Sample Exam Web Development

Setup template

- `npm install` or `yarn install`

to run the react project:

- `npm run watch` or `yarn run watch`
- `dotnet run` (Inside a new terminal)

to run and test your typescript file

- run `yarn tsc src/ExamPart2.ts` command to compile the ts file.
- run `node src/ExamPart2.js` command to run the js file.

Your solutions

- **ExamPart1.tsx** should contaain answers of **Question 1,2,3** and **4**
- **ExamPart2.ts** should contain the answers of **Question 5**

Question 1: Display a single person object | 10 Points

Create a stateless React component named `PersonCard` as a **function component.** The component should accept a person as property and display the person's name and a role-based icon:

- 🎓 for Students.
- 🏫 for Teachers.
- 🏢 for Staff.

Also display additional details like:

- Grade for Students.
- Subjects for Teachers.
- Department for Staff.

Only render the properties if the person has them

Sample output: 🎓 John | Grade: 7,5 🏫 Steve | Subject: OODP, Web, Algorithms 🏢 Bob| Department: Tech Support

NOTE: Check the type implementation in ExamPart1.tsx file

Points distribution:

- All properties are rendered correctly 1pt
- Display emojis based on role 1pt
- Usage of interface/type for component properties. 2pt
- Implemented function/component for the icon and role switcher. 2.5pt

- Component should be a function component with no state 1pt
- Guard condition inside the render for properties that are not relevant based on the role. 2.5pt

HINT for GUID generation: `crypto.randomUUID()` can be used to generate GUIDs for testing purposes.

Question 2: Person registration form | 15 Points

Create a React class component named PersonForm that allows users to input information for creating a person.The form should collect the following details:

- Name: A text input field to enter the person's name.
- Role: A dropdown menu to select the role of the person, with three options: Student, Teacher, and Staff.
- Role-Specific Details: -- If the role is Student, a field to input the student's Grade. -- If the role is Teacher, a field to input the Subjects (comma-separated). -- If the role is Staff, a field to input the Department.

Upon clicking the "Submit" button, the form should display an alert with the following information:

- The name entered by the user.
- The selected role.
- The relevant additional details based on the selected role (e.g., grade for students, subjects for teachers, or department for staff).

The form should manage its state using React class components and display only relevant information.

Points distribution:

- All the form fields should be present. 5 in total. 1pt
- Guard conditions to hide irrelevant fields based on role 2.5pt
- Usage of interface/type for the form state 2.5pt
- The state should be updated using the onChange handler 2pt
- Their should be an external handler function for the state updaters 2pt
- Usage of `preventDefault` inside the submit handler 1pt
- Show the result inside an `alert box` 2pt
- Show only the relevant items inside the `alert box` 2pt

HINT for the dropdown list: `<select onChange={}> <option value="student">Student</option> <option value="teacher">Teacher</option> <option value="staff">Staff</option> </select>`

HINT for the `preventDefault`: is a method of Event interface which can be used in combination with a submit button.

HINT for the `alert box`: `alert("I am an alert box!")`

Question 3: Integration with the backend: 15 Points

Write a TypeScript function `fetchPersons` that fetches data from an API endpoint such as /api/persons/. The endpoint accepts an optional query parameter role. If role is specified, the API returns persons specific

to that role (Student, Teacher, or Staff). If role is not provided, the API returns all persons. The function should:

- Use `async` and `await`
- Return an `Option` type:
- `None` if an error occurs, including the HTTP status code and error message.
- `Some` if the request is successful, containing a list of Person objects or specific types like Student, Teacher, or Staff.
- Support optional filtering for the role parameter.

Points distribution:

- Usage of the async and await operator 5pt
- Error handling with either Option or Promises 3pt
- Has an optional role parameter 4pt
- The role parameter should be correctly passed to the url 3pt

Question 4: Render a list of persons: 15 Points

Create a React component named `PersonList` that:

- Accepts an array of `Person` objects as a prop.
- Uses the `PersonCard` component to render each person in the list. Not using `PersonCard` will result in a 5-point deduction.
- Provide a toggle button to sort the list by name in ascending or descending order.

HINT for the `sort`: `myarray = [1,2,3,4] myarray.sort(<a predicate>) // sorts the array based on the predicate`

Points distribution:

- Usage of interface/type for the properties. 4pt
- Persons should come from the props 5pt
- Re-use the `PersonCard` component 3pt
- Don't forget the key on the map after persons. 3pt

Question 5: Filter list: 15 Points

Write a TypeScript curried function `filterList` that:

- Accepts a lambda expression (predicate function) as the first argument.
- Accepts an array of generic objects (T[]) as the second argument.
- Returns a custom `List<T>`

NOTE: Check the type implementation in ExamPart2.ts file

Hint: Write some constructor functions to create objects of types Empty and Full.

An example call to the filterlist function: `filterList<number>((a:number)=>a%2 != 0) ([0,1,2,3,4,5,6,7,8,9])`

Output should look like: `{ kind: "node", value: 1, next:{ kind: "node", value: 3, next: { kind: "node", value: 5, next: { kind: "node", value: 7, next:{ kind: "node", value: 9,next:{ kind: "empty" } } } } } }`

Points distribution:

- Constructor functions are implemented. 5pt
- Predicate logic is used correctly. 5pt
- Function does not result in endless loop. 5pt