Batch: T6

Practical No. 5

Title of Assignment: Study and implementation of ReactJs

Student Name: Parshwa Herwade

Student PRN: 22510064

Perform following problem statements using ReactJs

Problem Statement 0: Basics of ReactJs

1.What is React and what problem does it solve?

ANS. **React** is a JavaScript library developed by Facebook for building user interfaces, particularly single-page applications where you need a responsive and dynamic user experience. It solves several problems:

- **Component-Based Architecture**: React allows developers to build encapsulated components that manage their own state and compose them to make complex UIs. This promotes reusability and modularity.
- **Declarative Syntax**: React's declarative syntax allows developers to describe what the UI should look like for different states of the application, making the code more predictable and easier to debug.
- **Virtual DOM**: React uses a Virtual DOM to optimize updates to the real DOM. It efficiently updates the UI by comparing the Virtual DOM with the real DOM and applying only the necessary changes, which improves performance.
- **One-Way Data Binding**: React enforces a unidirectional data flow, making the data flow more predictable and easier to understand.

2.What are React components and how are they used?

ANS. **React Components** are the building blocks of a React application. Each component is a JavaScript function or class that optionally accepts inputs (known as "props") and returns a React element that describes how the UI should appear.

- **Function Components**: These are stateless components written as JavaScript functions. They can use hooks to manage state and side effects.
- **Class Components**: These are stateful components written as ES6 classes. They can hold local state and have lifecycle methods.

3.What is JSX in React?

ANS. **JSX** (JavaScript XML) is a syntax extension for JavaScript used in React. It allows you to write HTML-like code inside JavaScript files. JSX makes the code easier to read and write by combining HTML structure with JavaScript logic.

```
const element = <h1>Hello, world!</h1>;
```

JSX is transpiled into regular JavaScript function calls by tools like Babel. For example, the above JSX is transformed into:

```
const element = React.createElement('h1', null, 'Hello, world!');
```

4.What are props in React and how do they differ from state?

ANS. **Props** (short for properties) are read-only attributes passed from a parent component to a child component. They are used to pass data and event handlers down to child components. Props cannot be modified by the child component that receives them.

State is a local data storage that is managed within a component. Unlike props, state can be changed by the component itself, usually in response to user actions or other events. State changes are asynchronous and can trigger a re-render of the component.

Difference:

- **Props**: Read-only, passed from parent to child, used for data that doesn't change.
- **State**: Read-write, managed within the component, used for data that changes over time

5.What is state in React and how does it work?

ANS. **State** is an object that holds data that may change over the lifecycle of a component. It allows components to maintain internal data and manage it dynamically.

- **Class Components**: State is initialized in the constructor and can be updated using this.setState().

```
class MyComponent extends React.Component {
 constructor(props) {
  super(props);
  this.state = { count: 0 };
 }

 increment = () => {
  this.setState({ count: this.state.count + 1 });
 };
```

```
  render() {
    return (
      <div>
        <p>{this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
  }
}
```

<u>Function Components</u>: State is managed using the useState hook.

```
import React, { useState } from 'react';

function MyComponent() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>{count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

6.What are React lifecycle methods, and why are they important?

ANS. **Lifecycle methods** are hooks in class components that allow you to run code at specific points in a component's lifecycle. They are crucial for performing operations such as fetching data, setting up subscriptions, and cleaning up resources.

- **Mounting**: Methods like componentDidMount() run after a component is added to the DOM.
- **Updating**: Methods like componentDidUpdate() run after the component's state or props have changed.

- **Unmounting**: Methods like componentWillUnmount() run before the component is removed from the DOM.

7. Elaborate following with respect to ReactJs

o **Event Handling**:

- **React's Synthetic Event System**: React uses a synthetic event system to handle events in a cross-browser compatible way. This system is a wrapper around the browser's native events and provides a consistent API.
- **Event Handlers**: Event handlers are attached directly to JSX elements and can be passed as props. Common events include onClick, onChange, onSubmit, etc.

o **Conditional Rendering**:

- **Rendering Based on Conditions**: React allows conditional rendering using JavaScript operators. This makes it easy to display different elements or components based on certain conditions.
- **Techniques**:
  - **Inline Conditional Rendering**: Using JavaScript operators like && and ternary operators.
  - **Conditional Statements**: Using if statements inside render methods or function bodies.

o **Lists and Keys**:

- **Rendering Lists**: You can render lists of items by using the map() method. Each item in the list should have a unique key prop to help React identify which items have changed.
- **Keys**: Keys are unique identifiers that help React optimize the rendering process by efficiently updating and reordering items.

o **Forms**:

- **Controlled Components**: In React, form elements are typically controlled components, meaning their value is controlled by the component's state.
- **Handling Input**: Use state to manage form data and handle form submission with event handlers.

o **Hooks**:

- **Introduction**: Hooks are functions that allow you to use state and other React features in functional components.

- **Common Hooks**:
    - **useState**: Manages local state.
    - **useEffect**: Handles side effects such as data fetching and subscriptions.
    - **useContext**: Accesses context values.
    - **useReducer**: Manages state logic with reducers.

o **React Router**:

- **Purpose**: React Router is a library for handling routing in React applications. It allows you to define multiple routes and navigate between them.
- **Components**:
    - **BrowserRouter**: The router that uses HTML5 history API.
    - **Route**: Defines a path and the component to render.
    - **Link**: Creates navigable links.

o **State Management**:

- **Purpose**: State management involves handling and updating the state of an application. React offers various ways to manage state, including local state, context, and external libraries.
- **Options**:
    - **Local State**: Managed within components using useState or class-based state.
    - **Global State**: Managed using libraries like Redux or MobX.
    - **Context API**: For sharing state across components without prop drilling.

o **React Context API**:

- **Purpose**: The Context API provides a way to share state between components without having to pass props manually through every level of the component tree.
- **Creating Context**:
    - **React.createContext**: Creates a context object.
    - **Context.Provider**: Provides context values to its descendants.
    - **Context.Consumer**: Consumes context values in a component.

8.How can you optimize the performance of a React application?

ANS.

- Use Memoization: Prevent unnecessary re-renders with React.memo, useMemo, and useCallback.
- Code Splitting: Load components lazily with React.lazy() and Suspense.
- Avoid Anonymous Functions: Use useCallback to avoid recreating functions.
- Optimize Component Rendering: Use React.PureComponent or hooks to reduce renders.
- Use Virtualization: Render large lists efficiently with react-window or react-virtualized.
- Optimize Images and Assets: Use optimized formats and lazy load images.
- Reduce Unnecessary State Updates: Batch updates and minimize state changes.
- Minimize Reconciliation: Ensure list keys are unique and stable.
- Server-Side Rendering (SSR): Use Next.js for faster initial loads and better SEO.
- Profiling and Monitoring: Identify bottlenecks with React Profiler and DevTools.

Problem Statement 1: Star Wars character app

(In this problem statement, example of Star Wars is given, you may choose any characters

from the series of the movie like Harry Potter, etc. Every group in a batch will have

different characters.)

Using a public API, display a list of all Star Wars characters using the endpoint

“/people”. The API has paging, so the developer must also implement pagination.

Also, a simple loader for fetching/refetching data as well as handling the error

state (i.e., if the API server is down).

For every user, we’d like to display a card with the name of each character along

with a random picture for each character (see Picsum photos for random picture

inspiration). Each character card should be colored based on their species and

have some kind of animation when the user hovers over the card. When we click

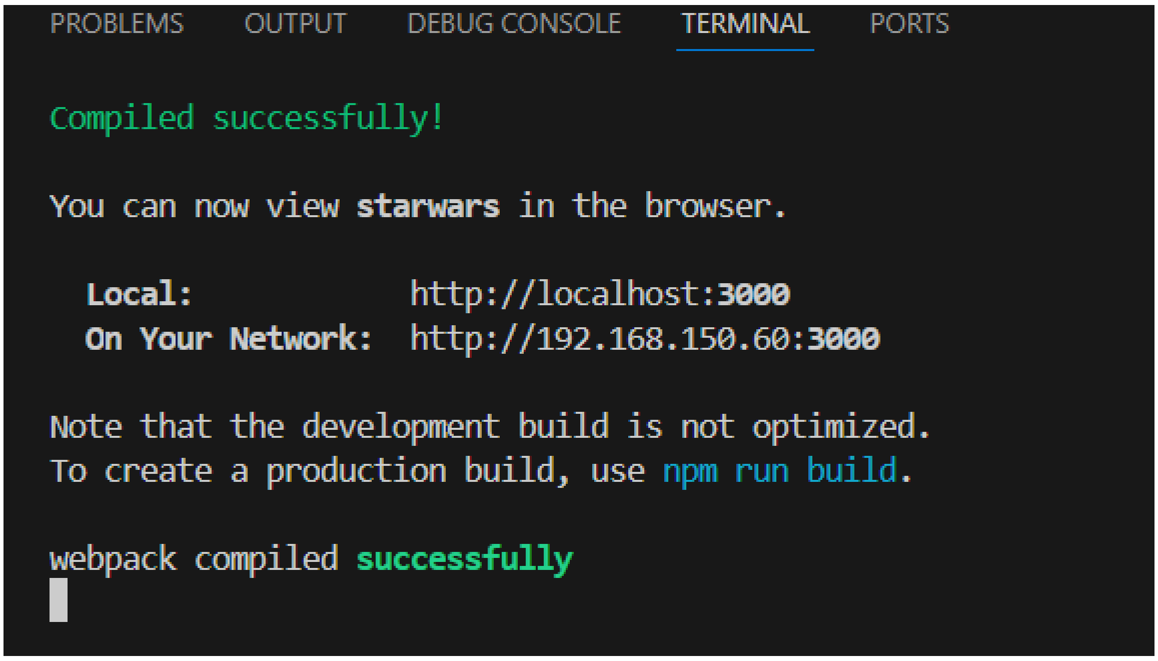
on a character’s card, more information should appear in a modal about the

character.

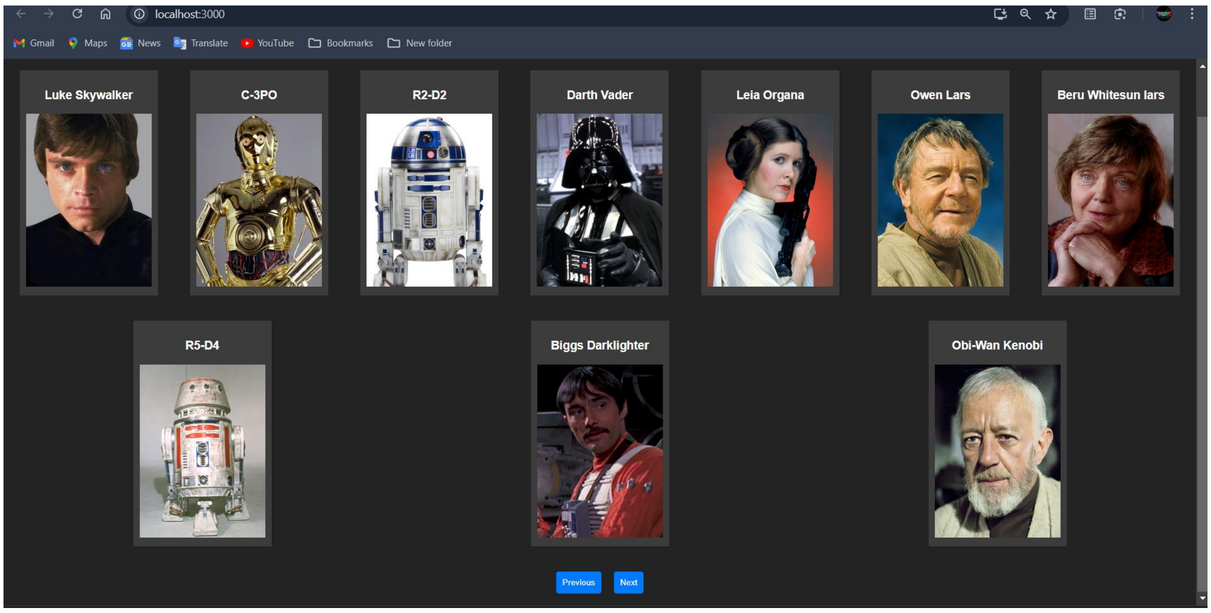
In the character details modal, we’d like to display information about the person:

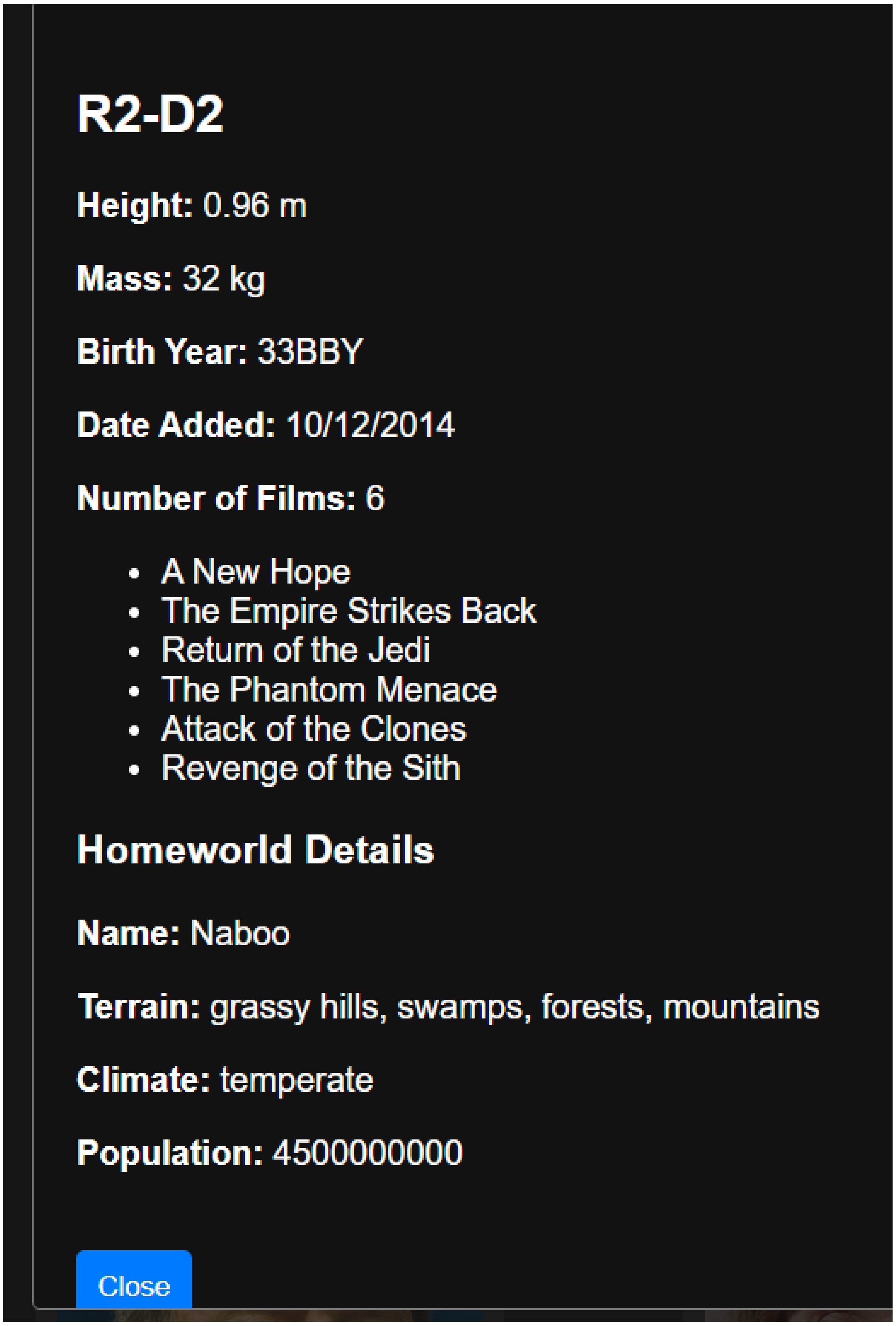
name as the header of the modal, height displayed in meters, mass in kg, date person was added to the API (in dd-MM-yyyy format), number of films the person appears in and their birth year. We should also fetch information about the person's homeworld and display its name, terrain, climate, and amount of residents.

ANS.



```
Compiled successfully!

You can now view starwars in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.150.60:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```





R2-D2

Height: 0.96 m

Mass: 32 kg

Birth Year: 33BBY

Date Added: 10/12/2014

Number of Films: 6

- A New Hope
- The Empire Strikes Back
- Return of the Jedi
- The Phantom Menace
- Attack of the Clones
- Revenge of the Sith

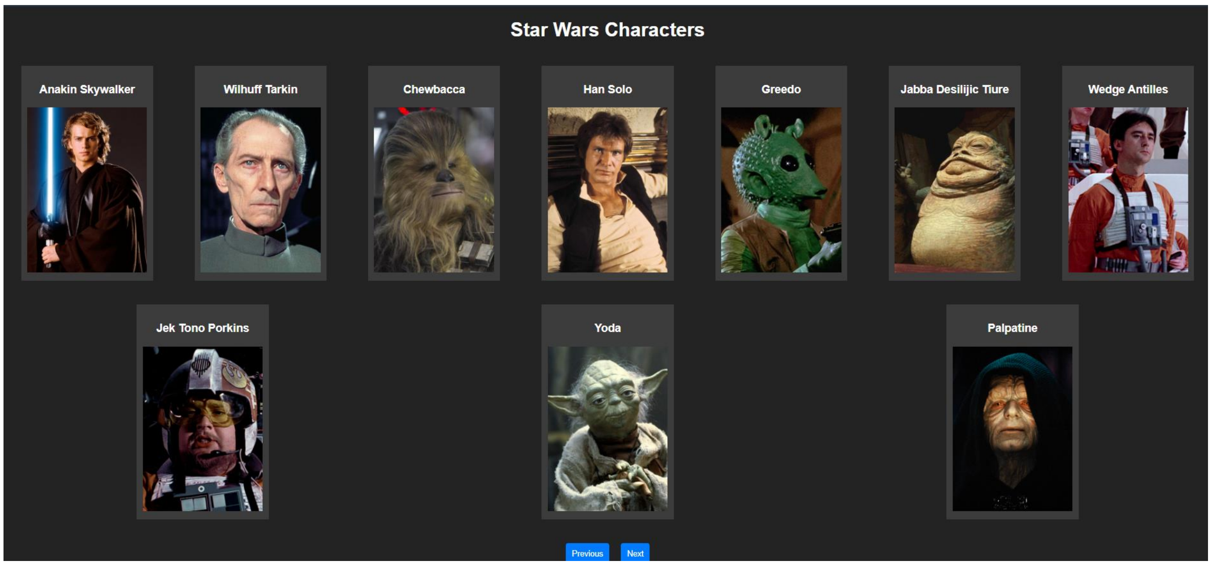
Homeworld Details

Name: Naboo

Terrain: grassy hills, swamps, forests, mountains

Climate: temperate

Population: 4500000000


Star Wars Characters
Anakin Skywalker
Wilhuff Tarkin
Chewbacca
Han Solo
Greedo
Jabba Desilijic Tiure
Wedge Antilles
Jek Tono Porkins
Yoda
Palpatine
Previous
Next