**Question 1:**

It is a common practice to add new features in ReactJS applications using components. The commonly used structures are functional and class components. Explain the syntax of functional and class components in ReactJS with a simple code snippet.

A screenshot of a computer program

Description automatically generated

**Question 2:**

Explain the process of embedding user interface template in Reactjs with a simple code

snippet.

1. Create a React Component: Define a component to hold the template.

2. Define the Template: Use JSX syntax to describe the structure and content of the UI.

3. Customize the Template: Add dynamic content using props or state values.

4. Render the Component: Use the component as a child element in another component.

5. Mount the Component: Render the root component in the DOM.

6. Transpile JSX: Convert JSX code to plain JavaScript using tools like Babel.

7. Execute and Render: React handles efficient rendering and updates.

import React from 'react';

class UIComponent extends React.Component {

render() {

return (

<div>

<h1>Hello, World!</h1>

<p>This is a sample UI component.</p>

</div>

);

}

}

export default UIComponent;

In the code snippet above, we define a React component called UIComponent by extending the React.Component class. Within the render() method of the component, we define the template using JSX syntax.

The template consists of a div element that wraps the content. Inside the div, we have an h1 element displaying the text "Hello, World!" and a p element with the message "This is a sample UI component."

To use this component in another part of your application, you can import it and render it as a child element within another component. For example:

import React from 'react';

import UIComponent from './UIComponent';

function App() {

return (

<div>

<h1>My App</h1>

<UIComponent />

</div>

);

}

export default App;

**Question 3:**

Describe the importance of implementing security measures in React applications. Discuss at least three security features that can help address potential security risks and vulnerabilities. Provide examples of how these features can enhance the security of a React application.

A close-up of a document

Description automatically generated

**Practical Performance: Building a Simple Todo App**

**Business Case:**

You are working for a small startup that wants to build a simple web application for managing todo items. The application should allow users to create, read, update, and delete todo items. The application should have a clean and user-friendly interface and should be built using ReactJS. The startup also wants the application to be thoroughly tested to ensure that it is reliable and free of bugs.

**Task:**

Your task is to build the todo application using ReactJS, and to thoroughly test it to ensure that it is reliable and free of bugs. The application should have the following features:

* A user can add a new todo item to the list
* A user can mark a todo item as completed
* A user can edit the title of a todo item
* A user can delete a todo item from the list

You can use the following APIs for managing the ToDo Items. You would need to write the code for the APIs. ***For the UI, you can assume a simple UI to write the code.***

**GET /api/todos**

* Returns a list of all todo items

**GET /api/todos/:id**

* Returns the todo item with the specified id

**POST /api/todos**

* Creates a new todo item
* Request body should contain the following properties:
  + title (string)
  + description (string)
  + dueDate (date)
  + completed (boolean)

**PUT /api/todos/:id**

* Updates the todo item with the specified id
* Request body should contain the following properties:
  + title (string)
  + description (string)
  + dueDate (date)
  + completed (boolean)

**DELETE /api/todos/:id**

* Deletes the todo item with the specified id

**Deliverables:**

You should submit the following deliverables:

* A working todo application that meets the above requirements and is built using ReactJS.
* A suite of automated tests that thoroughly test the application and ensure that it is reliable and free of bugs.

**Part 1: Application Development**

**Question 1a**

Describe the application development tools and methodologies you would use to build the todo management web application in ReactJS.

|  |
| --- |
| To build the todo management web application in ReactJS and ensure that it meets the agreed specifications, you would typically use a combination of development tools and methodologies. Below is the description and demonstration of some tools and methodologies you can employ:  1. Code Editor: You can use popular code editors such as Visual Studio Code, Atom, or Sublime Text to write your ReactJS code. These editors provide syntax highlighting, auto-completion, and other useful features for efficient coding.  2. Package Manager: Utilize a package manager like npm (Node Package Manager) or Yarn to manage your application's dependencies and install required packages. These tools allow you to easily add, remove, and update packages, ensuring consistent and reliable development.  3. ReactJS: As mentioned in the provided information, the todo application is built using ReactJS. React is a widely used JavaScript library for building user interfaces. It provides a component-based architecture, making it easy to develop reusable and modular UI components. |

**Question 1b**

Demonstrate with the appropriate endpoints how would you ensure that the application meets the agreed specifications? Focusing on code bundling, security of your API.

|  |
| --- |
| Here's an example of how the main code block could be expanded upon:  The below is a sample answer. The actual answer of the learner may differ but should cover the requirements as stated in the question.    **Endpoints and Responses**  **GET /todos**  **Response:**  [  {  "id": 1,  "text": "Buy groceries",  "completed": false  },  {  "id": 2,  "text": "Pay bills",  "completed": true  }  ]  **POST /todos**  **Request:**  {  "text": "Do laundry"  }  **Response:**  {  "id": 3,  "text": "Do laundry",  "completed": false  }  **PUT /todos/:id**  **Request:**  {  "completed": true  }  **Response:**  {  "id": 2,  "text": "Pay bills",  "completed": true  }  **DELETE /todos/:id**  **Response:**  {  "success": true  }  In this example, the `handleAddTodo` function generates a unique ID for each new todo item and sets the initial `completed` value to `false`. The `handleDeleteTodo` function filters out the todo item with the specified ID from the `todos` array.  To ensure that the application meets the agreed specifications, you can follow these steps:  1. Requirement Gathering: Understand the exact requirements and specifications of the application from stakeholders or product owners. Document the requirements to have a clear understanding of what needs to be implemented.  2. Test-Driven Development (TDD): Use TDD methodologies to write test cases before implementing the corresponding functionality. This ensures that the code meets the requirements and helps catch any potential bugs early in the development process.  3. Unit Testing: Write comprehensive unit tests for each component and functionality of the application. Utilize testing frameworks like Jest and testing libraries like React Testing Library to write and execute the tests.  . |

**Question 1c**

Demonstrate how you will add a new feature called “ToDoHeader” and modify TodoList.css to your ToDo ReactJS App. Also show the directory structure that adheres to the application development standard

|  |
| --- |
| **The actual structure can vary, but here is the example of the folder structure**  **└── src**  **├── components**  **│ ├── TodoApp.js**  **│ ├── TodoList.js**  **│ ├── TodoItem.js**  **│ ├── TodoForm.js**  **│ ├── TodoFilter.js**  **│ └── TodoStats.js**  **├── styles**  **│ ├── TodoApp.css**  **│ ├── TodoList.css**  **│ ├── TodoItem.css**  **│ ├── TodoForm.css**  **│ ├── TodoFilter.css**  **│ └── TodoStats.css**  **└── App.js**  **└── index.js**   * src: This is the main folder where all the application source code resides. * components: This folder contains all the React components used in the Todo app. * TodoApp.js: The main Todo app component that serves as the entry point for the Todo app. * TodoList.js: Component responsible for rendering the list of todos. * TodoItem.js: Component representing an individual todo item. * TodoForm.js: Component for adding new todos. * TodoFilter.js: Component for filtering the todos based on their status. * TodoStats.js: Component for displaying statistics related to the todos. * styles: This folder contains the CSS files for styling the Todo app components. * App.js: The main component that acts as the container for the Todo app. * index.js: The entry point of the application where the ReactDOM renders the app component into the DOM**.**   In the above code, we define the TodoHeader component, which renders the header of the Todo app. It displays the title "Todo App" using an **h1** element.  To integrate the TodoHeader component into the existing Todo app, you need to make the following changes:  App.js:  import React from 'react';  import TodoHeader from './components/TodoHeader';  import TodoList from './components/TodoList';  import TodoForm from './components/TodoForm';  const App = () => {  return (  <div>  <TodoHeader />  <TodoForm />  <TodoList />  </div>  );  };  export default App;  css Files:  .todo-list {  list-style: none;  padding: 0;  margin: 0;  }  .todo-item {  display: flex;  align-items: center;  margin-bottom: 10px;  }  .todo-item.completed {  text-decoration: line-through;  color: gray;  }  .todo-item .checkbox {  margin-right: 10px;  }  .todo-item .text {  flex-grow: 1;  } |

**Question 2**

Explain how you would adhere to organizational standards in application development and documentation while building the todo management web application. How would you document the internal design of the application for future maintenance and enhancement?

|  |
| --- |
|  |

**Question 3a**

Discuss the tools and techniques you would utilize for coding and programming the student management web application.

|  |
| --- |
|  |

**Question 3b**

Discuss and show code snippets how would you employ basic debugging tools and techniques to identify and resolve application errors or problems?

|  |
| --- |
| Below are some code snippets showcasing tools and techniques that can be utilized for coding and programming the todo management web application, including basic debugging tools and techniques:    ```    5. Debugging Tools and Extensions:  Using React DevTools browser extension provides insights into the component hierarchy and state changes. For example, you can inspect the props and state of a specific component:    These code snippets demonstrate the practical implementation of the mentioned tools and techniques for coding and programming the todo management web application. They can aid in identifying and resolving application errors or problems effectively during development. |

**Part 2: Automated Testing**

**Question 4**

Explain and demonstrate concisely the software testing process you would follow to ensure the reliability and bug-free nature of the todo management web application. Show unit testing codes for defects/errors/potential security vulnerabilities checking through software tests.

|  |
| --- |
| To ensure the reliability and bug-free nature of the todo management web application, you can follow a software testing process that includes various stages and techniques. Below is an explanation of the process, along with code for unit testing to check defects, errors, and potential security vulnerabilities:  1. Test Planning and Design:  - Identify the components and functions that need to be tested.  - Create unit test cases to validate the behavior of individual functions or modules.  - Focus on testing different scenarios, boundary cases, and error conditions.  2. Unit Testing:  - Use a testing library such as Jest to write unit tests.  - Write test cases for each function or component to cover different code paths and scenarios.  - Use assertions to verify the expected behavior and outcomes.  - Incorporate mocking and stubbing to isolate dependencies and control inputs/outputs.  Here are code examples demonstrating unit tests for defects, errors, and potential security vulnerabilities:  A screenshot of a computer program  Description automatically generated  A screenshot of a computer program  Description automatically generated  A screenshot of a computer program  Description automatically generated  // Attempt to add a potentially malicious script  fireEvent.change(inputElement, { target: { value: '<script>alert("XSS");</script>' } });  fireEvent.click(addButtonElement);  expect(mockOnDeleteTodo).not.toHaveBeenCalled();  });  ```  In these examples, we demonstrate unit tests for different scenarios, including adding a todo item, deleting a todo item, and input validation/security. The tests validate the expected behavior and outcomes, ensuring that defects, errors, and potential security vulnerabilities are identified and addressed through the software tests. |

**Question 5**

Describe commonly encountered errors that may occur during the development of the todo management web application. How would you address and make simple revisions or modifications to the application to resolve these errors?

|  |
| --- |
| During the development of the todo management web application, several commonly encountered errors may occur. Below are examples of such errors and how you can address them by making simple revisions or modifications to the application:  1. TypeError: Cannot read property '...' of undefined:  - This error occurs when trying to access a property or method of an undefined or null value.  - To address this error, you can add conditional checks or provide default values to handle potential undefined or null values. For example:  ```jsx  const title = todo ? todo.title : '';  ``` |

**Submission Guidelines:**

* Submit your code for the todo application and the tests in a **single zip file.**
* **Remember to remove node\_modules folder before creting a zip file**
* Provide clear instructions for running the application and the tests.
* Ensure that your code is well-documented and easy to understand.