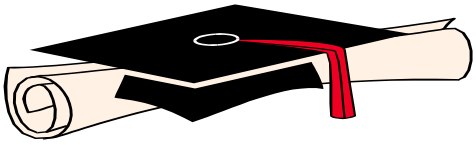
REACT

Semester: 1	**The Exam Code:** 01
Duration: 90 minutes	**Total Marks:** 20



Practical Examination Paper

Do not write on this question paper and return it to the Invigilator after the examination.

ReactJS Practical Assessment: Financial Transaction Management Application

Total Points: 20
Duration: 90 minutes

Overview

In this assessment, you will develop a financial transaction management application using ReactJS. The application will allow users to add, view, edit, delete, and sort transactions based on attributes like description, type, and amount.

Question 1: Setting up the Project and Main Layout (6 Points)

a) Project Setup and Basic Layout (3 Points)

Task:

- Create a React project named `transaction-manager`.
- Set up the main screen to display a list of transactions with columns: `Description`, `Type`, and `Amount`.
- Add a button labeled "Add New Transaction" to navigate to a form for adding a new transaction.

Instructions:

- Use `create-react-app` or another preferred setup method to create the project.
- Define a `Transaction` model with attributes for description, type, and amount.
- Implement the main screen as a `TransactionList` component and render placeholder data.

UI Sketch:

```
-----------------------------------------------
|       Financial Transaction Manager        |
-----------------------------------------------
| Description    | Type        | Amount      |
-----------------------------------------------
| Salary         | Income      | $3000       |
| Groceries      | Expense     | $200        |
```

```
| Investment     | Income      | $500        |
-----------------------------------------------
|                 [ Add New Transaction ]   |
-----------------------------------------------
```

Points Distribution:

- Project setup and package installation: **1 Point**
- Creating `TransactionList` component with placeholder data: **1 Point**
- "Add New Transaction" button functionality: **1 Point**

---

b) Implement Transaction Model (3 Points)

Task:
Define a `Transaction` data model with attributes `description`, `type`, and `amount`. Use this model to structure the application's data.

Points Distribution:

- Defining the `Transaction` model: **1 Point**
- Proper use of model attributes in `TransactionList`: **2 Points**

Question 2: Form for Adding and Updating Transactions (6 Points)

a) Create Add Transaction Form (3 Points)

Task:

- Create a form component named `AddTransactionForm` where users can input transaction details.
- The form should include:
  - A text field for the description.
  - A dropdown for the type (`Income` or `Expense`).
  - A number input for the amount.
  - A "Save" button to submit the form data.

Instructions:

- Validate inputs to ensure:
  - `Description` is not empty.

    - `Amount` is a positive number.
- Display an error message if validation fails.
- On successful submission, add the transaction to the list on the main screen.

UI Sketch:

```
-------------------------------------
|         Add New Transaction       |
-------------------------------------
| Description:  [_________________] |
| Type:         [ Income  v]        |
| Amount:       [___________]       |
|                                     |
|               [  Save  ]           |
-------------------------------------
```

Points Distribution:

- Correct input fields for description, type, and amount: **1 Point**
- Validation and error handling: **1 Point**
- Submission functionality with data addition: **1 Point**

---

b) Updating Transaction List on Submission (3 Points)

Task:

- After adding a transaction, return to the main screen and update the transaction list dynamically.
- Implement functionality using React state and `useState` or `useReducer`.

Points Distribution:

- Correct use of state to manage transaction list: **2 Points**
- Updating the transaction list after form submission: **1 Point**

Question 3: Editing and Deleting Transactions (6 Points)

a) Edit Transaction Functionality (3 Points)

Task:

- Enable editing for each transaction entry on the main screen.
- Add an "Edit" button next to each transaction that opens the `AddTransactionForm` pre-filled with the selected transaction's details.
- On form submission, update the existing transaction entry.

UI Sketch:

```
-----------------------------------------------
| Description    | Type        | Amount      | Actions   |
-----------------------------------------------
| Salary         | Income      | $3000       | [ Edit ]  |
| Groceries      | Expense     | $200        | [ Edit ]  |
| Investment     | Income      | $500        | [ Edit ]  |
-----------------------------------------------
```

Points Distribution:

- Correct handling of pre-filled form for editing: **1 Point**
- Updating transaction data on form submission: **2 Points**

---

b) Delete Transaction Functionality (3 Points)

Task:

- Add a "Delete" button next to each transaction that removes the transaction from the list.
- Confirm the deletion action with the user before removing the transaction.

Points Distribution:

- Correctly implementing deletion functionality: **2 Points**
- Confirmation dialog for deletion: **1 Point**

Question 4: Sorting Transactions by Amount (2 Points)

Task:

- Add sorting functionality to display transactions in descending order of the amount.
- Implement a button labeled "Sort by Amount" to toggle the sorting order.

UI Sketch:

```
-----------------------------------------------
|       Financial Transaction Manager       |
-----------------------------------------------
| [ Sort by Amount ]                          |
| Description    | Type        | Amount      |
-----------------------------------------------
| Salary         | Income      | $3000       |
| Investment     | Income      | $500        |
| Groceries      | Expense     | $200        |
-----------------------------------------------
```

Points Distribution:

- Implementing sort by amount: **1 Point**
- Toggle functionality for sorting order: **1 Point**

Instructions for Submission

- Organize React components logically in separate files.
- Use React hooks such as `useState` and `useEffect` where necessary.
- Include clear and concise comments explaining the logic.
- Submit all relevant files, including components, `App.js`, and `package.json`, in a zipped folder labeled: `ReactJS_FinanceExam_<YourName>.zip`.