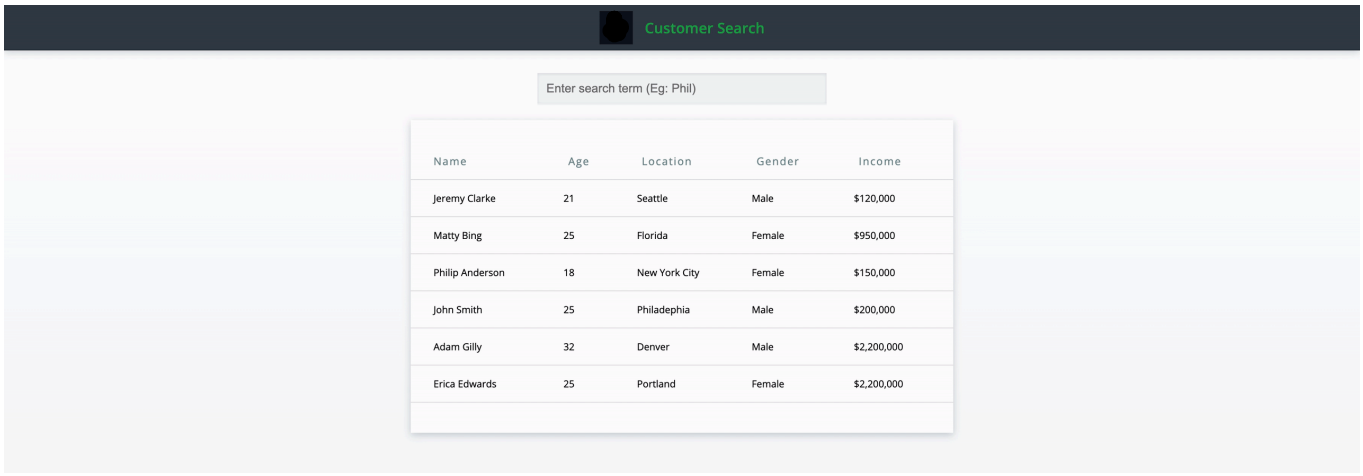
Question - 1

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:

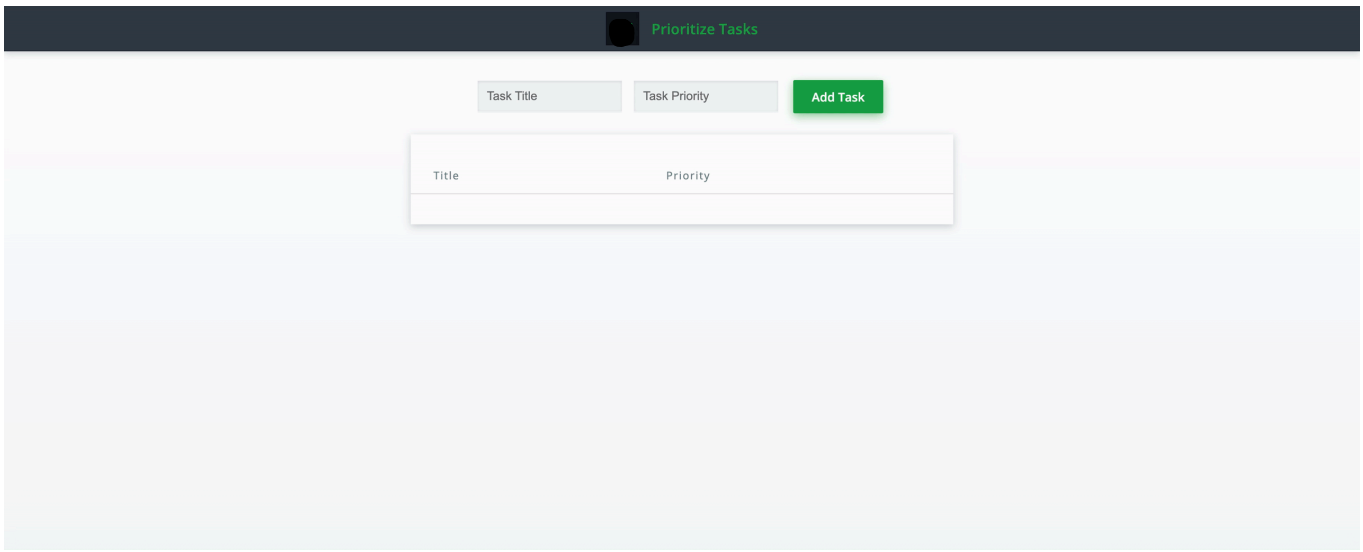
- Components have data-testid attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The files that should be modified by the candidate are `src/components/CustomerList.tsx` and `src/components/SearchCustomer.tsx`. They are open by default in the system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

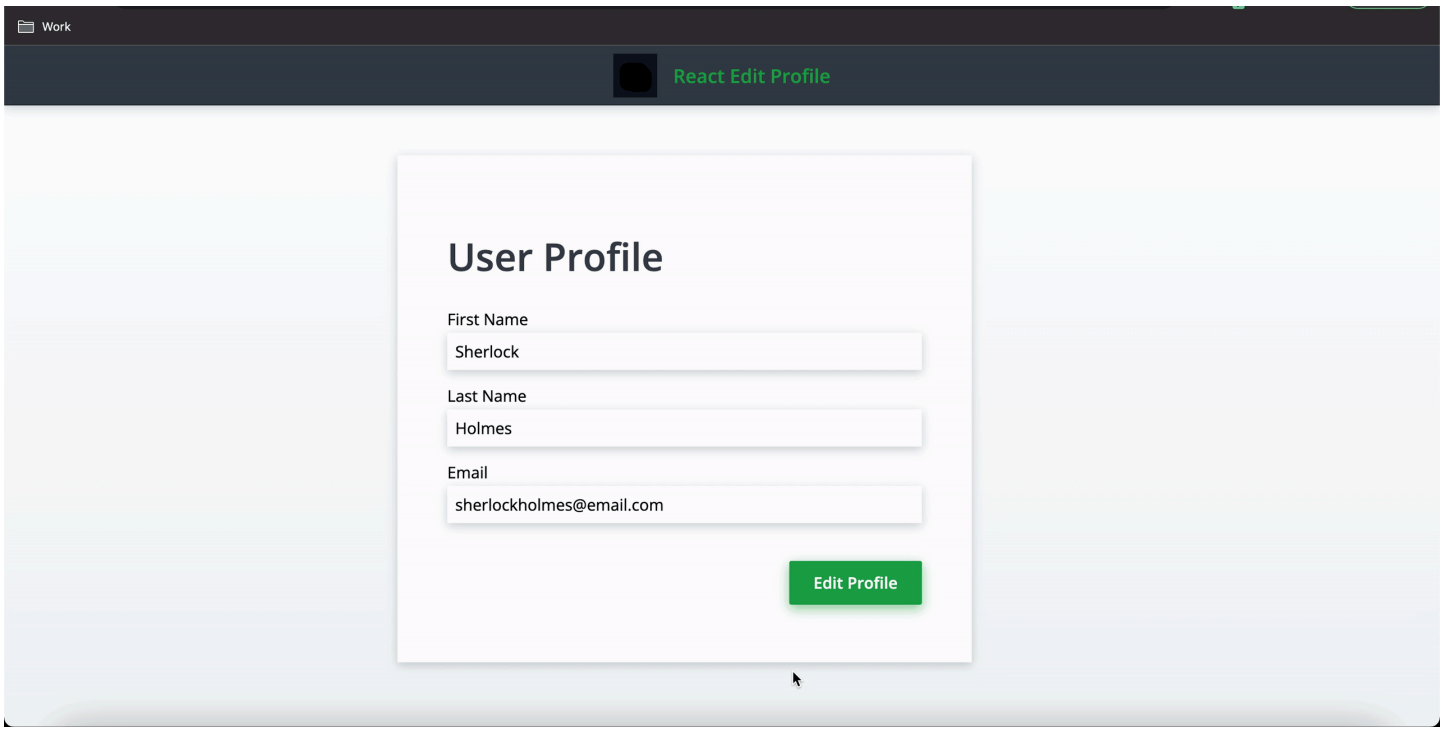
- Components have data-testid attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The file that should be modified by the candidate `src/components/Books/index.tsx` is open by default in the system editor.
- Avoid making changes to other files in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

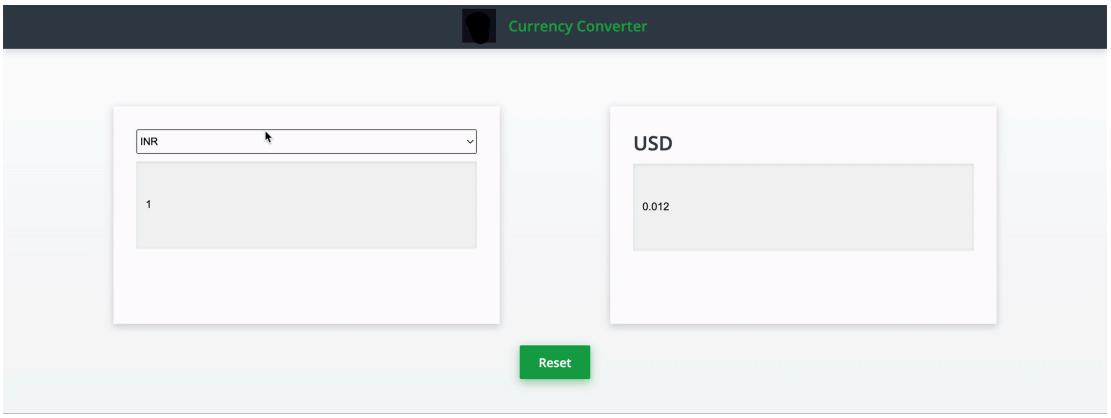
The component has data-testid attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.

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

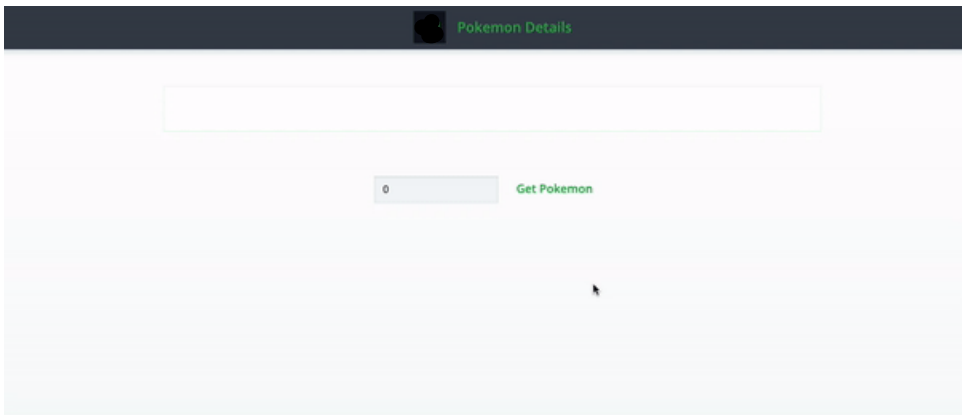
Note:
- Components have data-testid attributes for test cases and certain classes and ids for rendering purposes. They should not be changed.
- The only file that should be modified by the candidate is "src/components/currency-converter/index.tsx". It is open by default in the system editor.
- Avoid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