
     - Add a new entry in the log table, displaying the *exercise-type*, *duration* (in minutes), and *calories-burned*.
     - Clear the input fields after successfully adding the entry.
   - On clicking the *Reset Log* button:
     - All logs are cleared and reset to an empty list.
     - The error message should also be cleared.
3. Log Display:
   - All log entries should appear within the application, under the *Activity Log* heading.
   - Each log entry, displays:
     - Exercise Type: the type of exercise logged
     - Duration: the time spent in minutes
     - Calories Burned: the calories burned during the activity
4. Error Handling:
   - Display the error message, if the user tries to submit invalid data, such as empty fields or non-positive numbers.
     - "Exercise type must not be empty." : If the exercise type input is empty.
     - "Duration must be a positive number.": If the duration input is empty or non-positive.
     - "Calories must be a positive number.": If the calories burned input is empty or non-positive.

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
input-exerciseType	Input box to enter the type of exercise
input-duration	Input box to enter the duration of the activity (in minutes)
input-caloriesBurned	Input box to enter the calories burned

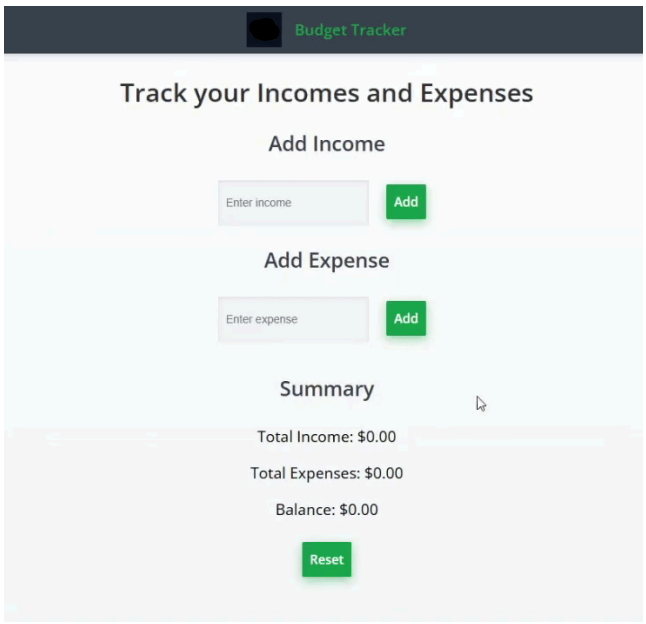
btn-logActivity	Button to log the activity
btn-resetLog	Button to reset all the activities
log-list	List of all the logs
log-entry	To display the user's logged activities
error-message	Error message for displaying the invalid inputs

Question - 4

React: Budget Balance Assistant

In this application, the task is to build a budget tracker that helps users manage their income and expenses effectively. Edit the *BudgetTracker.js*, *IncomeForm.js*, *ExpenseForm.js* and *Summary.js* components within the project structure to ensure the application functions as described below:

Hide animation



Functionality Requirements:

1. Initial State of the project:
   - The income and expense input fields should be empty and must have the attribute type as number.
   - Ensure that the total income, total expenses, and balance are initialized to 0.
2. Validation:
   - Positive Numbers Only:
     - The income and expense values must be positive. Negative or zero values should not be added.
   - Empty Input Handling:
     - No update to the total income, expense and balance, if non-positive income or expense is added.
3. Income and Expense Functionality:
   - Add Income: When the user enters a positive number in the income input box and clicks the "Add" button:
     - The value is added to the total income.
     - The balance is updated as total income - total expenses.
     - The income input box is cleared.
   - Add Expense: When the user enters a positive number in the expense input box and clicks the "Add" button:
     - The value is added to the total expenses.
     - The balance is updated as total income - total expenses.
     - The expense input box is cleared.
4. Reset Functionality:
   - Add a "Reset" button which resets the Total Income, Total Expenses, and Balance to 0.
   - Clicking the "Reset" button ensures the application returns to its initial state.

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

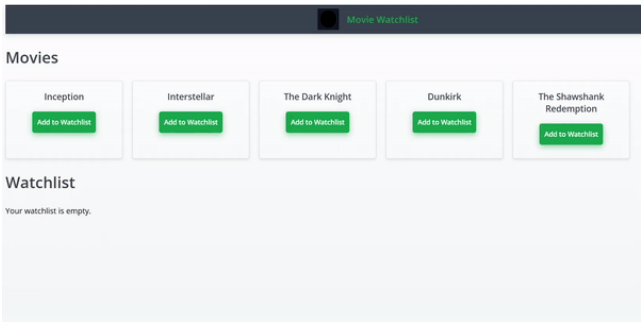
Table of data-testid

data-testid	Description
income-input	Input box to enter the income amount.
add-income-btn	Button to add the entered income.
expense-input	Input box to enter the expense amount.
add-expense-btn	Button to add the entered expense.
total-income	Text displaying the total income value.
total-expenses	Text displaying the total expenses value.
balance	Text displaying the current balance (income-expense).
reset-btn	Button to reset the income, expense, and balance.

Question - 5
React: Movie Watchlist

In this application, the task is to manage a user's movie watchlist. Edit the *Watchlist.js*, *WatchlistApp.js*, and *MovieCard.j*s components within the project structure to ensure the application functions as described below:

Hide animation



Functionality Requirements:

1. Initial State of the project:
   - All movies should be displayed as cards in the "Movies" section, each with an "Add to Watchlist" button. The MovieCard component is used to generate a card for movie.
   - The watchlist section should be empty initially, displaying a message: "Your watchlist is empty."
   - The "Add to Watchlist" button should be enabled for all movies by default.
2. Watchlist Addition Functionality:
   - When the "Add to Watchlist" button is clicked on a movie, the movie should be added to the watchlist.
   - The "Add to Watchlist" button should be disabled for movies already in the watchlist, and it's text should be changed to "Added".
3. Watchlist Display Functionality:
   - The watchlist should display movies as cards, similar to the main movie section, with each card containing a "Remove" button, using the same MovieCard component.
   - Clicking the "Remove" button should remove the movie from the watchlist, and the corresponding "Add to Watchlist" button should be re-enabled.
4. Clear Watchlist Functionality:
   - A "Clear Watchlist" button should be displayed in the watchlist section if there are movies in the watchlist.
   - Clicking the "Clear Watchlist" button should remove all movies from the watchlist and display the message: "Your watchlist is empty."
5. Movie Count Display:
   - The total number of movies in the watchlist should be displayed in a badge/icon near the watchlist title. For example: "Watchlist (2)".

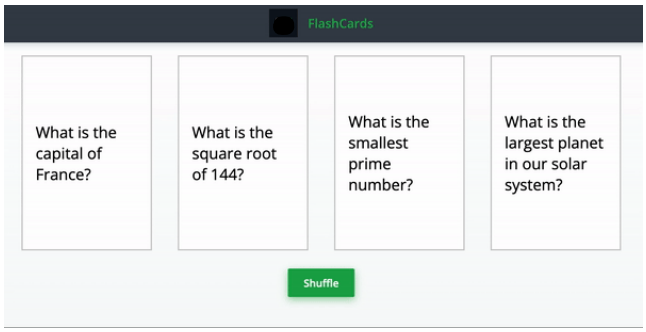
Note:
The following data-testid attributes are required in the components for the test cases to pass, do not change/delete them:

Table of data-testid

data-testid	Description
movie-card-{id}	Movie card displaying the movie alongwith Remove button.
movie-title-{id}	The title of the movie displayed in either the movie list or watchlist.
add-btn-{id}	Button to add the movie with the specified ID to the watchlist.
remove-btn-{id}	Button to remove the movie with the specified ID from the watchlist.
watchlist-container	Displays all the movies in the watchlist alongwith its count.
watchlist-empty	Message indicating that the watchlist is empty.
movie-container	Displays all the movies in the watchlist.
clear-watchlist-btn	Button to clear all movies from the watchlist.

Question - 1
React (Typescript): Flashcards

Hide animation



Using Typescript and React, design a flashcard app that displays a series of flashcards with questions on the front and answers on the back. Certain core React functionalities are already implemented.

The application has two components: *FlashCardDeck.tsx* and *FlashCard.tsx* where the functionalities should be implemented.

The component must have the following functionalities:
- Display a series of flashcards with questions on the front and answers on the back.
- Clicking a flashcard should flip it to reveal the other side.
- Update the *isFlipped* constant to a state variable in *FlashCard/index.tsx*:
  - When *isFlipped* is true
    - *flipped* is appended to divs with the class name *flashcard-content*.
    - the answer is shown.
  - When *isFlipped* is false
    - '' is appended to divs with the class name *flashcard-content*.
    - the question is shown.
- Clicking the *Shuffle* button should reorder the flashcards.
  - Ensure the shuffled order of the flashcards is different from the original order present in *src/data/cards-data.ts*.
- All the types are defined under file *src/types/FlashCard.ts*

Create a type for *FlashCard* in *src/types/FlashCard.ts* with the following properties:
- *id,* a *number*
- *question*, a *string*
- *answer*, a *string*

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

Attribute	Component
flashcard-deck	the main FlashCardApp component
flashcard-container-{card.id}	an individual FlashCard
flashcard-question-{card.id}	the question text in a FlashCard
flashcard-answer-{card.id}	the answer text in FlashCard

shuffle-button	a button to shuffle the flashcards

Note:
- Components have *data-testid* attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The files that should be modified by the candidate are *src/components/FlashCardDeck.tsx*, *src/components/FlashCard.tsx*, and *src/types/FlashCard.ts*.

Question - 2
React: Image Preview

Complete a React image preview application as shown below to pass all the unit tests. Certain core React functionalities are already implemented.

Show animation

The application has two components:
- The ImagePreview component that allows users to view and hide images by clicking on them.
- The HiddenImageDiv component that should be rendered when an image is to be hidden,

The image data is passed to the component as "images" to render in the component.

The application has the following functionalities:
- The ImagePreview component renders the following conditionally :
  - If the *visible* attribute is true, then the image with the *src* and *alt* attributes passed in the image data.
  - If the *visible* attribute is false, then the *HiddenImageDiv* component
- Clicking an image should hide it from the DOM, and the *HiddenImageDiv* should be visible in its place.
- Clicking the *HiddenImageDiv* component should render the original image again.
- On clicking the *ShowAll* button, all the images should be visible, and any *HiddenImageDiv* component should be hidden.
- On clicking the *HideAll* button, all the images should be hidden, and only the *HiddenImageDiv* components should be visible.

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

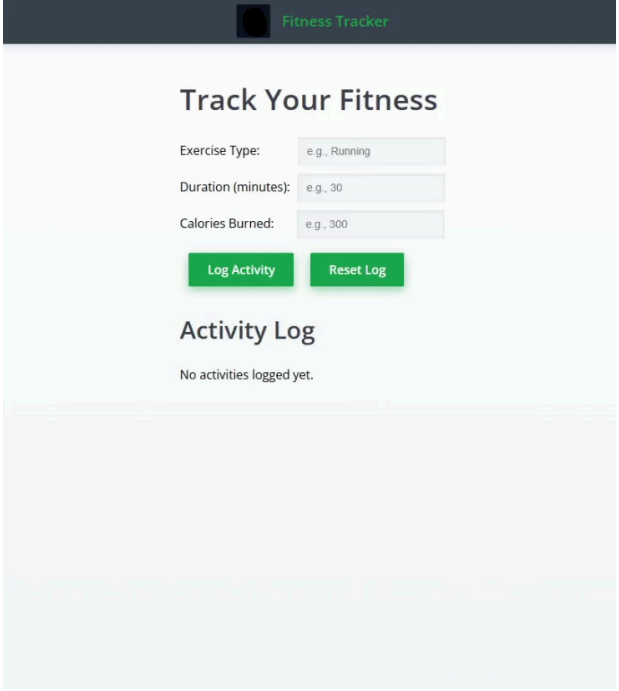
Attribute	Component
images-div	Images display div
show-all-btn	Show all images button
hide-all-btn	Hide all images button

Note:
- Components have *data-testid* attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The only file that should be modified by the candidate is the src/components/ImagePreview.js.

Question - 3
React: FitTrack Pro

In this application, the task is to create a fitness tracker that allows users to log their daily activities, including exercise type, duration, and calories burned. Edit the *FitnessTracker.js*, *LogForm.js,* and *LogList.js* within the project structure to ensure the application functions as described below:

Hide animation



Functionality Requirements:

1. Initial State of the project:
   - The exercise-type, duration, and calories-burned input fields should be empty initially.
   - The exercise-type input field should be of type text.
   - The duration and calories-burned input fields should be of type number.
   - The *Log Activity* button should not add any log until all fields have valid inputs.
2. Logging functionality:
   - The *Log Activity* button should add a log only when all fields have valid values.
   - The duration and calories burned should be non-zero and positive numbers.
   - On clicking the *Log Activity* button:
     - Add a new entry in the log table, displaying the *exercise-type*, *duration* (in minutes), and *calories-burned*.
     - Clear the input fields after successfully adding the entry.
   - On clicking the *Reset Log* button:
     - All logs are cleared and reset to an empty list.
     - The error message should also be cleared.
3. Log Display:
   - All log entries should appear within the application, under the *Activity Log* heading.
   - Each log entry, displays:
     - Exercise Type: the type of exercise logged
     - Duration: the time spent in minutes
     - Calories Burned: the calories burned during the activity
4. Error Handling:
   - Display the error message, if the user tries to submit invalid data, such as empty fields or non-positive numbers.
     - "Exercise type must not be empty." : If the exercise type input is empty.
     - "Duration must be a positive number.": If the duration input is empty or non-positive.
     - "Calories must be a positive number.": If the calories burned input is empty or non-positive.

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
input-exerciseType	Input box to enter the type of exercise
input-duration	Input box to enter the duration of the activity (in minutes)
input-caloriesBurned	Input box to enter the calories burned

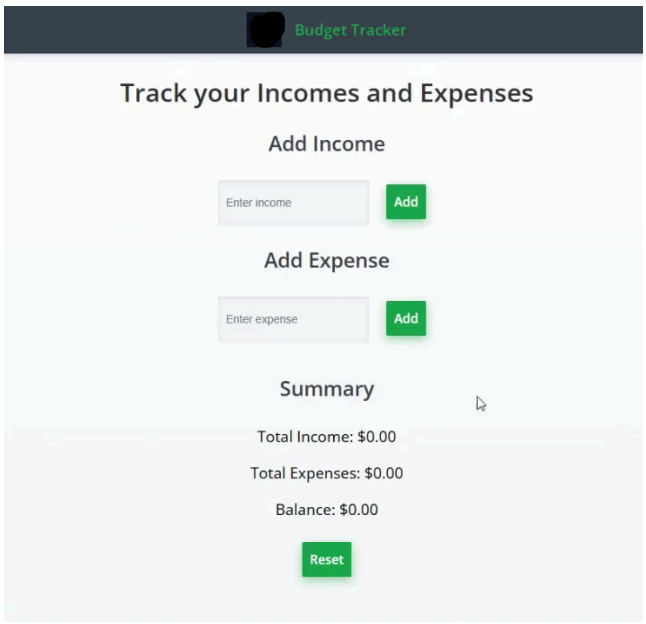
btn-logActivity	Button to log the activity
btn-resetLog	Button to reset all the activities
log-list	List of all the logs
log-entry	To display the user's logged activities
error-message	Error message for displaying the invalid inputs

Question - 4

React: Budget Balance Assistant

In this application, the task is to build a budget tracker that helps users manage their income and expenses effectively. Edit the *BudgetTracker.js*, *IncomeForm.js*, *ExpenseForm.js* and *Summary.js* components within the project structure to ensure the application functions as described below:

Hide animation



Functionality Requirements:

1. Initial State of the project:
   - The income and expense input fields should be empty and must have the attribute type as number.
   - Ensure that the total income, total expenses, and balance are initialized to 0.
2. Validation:
   - Positive Numbers Only:
     - The income and expense values must be positive. Negative or zero values should not be added.
   - Empty Input Handling:
     - No update to the total income, expense and balance, if non-positive income or expense is added.
3. Income and Expense Functionality:
   - Add Income: When the user enters a positive number in the income input box and clicks the "Add" button:
     - The value is added to the total income.
     - The balance is updated as total income - total expenses.
     - The income input box is cleared.
   - Add Expense: When the user enters a positive number in the expense input box and clicks the "Add" button:
     - The value is added to the total expenses.
     - The balance is updated as total income - total expenses.
     - The expense input box is cleared.
4. Reset Functionality:
   - Add a "Reset" button which resets the Total Income, Total Expenses, and Balance to 0.
   - Clicking the "Reset" button ensures the application returns to its initial state.

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

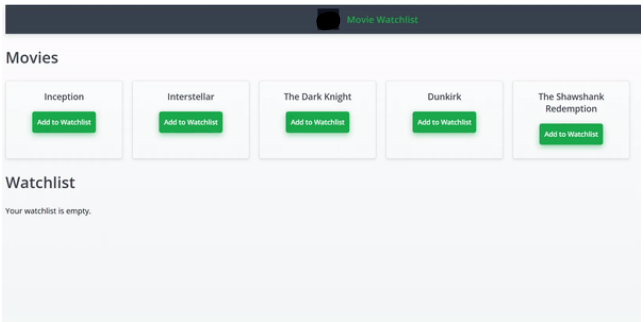
Table of data-testid

data-testid	Description
income-input	Input box to enter the income amount.
add-income-btn	Button to add the entered income.
expense-input	Input box to enter the expense amount.
add-expense-btn	Button to add the entered expense.
total-income	Text displaying the total income value.
total-expenses	Text displaying the total expenses value.
balance	Text displaying the current balance (income-expense).
reset-btn	Button to reset the income, expense, and balance.

Question - 5
React: Movie Watchlist

In this application, the task is to manage a user's movie watchlist. Edit the *Watchlist.js*, *WatchlistApp.js*, and *MovieCard.j*s components within the project structure to ensure the application functions as described below:

Hide animation



Functionality Requirements:

1. Initial State of the project:
   - All movies should be displayed as cards in the "Movies" section, each with an "Add to Watchlist" button. The MovieCard component is used to generate a card for movie.
   - The watchlist section should be empty initially, displaying a message: "Your watchlist is empty."
   - The "Add to Watchlist" button should be enabled for all movies by default.
2. Watchlist Addition Functionality:
   - When the "Add to Watchlist" button is clicked on a movie, the movie should be added to the watchlist.
   - The "Add to Watchlist" button should be disabled for movies already in the watchlist, and it's text should be changed to "Added".
3. Watchlist Display Functionality:
   - The watchlist should display movies as cards, similar to the main movie section, with each card containing a "Remove" button, using the same MovieCard component.
   - Clicking the "Remove" button should remove the movie from the watchlist, and the corresponding "Add to Watchlist" button should be re-enabled.
4. Clear Watchlist Functionality:
   - A "Clear Watchlist" button should be displayed in the watchlist section if there are movies in the watchlist.
   - Clicking the "Clear Watchlist" button should remove all movies from the watchlist and display the message: "Your watchlist is empty."
5. Movie Count Display:
   - The total number of movies in the watchlist should be displayed in a badge/icon near the watchlist title. For example: "Watchlist (2)".

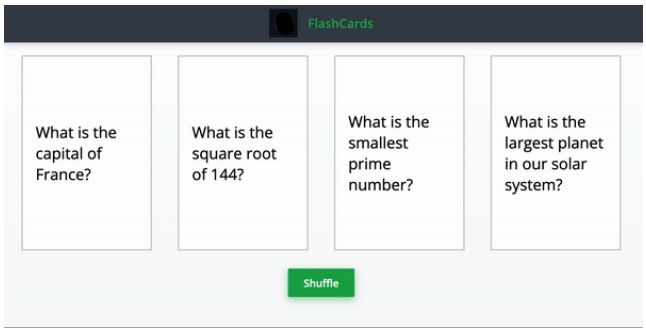
Note:
The following data-testid attributes are required in the components for the test cases to pass, do not change/delete them:

Table of data-testid

data-testid	Description
movie-card-{id}	Movie card displaying the movie alongwith Remove button.
movie-title-{id}	The title of the movie displayed in either the movie list or watchlist.
add-btn-{id}	Button to add the movie with the specified ID to the watchlist.
remove-btn-{id}	Button to remove the movie with the specified ID from the watchlist.
watchlist-container	Displays all the movies in the watchlist alongwith its count.
watchlist-empty	Message indicating that the watchlist is empty.
movie-container	Displays all the movies in the watchlist.
clear-watchlist-btn	Button to clear all movies from the watchlist.

Question - 1
React (Typescript): Flashcards

Hide animation



Using Typescript and React, design a flashcard app that displays a series of flashcards with questions on the front and answers on the back. Certain core React functionalities are already implemented.

The application has two components: *FlashCardDeck.tsx* and *FlashCard.tsx* where the functionalities should be implemented.

The component must have the following functionalities:
- Display a series of flashcards with questions on the front and answers on the back.
- Clicking a flashcard should flip it to reveal the other side.
- Update the *isFlipped* constant to a state variable in *FlashCard/index.tsx*:
  - When *isFlipped* is true
    - *flipped* is appended to divs with the class name *flashcard-content*.
    - the answer is shown.
  - When *isFlipped* is false
    - '' is appended to divs with the class name *flashcard-content*.
    - the question is shown.
- Clicking the *Shuffle* button should reorder the flashcards.
  - Ensure the shuffled order of the flashcards is different from the original order present in *src/data/cards-data.ts*.
- All the types are defined under file *src/types/FlashCard.ts*

Create a type for *FlashCard* in *src/types/FlashCard.ts* with the following properties:
- *id,* a *number*
- *question*, a *string*
- *answer*, a *string*

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

Attribute	Component
flashcard-deck	the main FlashCardApp component
flashcard-container-{card.id}	an individual FlashCard
flashcard-question-{card.id}	the question text in a FlashCard
flashcard-answer-{card.id}	the answer text in FlashCard

shuffle-button	a button to shuffle the flashcards

Note:
- Components have *data-testid* attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The files that should be modified by the candidate are *src/components/FlashCardDeck.tsx*, *src/components/FlashCard.tsx*, and *src/types/FlashCard.ts*.

Question - 2
React: Image Preview

Complete a React image preview application as shown below to pass all the unit tests. Certain core React functionalities are already implemented.

Show animation

The application has two components:
- The ImagePreview component that allows users to view and hide images by clicking on them.
- The HiddenImageDiv component that should be rendered when an image is to be hidden,

The image data is passed to the component as "images" to render in the component.

The application has the following functionalities:
- The ImagePreview component renders the following conditionally :
  - If the *visible* attribute is true, then the image with the *src* and *alt* attributes passed in the image data.
  - If the *visible* attribute is false, then the *HiddenImageDiv* component
- Clicking an image should hide it from the DOM, and the *HiddenImageDiv* should be visible in its place.
- Clicking the *HiddenImageDiv* component should render the original image again.
- On clicking the *ShowAll* button, all the images should be visible, and any *HiddenImageDiv* component should be hidden.
- On clicking the *HideAll* button, all the images should be hidden, and only the *HiddenImageDiv* components should be visible.

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

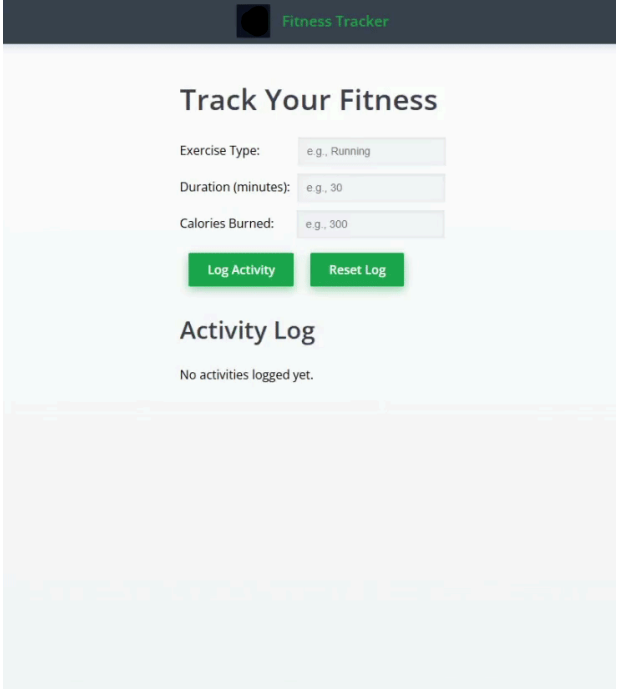
Attribute	Component
images-div	Images display div
show-all-btn	Show all images button
hide-all-btn	Hide all images button

Note:
- Components have *data-testid* attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The only file that should be modified by the candidate is the src/components/ImagePreview.js.

Question - 3
React: FitTrack Pro

In this application, the task is to create a fitness tracker that allows users to log their daily activities, including exercise type, duration, and calories burned. Edit the *FitnessTracker.js*, *LogForm.js,* and *LogList.js* within the project structure to ensure the application functions as described below:

Hide animation



Functionality Requirements:

1. Initial State of the project:
   - The exercise-type, duration, and calories-burned input fields should be empty initially.
   - The exercise-type input field should be of type text.
   - The duration and calories-burned input fields should be of type number.
   - The *Log Activity* button should not add any log until all fields have valid inputs.
2. Logging functionality:
   - The *Log Activity* button should add a log only when all fields have valid values.
   - The duration and calories burned should be non-zero and positive numbers.
   - On clicking the *Log Activity* button:
     - Add a new entry in the log table, displaying the *exercise-type*, *duration* (in minutes), and *calories-burned*.
     - Clear the input fields after successfully adding the entry.
   - On clicking the *Reset Log* button:
     - All logs are cleared and reset to an empty list.
     - The error message should also be cleared.
3. Log Display:
   - All log entries should appear within the application, under the *Activity Log* heading.
   - Each log entry, displays:
     - Exercise Type: the type of exercise logged
     - Duration: the time spent in minutes
     - Calories Burned: the calories burned during the activity
4. Error Handling:
   - Display the error message, if the user tries to submit invalid data, such as empty fields or non-positive numbers.
     - "Exercise type must not be empty." : If the exercise type input is empty.
     - "Duration must be a positive number.": If the duration input is empty or non-positive.
     - "Calories must be a positive number.": If the calories burned input is empty or non-positive.

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
input-exerciseType	Input box to enter the type of exercise
input-duration	Input box to enter the duration of the activity (in minutes)
input-caloriesBurned	Input box to enter the calories burned

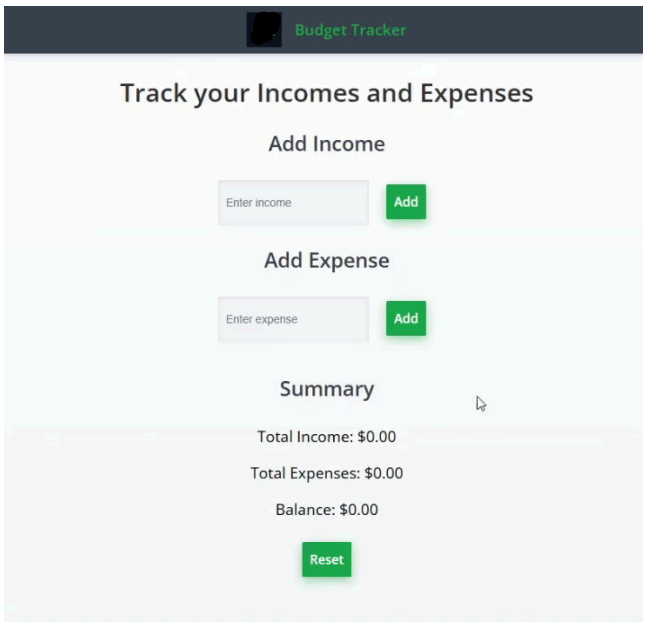
btn-logActivity	Button to log the activity
btn-resetLog	Button to reset all the activities
log-list	List of all the logs
log-entry	To display the user's logged activities
error-message	Error message for displaying the invalid inputs

Question - 4

React: Budget Balance Assistant

In this application, the task is to build a budget tracker that helps users manage their income and expenses effectively. Edit the *BudgetTracker.js*, *IncomeForm.js*, *ExpenseForm.js* and *Summary.js* components within the project structure to ensure the application functions as described below:

Hide animation



Functionality Requirements:

1. Initial State of the project:
   - The income and expense input fields should be empty and must have the attribute type as number.
   - Ensure that the total income, total expenses, and balance are initialized to 0.
2. Validation:
   - Positive Numbers Only:
     - The income and expense values must be positive. Negative or zero values should not be added.
   - Empty Input Handling:
     - No update to the total income, expense and balance, if non-positive income or expense is added.
3. Income and Expense Functionality:
   - Add Income: When the user enters a positive number in the income input box and clicks the "Add" button:
     - The value is added to the total income.
     - The balance is updated as total income - total expenses.
     - The income input box is cleared.
   - Add Expense: When the user enters a positive number in the expense input box and clicks the "Add" button:
     - The value is added to the total expenses.
     - The balance is updated as total income - total expenses.
     - The expense input box is cleared.
4. Reset Functionality:
   - Add a "Reset" button which resets the Total Income, Total Expenses, and Balance to 0.
   - Clicking the "Reset" button ensures the application returns to its initial state.

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

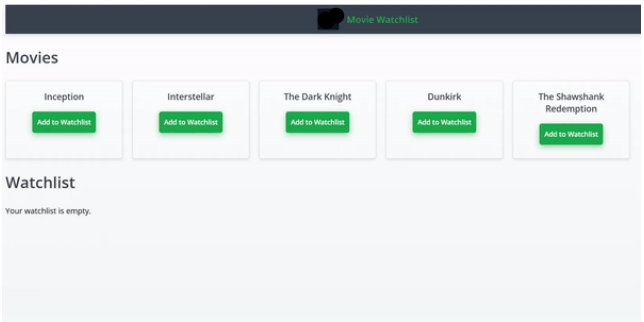
Table of data-testid

data-testid	Description
income-input	Input box to enter the income amount.
add-income-btn	Button to add the entered income.
expense-input	Input box to enter the expense amount.
add-expense-btn	Button to add the entered expense.
total-income	Text displaying the total income value.
total-expenses	Text displaying the total expenses value.
balance	Text displaying the current balance (income-expense).
reset-btn	Button to reset the income, expense, and balance.

Question - 5
React: Movie Watchlist

In this application, the task is to manage a user's movie watchlist. Edit the *Watchlist.js*, *WatchlistApp.js*, and *MovieCard.j*s components within the project structure to ensure the application functions as described below:

Hide animation



Functionality Requirements:

1. Initial State of the project:
   - All movies should be displayed as cards in the "Movies" section, each with an "Add to Watchlist" button. The MovieCard component is used to generate a card for movie.
   - The watchlist section should be empty initially, displaying a message: "Your watchlist is empty."
   - The "Add to Watchlist" button should be enabled for all movies by default.
2. Watchlist Addition Functionality:
   - When the "Add to Watchlist" button is clicked on a movie, the movie should be added to the watchlist.
   - The "Add to Watchlist" button should be disabled for movies already in the watchlist, and it's text should be changed to "Added".
3. Watchlist Display Functionality:
   - The watchlist should display movies as cards, similar to the main movie section, with each card containing a "Remove" button, using the same MovieCard component.
   - Clicking the "Remove" button should remove the movie from the watchlist, and the corresponding "Add to Watchlist" button should be re-enabled.
4. Clear Watchlist Functionality:
   - A "Clear Watchlist" button should be displayed in the watchlist section if there are movies in the watchlist.
   - Clicking the "Clear Watchlist" button should remove all movies from the watchlist and display the message: "Your watchlist is empty."
5. Movie Count Display:
   - The total number of movies in the watchlist should be displayed in a badge/icon near the watchlist title. For example: "Watchlist (2)".

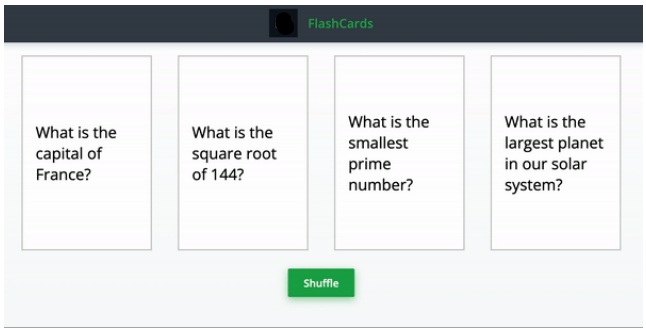
Note:
The following data-testid attributes are required in the components for the test cases to pass, do not change/delete them:

Table of data-testid

data-testid	Description
movie-card-{id}	Movie card displaying the movie alongwith Remove button.
movie-title-{id}	The title of the movie displayed in either the movie list or watchlist.
add-btn-{id}	Button to add the movie with the specified ID to the watchlist.
remove-btn-{id}	Button to remove the movie with the specified ID from the watchlist.
watchlist-container	Displays all the movies in the watchlist alongwith its count.
watchlist-empty	Message indicating that the watchlist is empty.
movie-container	Displays all the movies in the watchlist.
clear-watchlist-btn	Button to clear all movies from the watchlist.

Question - 1
React (Typescript): Flashcards

Hide animation



Using Typescript and React, design a flashcard app that displays a series of flashcards with questions on the front and answers on the back. Certain core React functionalities are already implemented.

The application has two components: *FlashCardDeck.tsx* and *FlashCard.tsx* where the functionalities should be implemented.

The component must have the following functionalities:
- Display a series of flashcards with questions on the front and answers on the back.
- Clicking a flashcard should flip it to reveal the other side.
- Update the *isFlipped* constant to a state variable in *FlashCard/index.tsx*:
  - When *isFlipped* is true
    - *flipped* is appended to divs with the class name *flashcard-content*.
    - the answer is shown.
  - When *isFlipped* is false
    - '' is appended to divs with the class name *flashcard-content*.
    - the question is shown.
- Clicking the *Shuffle* button should reorder the flashcards.
  - Ensure the shuffled order of the flashcards is different from the original order present in *src/data/cards-data.ts*.
- All the types are defined under file *src/types/FlashCard.ts*

Create a type for *FlashCard* in *src/types/FlashCard.ts* with the following properties:
- *id,* a *number*
- *question*, a *string*
- *answer*, a *string*

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

Attribute	Component
flashcard-deck	the main FlashCardApp component
flashcard-container-{card.id}	an individual FlashCard
flashcard-question-{card.id}	the question text in a FlashCard
flashcard-answer-{card.id}	the answer text in FlashCard

shuffle-button	a button to shuffle the flashcards

Note:
- Components have *data-testid* attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The files that should be modified by the candidate are *src/components/FlashCardDeck.tsx*, *src/components/FlashCard.tsx*, and *src/types/FlashCard.ts*.

Question - 2
React: Image Preview

Complete a React image preview application as shown below to pass all the unit tests. Certain core React functionalities are already implemented.

Show animation

The application has two components:
- The ImagePreview component that allows users to view and hide images by clicking on them.
- The HiddenImageDiv component that should be rendered when an image is to be hidden,

The image data is passed to the component as "images" to render in the component.

The application has the following functionalities:
- The ImagePreview component renders the following conditionally :
  - If the *visible* attribute is true, then the image with the *src* and *alt* attributes passed in the image data.
  - If the *visible* attribute is false, then the *HiddenImageDiv* component
- Clicking an image should hide it from the DOM, and the *HiddenImageDiv* should be visible in its place.
- Clicking the *HiddenImageDiv* component should render the original image again.
- On clicking the *ShowAll* button, all the images should be visible, and any *HiddenImageDiv* component should be hidden.
- On clicking the *HideAll* button, all the images should be hidden, and only the *HiddenImageDiv* components should be visible.

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

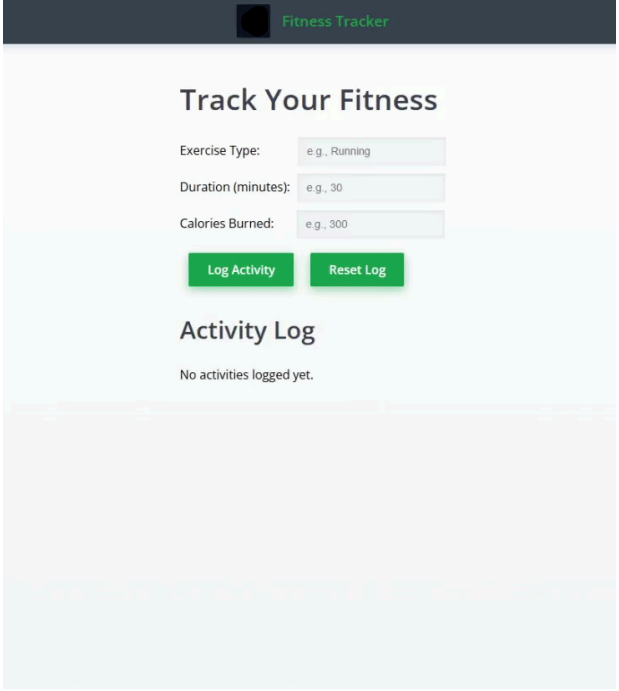
Attribute	Component
images-div	Images display div
show-all-btn	Show all images button
hide-all-btn	Hide all images button

Note:
- Components have *data-testid* attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The only file that should be modified by the candidate is the src/components/ImagePreview.js.

Question - 3
React: FitTrack Pro

In this application, the task is to create a fitness tracker that allows users to log their daily activities, including exercise type, duration, and calories burned. Edit the *FitnessTracker.js*, *LogForm.js,* and *LogList.js* within the project structure to ensure the application functions as described below:

Hide animation



Functionality Requirements:

1. Initial State of the project:
   - The exercise-type, duration, and calories-burned input fields should be empty initially.
   - The exercise-type input field should be of type text.
   - The duration and calories-burned input fields should be of type number.
   - The *Log Activity* button should not add any log until all fields have valid inputs.
2. Logging functionality:
   - The *Log Activity* button should add a log only when all fields have valid values.
   - The duration and calories burned should be non-zero and positive numbers.
   - On clicking the *Log Activity* button:
     - Add a new entry in the log table, displaying the *exercise-type*, *duration* (in minutes), and *calories-burned*.
     - Clear the input fields after successfully adding the entry.
   - On clicking the *Reset Log* button:
     - All logs are cleared and reset to an empty list.
     - The error message should also be cleared.
3. Log Display:
   - All log entries should appear within the application, under the *Activity Log* heading.
   - Each log entry, displays:
     - Exercise Type: the type of exercise logged
     - Duration: the time spent in minutes
     - Calories Burned: the calories burned during the activity
4. Error Handling:
   - Display the error message, if the user tries to submit invalid data, such as empty fields or non-positive numbers.
     - "Exercise type must not be empty." : If the exercise type input is empty.
     - "Duration must be a positive number.": If the duration input is empty or non-positive.
     - "Calories must be a positive number.": If the calories burned input is empty or non-positive.

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
input-exerciseType	Input box to enter the type of exercise
input-duration	Input box to enter the duration of the activity (in minutes)
input-caloriesBurned	Input box to enter the calories burned

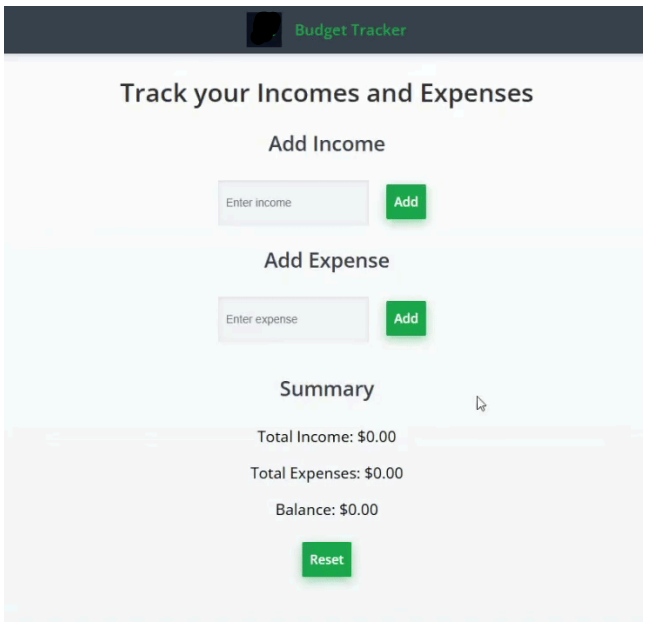
btn-logActivity	Button to log the activity
btn-resetLog	Button to reset all the activities
log-list	List of all the logs
log-entry	To display the user's logged activities
error-message	Error message for displaying the invalid inputs

Question - 4

React: Budget Balance Assistant

In this application, the task is to build a budget tracker that helps users manage their income and expenses effectively. Edit the *BudgetTracker.js*, *IncomeForm.js*, *ExpenseForm.js* and *Summary.js* components within the project structure to ensure the application functions as described below:

Hide animation



Functionality Requirements:

1. Initial State of the project:
   - The income and expense input fields should be empty and must have the attribute type as number.
   - Ensure that the total income, total expenses, and balance are initialized to 0.
2. Validation:
   - Positive Numbers Only:
     - The income and expense values must be positive. Negative or zero values should not be added.
   - Empty Input Handling:
     - No update to the total income, expense and balance, if non-positive income or expense is added.
3. Income and Expense Functionality:
   - Add Income: When the user enters a positive number in the income input box and clicks the "Add" button:
     - The value is added to the total income.
     - The balance is updated as total income - total expenses.
     - The income input box is cleared.
   - Add Expense: When the user enters a positive number in the expense input box and clicks the "Add" button:
     - The value is added to the total expenses.
     - The balance is updated as total income - total expenses.
     - The expense input box is cleared.
4. Reset Functionality:
   - Add a "Reset" button which resets the Total Income, Total Expenses, and Balance to 0.
   - Clicking the "Reset" button ensures the application returns to its initial state.

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

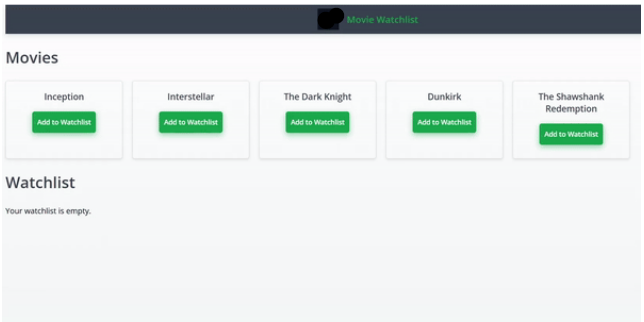
Table of data-testid

data-testid	Description
income-input	Input box to enter the income amount.
add-income-btn	Button to add the entered income.
expense-input	Input box to enter the expense amount.
add-expense-btn	Button to add the entered expense.
total-income	Text displaying the total income value.
total-expenses	Text displaying the total expenses value.
balance	Text displaying the current balance (income-expense).
reset-btn	Button to reset the income, expense, and balance.

Question - 5
React: Movie Watchlist

In this application, the task is to manage a user's movie watchlist. Edit the *Watchlist.js*, *WatchlistApp.js*, and *MovieCard.j*s components within the project structure to ensure the application functions as described below:

Hide animation



Functionality Requirements:

1. Initial State of the project:
   - All movies should be displayed as cards in the "Movies" section, each with an "Add to Watchlist" button. The MovieCard component is used to generate a card for movie.
   - The watchlist section should be empty initially, displaying a message: "Your watchlist is empty."
   - The "Add to Watchlist" button should be enabled for all movies by default.
2. Watchlist Addition Functionality:
   - When the "Add to Watchlist" button is clicked on a movie, the movie should be added to the watchlist.
   - The "Add to Watchlist" button should be disabled for movies already in the watchlist, and it's text should be changed to "Added".
3. Watchlist Display Functionality:
   - The watchlist should display movies as cards, similar to the main movie section, with each card containing a "Remove" button, using the same MovieCard component.
   - Clicking the "Remove" button should remove the movie from the watchlist, and the corresponding "Add to Watchlist" button should be re-enabled.
4. Clear Watchlist Functionality:
   - A "Clear Watchlist" button should be displayed in the watchlist section if there are movies in the watchlist.
   - Clicking the "Clear Watchlist" button should remove all movies from the watchlist and display the message: "Your watchlist is empty."
5. Movie Count Display:
   - The total number of movies in the watchlist should be displayed in a badge/icon near the watchlist title. For example: "Watchlist (2)".

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

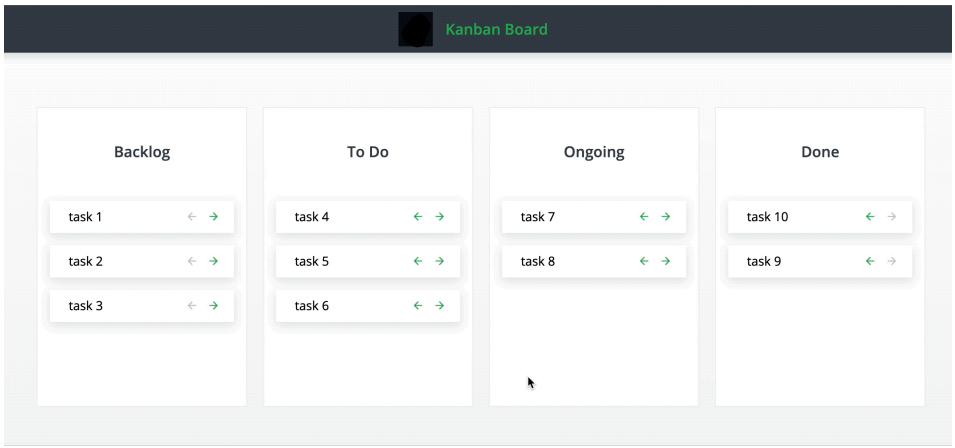
data-testid	Description
movie-card-{id}	Movie card displaying the movie alongwith Remove button.
movie-title-{id}	The title of the movie displayed in either the movie list or watchlist.
add-btn-{id}	Button to add the movie with the specified ID to the watchlist.
remove-btn-{id}	Button to remove the movie with the specified ID from the watchlist.
watchlist-container	Displays all the movies in the watchlist alongwith its count.
watchlist-empty	Message indicating that the watchlist is empty.
movie-container	Displays all the movies in the watchlist.
clear-watchlist-btn	Button to clear all movies from the watchlist.

Question - 1
React: 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:

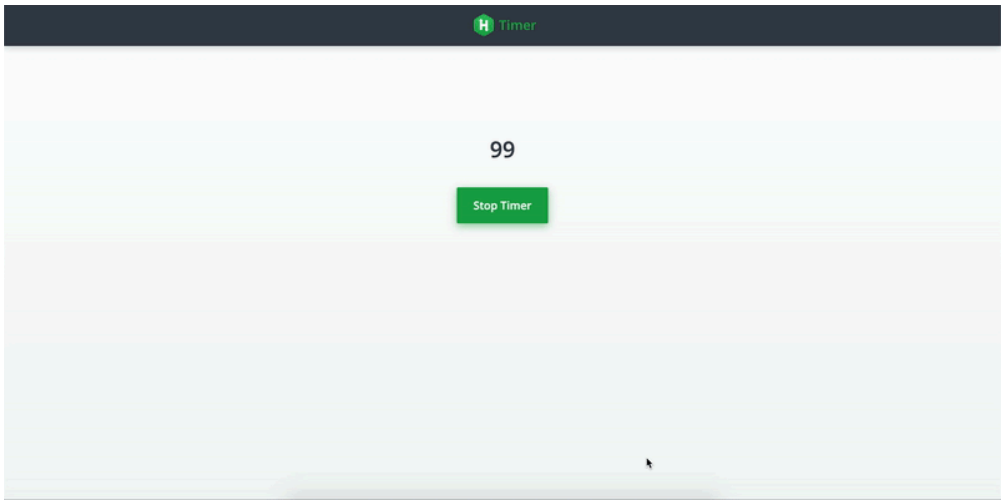
- The <ul> for the 'Backlog' stage should have the data-testid attribute 'stage-0'.
- The <ul> for the 'To Do' stage should have the data-testid attribute 'stage-1'.
- The <ul> for the 'Ongoing' stage should have the data-testid attribute 'stage-2'.
- The <ul> for the 'Done' stage should have the data-testid 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 - 2
React: 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:

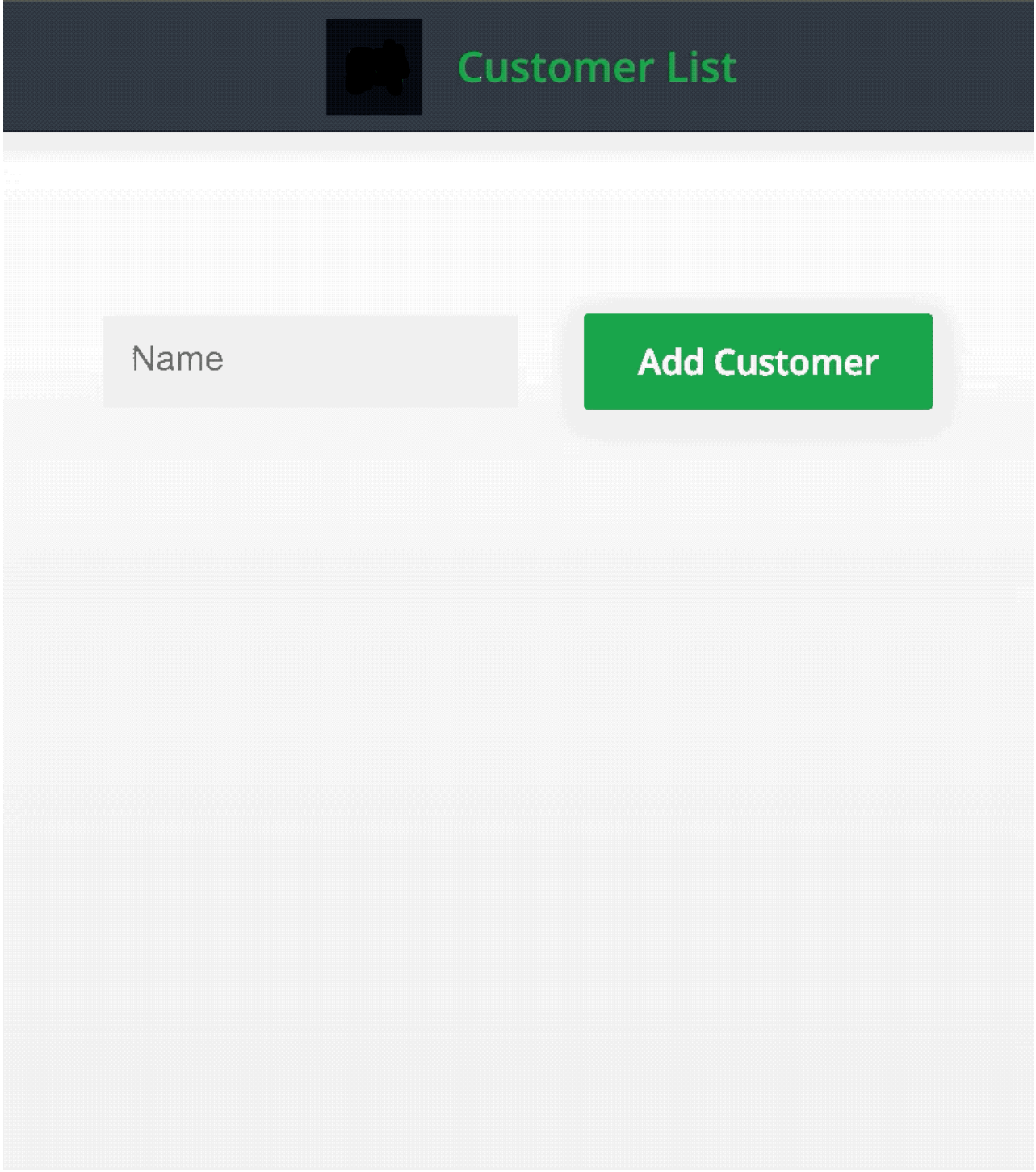
Component	Attribute
Title	*app-title*
Timer value	*timer-value*
Stop timer button	*stop-button*

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 - 3
React: Customer List

Create a customer list component as shown.

Hide animation



The component must have the following functionalities:

- The input should initially be empty.
- If no value is entered, clicking on the 'Add Customer' button should not do anything.
- Clicking on the 'Add Customer' button should add the input value to the list below. For this, add *
{input}
* to the <ul data-testid="customer-list"> element.
- After the value is added to the list, it should clear the value in the input box.
- Please note that the customer list <ul> element should only be rendered if it has at least one customer added, i.e. at least one <li> child. When the app is mounted, since no customers are added, the <ul> element should not be rendered.
- All the values added by the button should be rendered in the list below.

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

Component	Attribute
Input	*app-input*

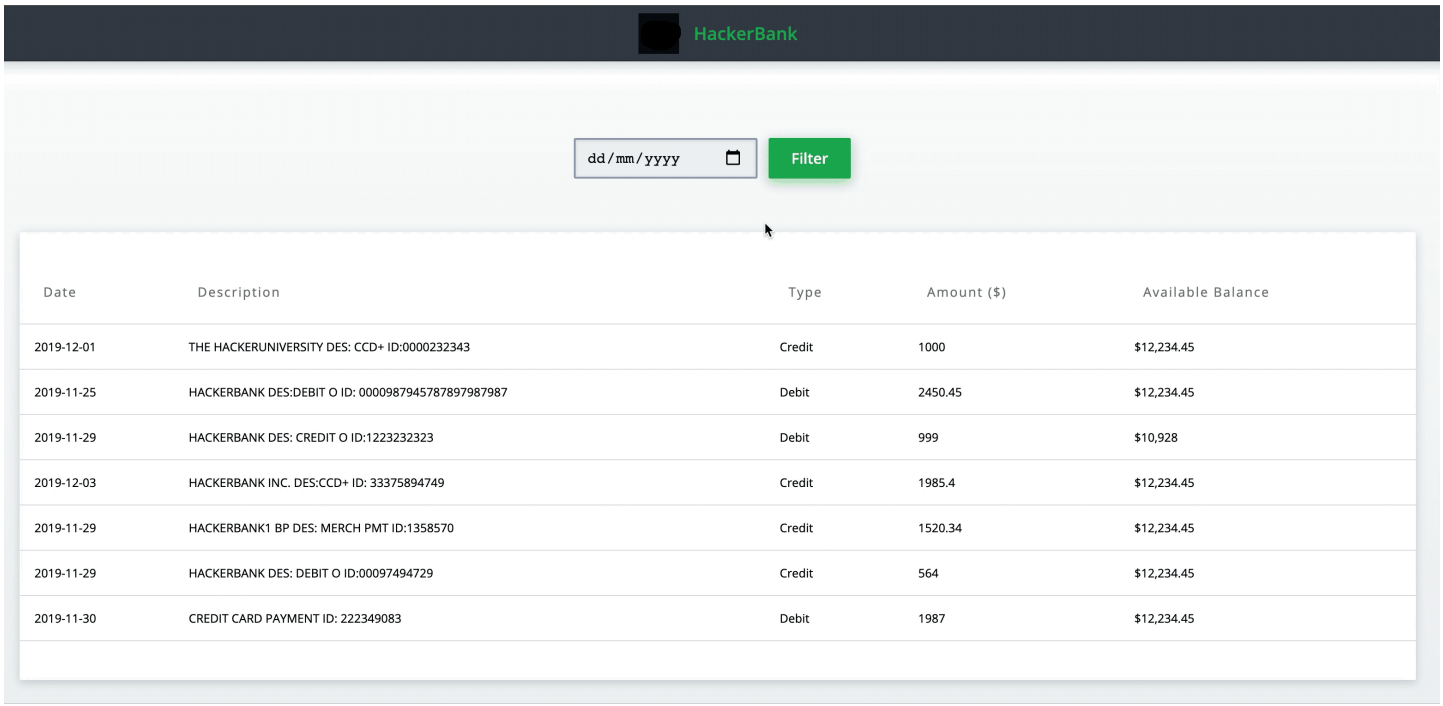
Button	*submit-button*
Customer list	*customer-list*
List elements	*list-item0, list-item1, ...*

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 - 4
React: HackerBank

Create a banking 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



Implement the following functionalities:

- All transactions are initially displayed inside the table in the order they are retrieved from the source. The source is passed down as '*txns*' prop to TransactionTable component in App.js.
- Picking the date from the date input and pressing the 'Filter' button should display all the records for that date in the table. If no date is chosen, the 'Filter' button should not do anything.
- Clicking on the 'Amount ($)' table header should sort the records in ascending order of amount. The behavior is the same for multiple clicks on 'Amount ($)'.

Each transaction object contains the following properties :

- *String* date: The date when the transaction took place in the format YYYY-MM-DD.
- *String* description: The description of the transaction.
- *Number* type: The type of transaction, where 0 denotes a credit transaction and 1 denotes a debit transaction.
- *Float* amount: The total amount of the transaction.
- *String* balance: The balance of the account after the transaction was completed, prefixed with a dollar sign ($).

```
{
    "date": "2019-12-03",
    "description": "HACKERBANK INC. DES:CCD+ ID: 33375894749",
    "type": 0,
```

```
    "amount": 1985.4,
    "balance": "$12,234.45"
}
```

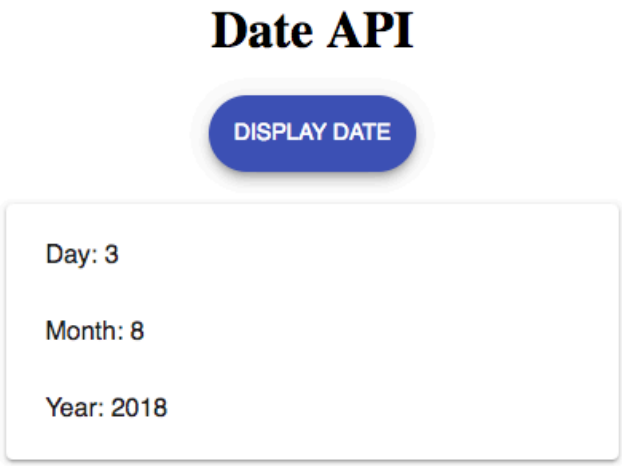
Question - 5
React: Date API

In the given single page React project, a button click prompts an HTTP GET request to an API endpoint that returns a *date* as its API response. Complete the project so it displays the current day, month, and year after clicking the button. Model the implementation after the instructions below.
Certain core React functionalities have already been implemented. Complete the React application in order to pass all the unit tests.

Demo

- Link to an animated GIF
- Scroll sideways below to explore the screenshots. Clicking each screenshot opens it in a new window.

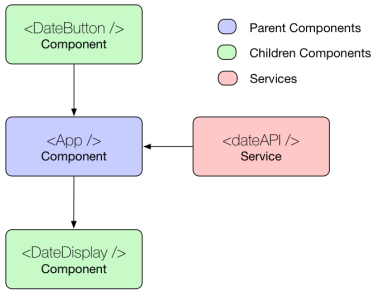
Application's initial state before clicking the button.



Display date after clicking the button.

Specifications

- Component flow diagram of the application is as follows:



▼ Framework Specific Instructions

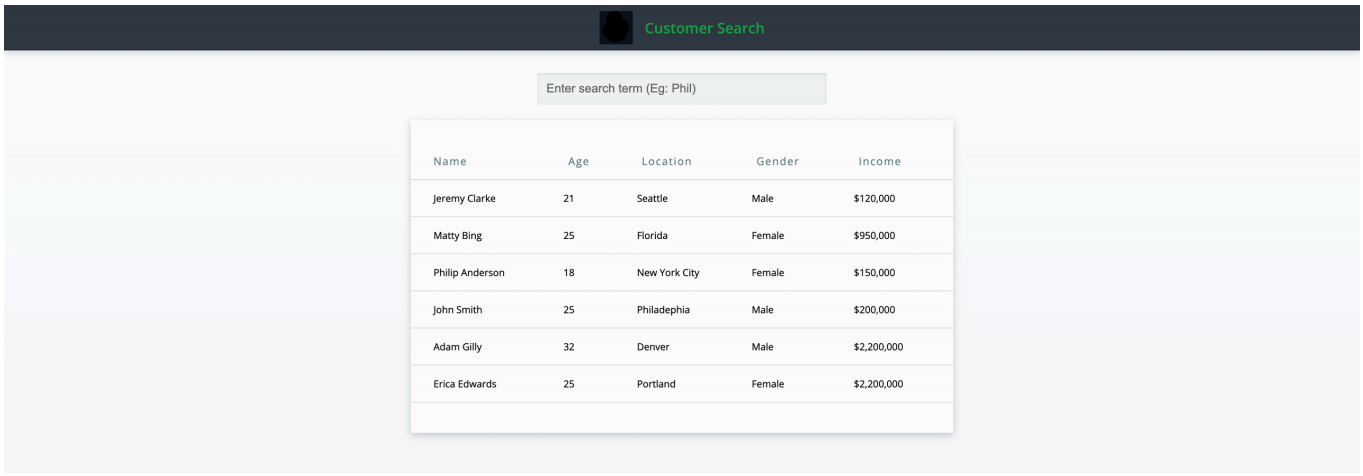
- The project uses React 16 by default. Changing the React version may interfere with tests and is strictly discouraged. You may refer to the React 16 docs here.
- The project uses create-react-app and react-scripts to automate serving and testing of the application.
- The project uses enzyme as a testing framework. Please refrain from changing the test framework or the tests themselves.
- The project uses React Material UI as a design framework. API docs are available here and can be used as a reference.

Question - 1

React(TypeScript): Customer Search

Complete the component as shown to pass all the test cases. Certain core React functionalities are already implemented.

Hide animation Show animation



The list of available customers and their details is imported as `List` in the SearchCustomer component. Use the list to render them as shown.

The data in List file is in this form.

```
{
   name: "Jeremy Clarke",
   age: 21,
   location: "Seattle",
   gender: "Male",
   income: "$120,000"
}
```

The application has 2 components:

- `SearchCustomer` - contains and search box and CustomerList component.
- `CustomerList` - displays the details of the customer based on the search term in a table.

The component must have the following functionalities:

- Initially, the input field must be empty. Whenever the input is empty, all the customers passed in the input to the component must be rendered in the list.
- The type of input in the input box should be `text`.
- As soon as the search term is typed in the input, search for customer records with any field that starts with the search term. For example, if the search term is "Phil", then records with the name "Philip Anderson" and the location "Philadelphia" should be displayed in the results.
- The filtered list should preserve the order customers are given in the input to the component.
- If the search term returns no customers, do not render the table. Instead, display the message "No Results Found!".
- The search results should be case-sensitive.

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

Attribute	Component

search-input	Search Input Box
searched-customers	List of searched customers
no-results	div when no customer matches the search term

Note:

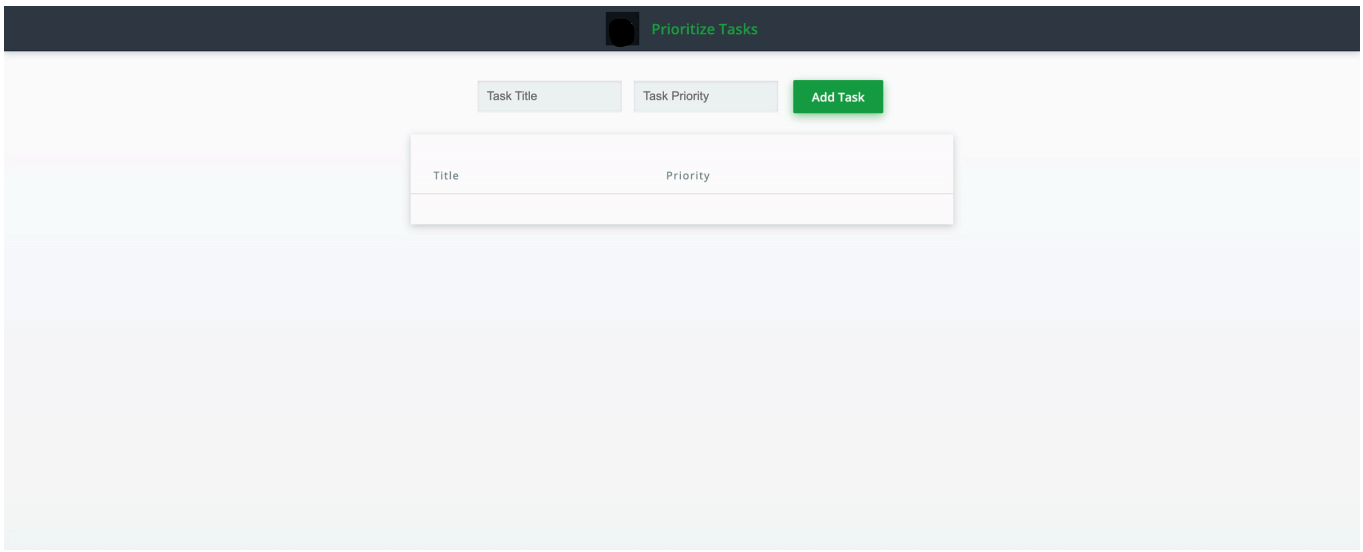
- Components have data-testid attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The files that should be modified by the candidate are `src/components/CustomerList.tsx` and `src/components/SearchCustomer.tsx`. They are open by default in the system editor.
- Avoid making changes to other files in the project structure.

Question - 2

React(TypeScript): Prioritize Tasks

Complete the component as shown to pass the test cases. Certain core React functionalities are already implemented.

Hide animation Show animation



The application has only one component, `Tasks`, where all the functionalities will be implemented.

The component must have the following functionalities:

- The type of input for the:
  - Task Name input box should be "text".
  - Task Priority input box should be "number".
- The initial view should have no tasks in the table.
- Clicking the `Add` button should:
  - add a task and its priority in the table and sort the tasks according to priority.
  - display an alert saying "Please enter both title and task priority" if any input boxes are empty.
  - reset the input boxes to empty after adding a valid task to the table.
- Clicking the `Delete` button should delete the corresponding task from the table.

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

Attribute	Component
input-task-name	Input box for task name
input-task-priority	Input box for task priority

submit-button	Button for adding task
tasksList	List of tasks

Note:

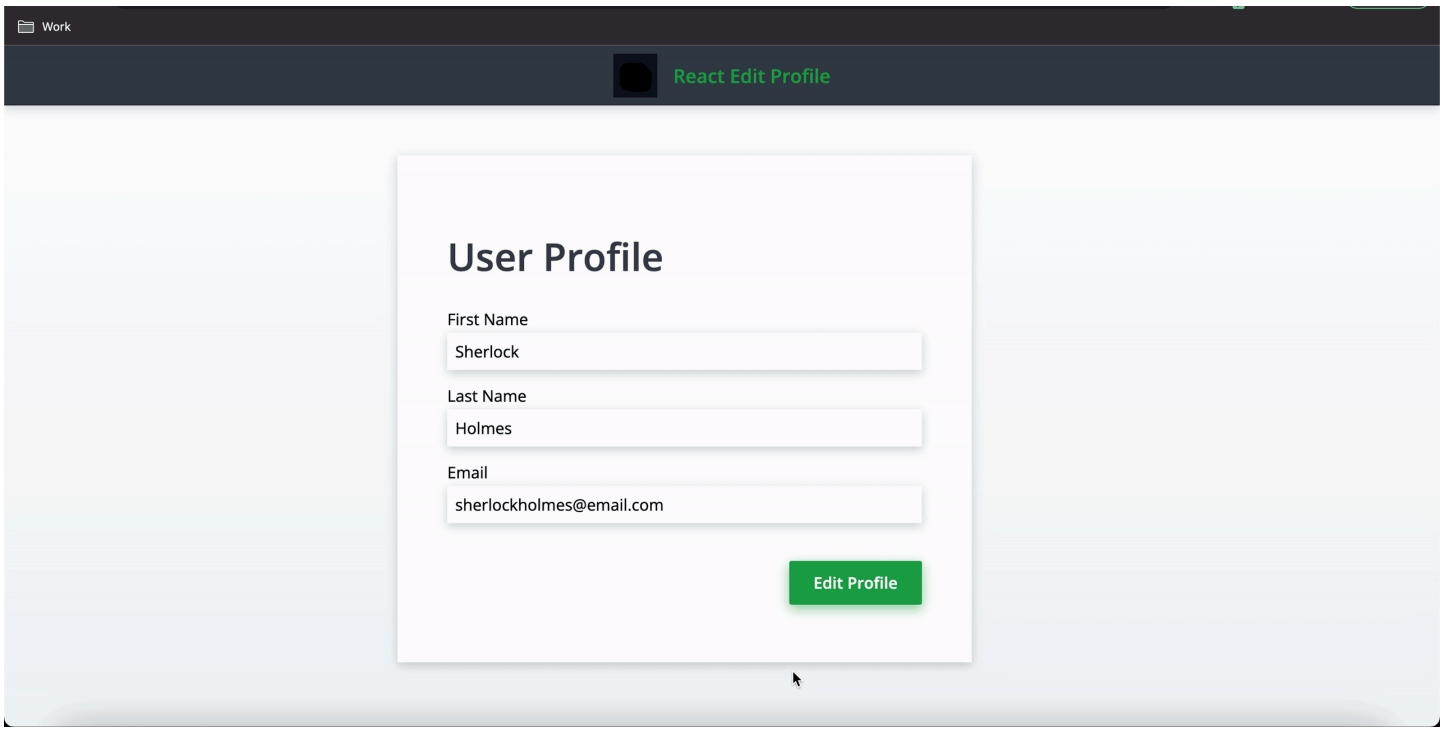
- Components have data-testid attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The file that should be modified by the candidate `src/components/Books/index.tsx` is open by default in the system editor.
- Avoid making changes to other files in the project structure.

Question - 3

React(TypeScript): Edit Profile

Complete a React Edit Profile application as shown in order to pass all the unit tests. Certain core React functionalities are already implemented.

Hide animation Show animation



The application has only 1 component:

- The Profile component enables the user to update his/her details.

The application has the following functionalities:

- The Profile component shows the user their details and allows editing.
- Initially, there should be 3 divs with the following values:
  - First Name: **Sherlock**
  - Last Name: **Holmes**
  - Email: **sherlockholmes@email.com**
- Initially, the button should have the text value Edit Profile.
- The following functionality should be implemented when the user clicks the 'Edit Profile' button.
  - The user can now edit the details in the divs.
  - The button should now have the text value "Save Changes".
- The following functionality should be implemented when the user edits the details and clicks the Save Changes button.
  - If one or more of the fields is empty, there should be an alert prompt with the text "Please enter all profile fields" and no values should be updated.
  - If all the fields are non-empty, there should be an alert prompt with the text "Profile updated successfully".
  - The input values should be the updated values and should be in uneditable divs.
  - The button text should be Edit Profile.

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

Description	Name
First Name	firstNameDiv
Last Name	lastNameDiv
Email	emailDiv
First Name <input>	firstNameInput
Last Name <input>	lastNameInput
Email <input>	emailInput
<button>.	changeButton

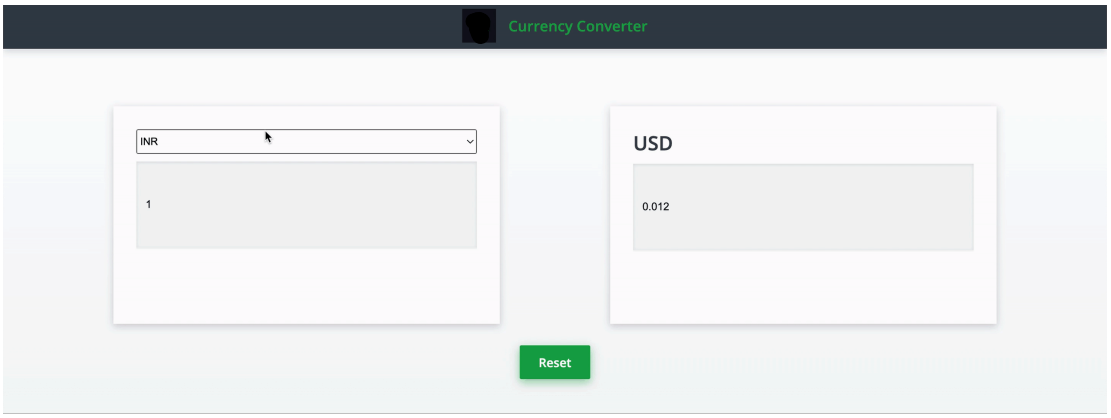
The component has data-testid attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.

Question - 4

React(TypeScript): Currency Converter

Complete the currency converter component that converts the value in any of the given currencies to USD as shown.

Hide animation Show animation



The list of available currencies is imported as "exchangeRates" in the component that can be used to render the list of currencies. The initial values for the currency and the input box are also imported from "exchangeRates.ts".

It is given that:

- The first input box is to be used to take input.
- The second input box is read-only to display the converted value.
- Only positive integer or decimal values can be taken as input.

The component must have the following functionalities:

- The default value for:
  - input box should be 1.
  - the currency to display in the select menu should be INR.
  - The second input box should be the conversion of 1 INR to USD i.e. 0.012.
- The select menu should have five currencies in the options.

- The conversion should happen automatically:
  - when there is any input change.
  - on selecting another currency.
- The converted value should be up to 3 decimal places.
- The placeholder text should change automatically according to the currency.
- The Reset button should reset the input to 1 and give the conversion in the current currency.

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

Component	Attribute
select menu	select-currency
first input box	input-value
second input box	converted-value
Clear Input button	clear-value

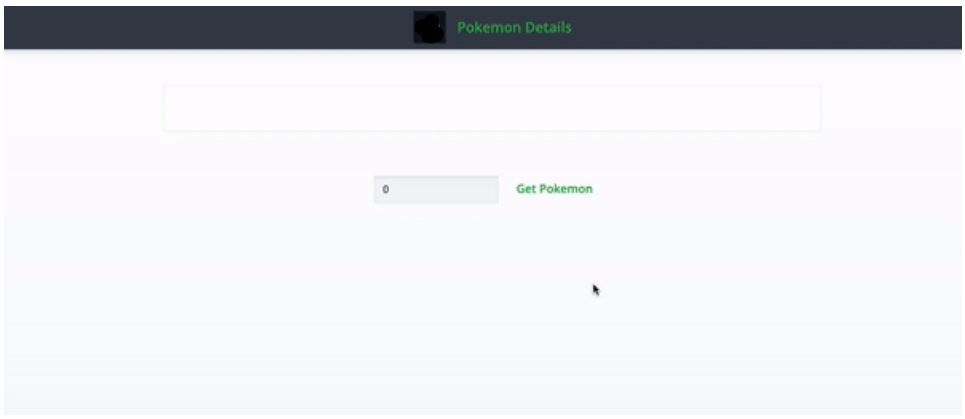
Note:
- Components have data-testid attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The only file that should be modified by the candidate is "src/components/currency-converter/index.tsx". It is open by default in the system editor.
- Avoid making changes to other files in the project structure.

Question - 5
React(TypeScript): Pokemon Details

After twenty-one years, Ash Ketchum was finally a Pokémon Champion. He decided to retire from Pokemon training and started software development. He is creating a pokemon search application, which takes a pokemon id as input and searches the pokemon database for the pokemon and its evolved and primitive versions.

Help Ash to complete the React Components as shown below.

Hide animation Show animation



The component should have the following functionalities:
- The input field should take the pokemon id as a number.
- The id should be in the range of 1 to 151. Otherwise, the component should show an alert with the text 'Pokemon ID must be between 1 and 151'.
- On getting a valid pokemon id, the component should make a GET request to the following API endpoint 'https://jsonmock.hackerrank.com/api/pokemon?id=<id>' which returns pokemon data in JSON format.
  - e.g. to get Pokemon with id = 1, the API endpoint will is https://jsonmock.hackerrank.com/api/pokemon?id=1
  - For an API call with a valid pokemon ID, the response will be:

```
{
  "data":{
    "id":1,
    "num":"001",
    "name":"Bulbasaur",
    "type":[
      "Grass",
      "Poison"
    ],
    "height":"0.71 m",
    "weight":"6.9 kg",
    "candy":"Bulbasaur Candy",
    "candy_count":25,
    "egg":"2 km",
    "spawn_chance":0.69,
    "avg_spawns":69,
    "spawn_time":"20:00",
    "multipliers":[1.58],
    "weaknesses":[
      "Fire",
      "Ice",
      "Flying",
      "Psychic"
    ],
    "next_evolution":[
      {
        "num":"002",
        "name":"Ivysaur"
      },
      {
        "num":"003",
        "name":"Venusaur"
      }
    ]
  }
}
```


- If the pokemon contains the next evolved version, the 'Next Evolution' button should be enabled.
- If the pokemon contains a previous evolved version (primitive version), the 'Prev Evolution' button should be enabled.
- On clicking the 'Next Evolution' button, the next evolved version of the pokemon should be fetched from the API and loaded.
- On clicking the 'Prev Evolution' button, the previous version of the pokemon should be fetched from the API and loaded.
- The Pokemon.tsx component should be used to render all the pokemon data by passing the pokemon as props.

Note:

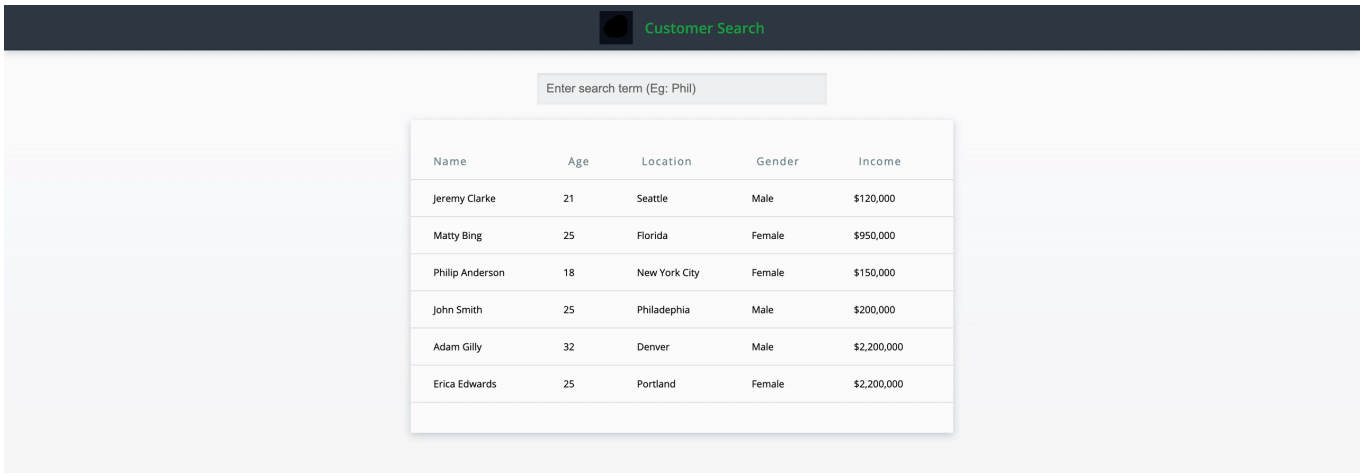
- Please note that components have the above data-testid attributes for test cases and certain classes and ids for rendering purposes. It is advised not to change them.
- The only file that should be modified by the candidate is "/src/components/pokemon-details/PokemonDetails.tsx", "src/components/pokemon/Pokemon.tsx" and is open by default by the system editor.
- It is advised for candidates to avoid making any changes to the rest of the files in the project structure.

Question - 1

React(TypeScript): Customer Search

Complete the component as shown to pass all the test cases. Certain core React functionalities are already implemented.

Hide animation Show animation



The list of available customers and their details is imported as `List` in the SearchCustomer component. Use the list to render them as shown.

The data in List file is in this form.

```
{
   name: "Jeremy Clarke",
   age: 21,
   location: "Seattle",
   gender: "Male",
   income: "$120,000"
}
```

The application has 2 components:

- `SearchCustomer` - contains and search box and CustomerList component.
- `CustomerList` - displays the details of the customer based on the search term in a table.

The component must have the following functionalities:

- Initially, the input field must be empty. Whenever the input is empty, all the customers passed in the input to the component must be rendered in the list.
- The type of input in the input box should be `text`.
- As soon as the search term is typed in the input, search for customer records with any field that starts with the search term. For example, if the search term is "Phil", then records with the name "Philip Anderson" and the location "Philadelphia" should be displayed in the results.
- The filtered list should preserve the order customers are given in the input to the component.
- If the search term returns no customers, do not render the table. Instead, display the message "No Results Found!".
- The search results should be case-sensitive.

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

Attribute	Component

search-input	Search Input Box
searched-customers	List of searched customers
no-results	div when no customer matches the search term

Note:

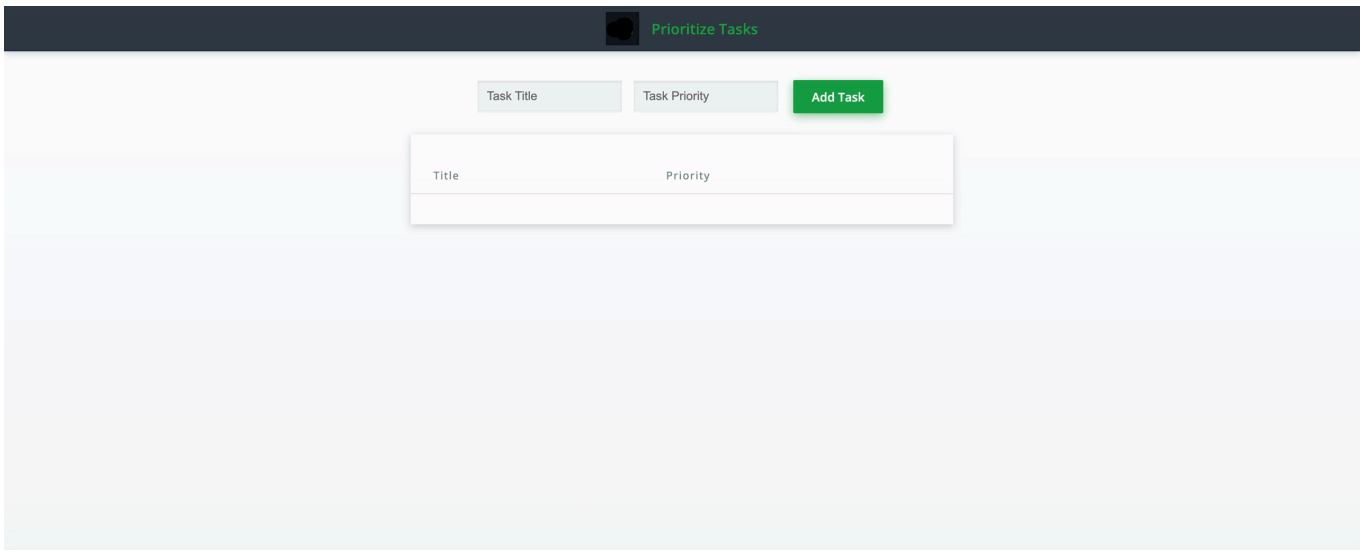
- Components have data-testid attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The files that should be modified by the candidate are `src/components/CustomerList.tsx` and `src/components/SearchCustomer.tsx`. They are open by default in the system editor.
- Avoid making changes to other files in the project structure.

Question - 2

React(TypeScript): Prioritize Tasks

Complete the component as shown to pass the test cases. Certain core React functionalities are already implemented.

Hide animation Show animation



The application has only one component, `Tasks`, where all the functionalities will be implemented.

The component must have the following functionalities:

- The type of input for the:
  - Task Name input box should be "text".
  - Task Priority input box should be "number".
- The initial view should have no tasks in the table.
- Clicking the `Add` button should:
  - add a task and its priority in the table and sort the tasks according to priority.
  - display an alert saying "Please enter both title and task priority" if any input boxes are empty.
  - reset the input boxes to empty after adding a valid task to the table.
- Clicking the `Delete` button should delete the corresponding task from the table.

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

Attribute	Component
input-task-name	Input box for task name
input-task-priority	Input box for task priority

submit-button	Button for adding task
tasksList	List of tasks

Note:

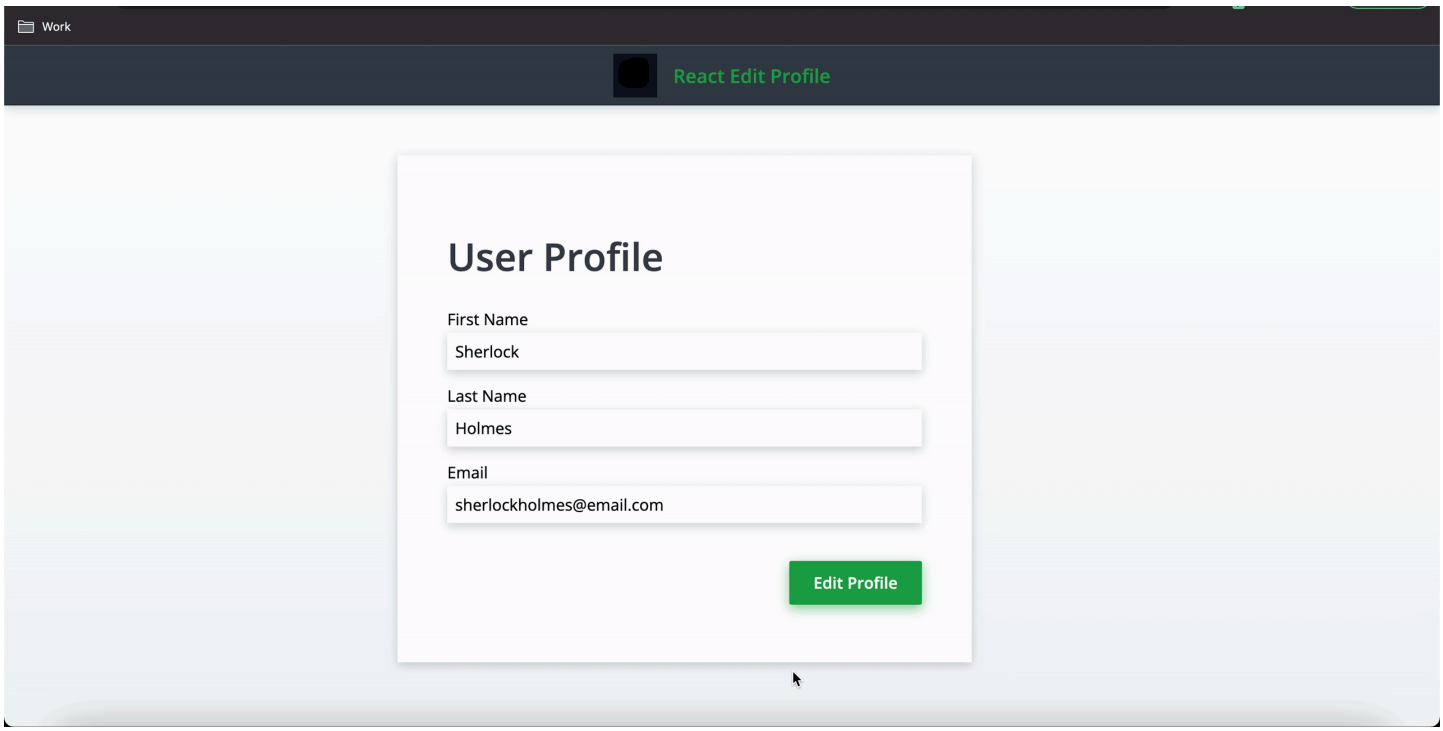
- Components have data-testid attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The file that should be modified by the candidate `src/components/Books/index.tsx` is open by default in the system editor.
- Avoid making changes to other files in the project structure.

Question - 3

React(TypeScript): Edit Profile

Complete a React Edit Profile application as shown in order to pass all the unit tests. Certain core React functionalities are already implemented.

Hide animation Show animation



The application has only 1 component:

- The Profile component enables the user to update his/her details.

The application has the following functionalities:

- The Profile component shows the user their details and allows editing.
- Initially, there should be 3 divs with the following values:
  - First Name: **Sherlock**
  - Last Name: **Holmes**
  - Email: **sherlockholmes@email.com**
- Initially, the button should have the text value Edit Profile.
- The following functionality should be implemented when the user clicks the 'Edit Profile' button.
  - The user can now edit the details in the divs.
  - The button should now have the text value "Save Changes".
- The following functionality should be implemented when the user edits the details and clicks the Save Changes button.
  - If one or more of the fields is empty, there should be an alert prompt with the text "Please enter all profile fields" and no values should be updated.
  - If all the fields are non-empty, there should be an alert prompt with the text "Profile updated successfully".
  - The input values should be the updated values and should be in uneditable divs.
  - The button text should be Edit Profile.

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

Description	Name
First Name	firstNameDiv
Last Name	lastNameDiv
Email	emailDiv
First Name <input>	firstNameInput
Last Name <input>	lastNameInput
Email <input>	emailInput
<button>.	changeButton

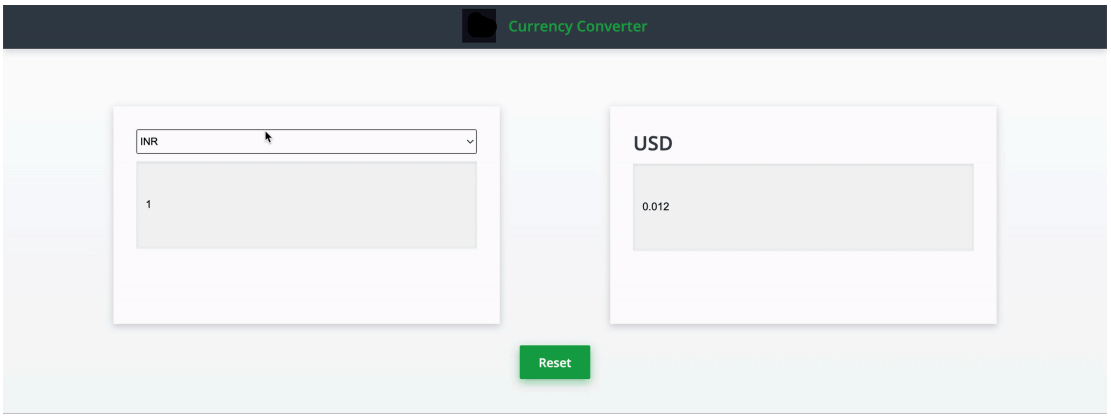
The component has data-testid attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.

Question - 4

React(TypeScript): Currency Converter

Complete the currency converter component that converts the value in any of the given currencies to USD as shown.

Hide animation Show animation



The list of available currencies is imported as "exchangeRates" in the component that can be used to render the list of currencies. The initial values for the currency and the input box are also imported from "exchangeRates.ts".

It is given that:

- The first input box is to be used to take input.
- The second input box is read-only to display the converted value.
- Only positive integer or decimal values can be taken as input.

The component must have the following functionalities:

- The default value for:
  - input box should be 1.
  - the currency to display in the select menu should be INR.
  - The second input box should be the conversion of 1 INR to USD i.e. 0.012.
- The select menu should have five currencies in the options.

- The conversion should happen automatically:
  - when there is any input change.
  - on selecting another currency.
- The converted value should be up to 3 decimal places.
- The placeholder text should change automatically according to the currency.
- The Reset button should reset the input to 1 and give the conversion in the current currency.

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

Component	Attribute
select menu	select-currency
first input box	input-value
second input box	converted-value
Clear Input button	clear-value

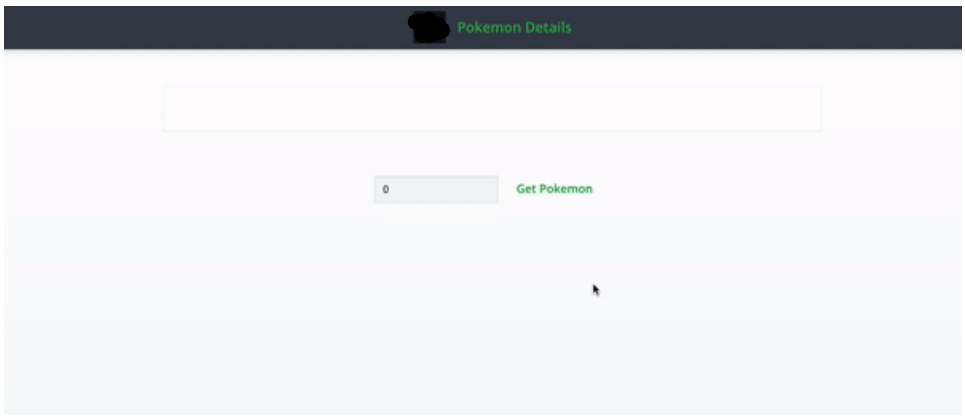
Note:
- Components have data-testid attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The only file that should be modified by the candidate is "src/components/currency-converter/index.tsx". It is open by default in the system editor.
- Avoid making changes to other files in the project structure.

Question - 5
React(TypeScript): Pokemon Details

After twenty-one years, Ash Ketchum was finally a Pokémon Champion. He decided to retire from Pokemon training and started software development. He is creating a pokemon search application, which takes a pokemon id as input and searches the pokemon database for the pokemon and its evolved and primitive versions.

Help Ash to complete the React Components as shown below.

Hide animation Show animation



The component should have the following functionalities:
- The input field should take the pokemon id as a number.
- The id should be in the range of 1 to 151. Otherwise, the component should show an alert with the text 'Pokemon ID must be between 1 and 151'.
- On getting a valid pokemon id, the component should make a GET request to the following API endpoint 'https://jsonmock.hackerrank.com/api/pokemon?id=<id>' which returns pokemon data in JSON format.
  - e.g. to get Pokemon with id = 1, the API endpoint will is https://jsonmock.hackerrank.com/api/pokemon?id=1
  - For an API call with a valid pokemon ID, the response will be:

```
{
  "data":{
    "id":1,
    "num":"001",
    "name":"Bulbasaur",
    "type":[
      "Grass",
      "Poison"
    ],
    "height":"0.71 m",
    "weight":"6.9 kg",
    "candy":"Bulbasaur Candy",
    "candy_count":25,
    "egg":"2 km",
    "spawn_chance":0.69,
    "avg_spawns":69,
    "spawn_time":"20:00",
    "multipliers":[1.58],
    "weaknesses":[
      "Fire",
      "Ice",
      "Flying",
      "Psychic"
    ],
    "next_evolution":[
      {
        "num":"002",
        "name":"Ivysaur"
      },
      {
        "num":"003",
        "name":"Venusaur"
      }
    ]
  }
}
```


- If the pokemon contains the next evolved version, the 'Next Evolution' button should be enabled.
- If the pokemon contains a previous evolved version (primitive version), the 'Prev Evolution' button should be enabled.
- On clicking the 'Next Evolution' button, the next evolved version of the pokemon should be fetched from the API and loaded.
- On clicking the 'Prev Evolution' button, the previous version of the pokemon should be fetched from the API and loaded.
- The Pokemon.tsx component should be used to render all the pokemon data by passing the pokemon as props.

Note:

- 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.
- The only file that should be modified by the candidate is "/src/components/pokemon-details/PokemonDetails.tsx", "src/components/pokemon/Pokemon.tsx" and is open by default by the system editor.
- It is advised for candidates to avoid making any changes to the rest of the files in the project structure.