# Lab 4 Questions

Q1

In React, a prop is used to pass data from a parent component to a child component. When a prop is passed to a child component, the data is read-only. Here is an example of how to use props in a React component.

*// parent component*

*class App extends React.Component {*

*render() {*

*return (*

*<div>*

*<h1>Your Notes</h1>*

*<NotesList notes={this.state.notes} />*

*</div>*

*);*

*}*

*}*

*// child component*

*class NotesList extends React.Component {*

*render() {*

*return (*

*<ul>*

*{this.props.notes.map((note) =>*

*<li key={note.id}>{note.title}</li>*

*)}*

*</ul>*

*);*

*}*

*}*

The code example above demonstrates the use of props in a notes application. The App component is the parent component and the NotesList component is the child component. The App component passes the notes prop to the NotesList component. The NotesList component can now see all the notes within the application and renders a list of all the notes titles.

In React a State is used to store data inside a component. States are initialized and managed by the component directly. Components should only modify and update a state using setState(). Here is an example on how to use states in React component.

*function App() {*

*const [notes, setNotes] = useState([]);*

*const [currentNote, setCurrentNote] = useState({*

*title: '',*

*text: ''*

*});*

*function inputHandler(e) {*

*setCurrentNote({*

*...currentNote,*

*[event.target.name]: e.target.value*

*});*

*}*

*function submitHandler(e) {*

*e.preventDefault();*

*setNotes([...notes, currentNote]);*

*setCurrentNote({*

*title: '',*

*text: ''*

*});*

*}*

*return (*

*<div>*

*<h1>My Notes</h1>*

*<form onSubmit={submitHandler}>*

*<input*

*type="text"*

*name="title"*

*value={currentNote.title}*

*onChange={inputHandler}*

*/>*

*<textarea*

*name="text"*

*value={currentNote.text}*

*onChange={inputHandler}*

*/>*

*<button type="submit">Add Note</button>*

*</form>*

*</div>*

*);*

*}*

The code example above demonstrates the use of states in a notes application. The App component contains 2 states, notes and currentNote. The inputHandler and the submitHandler contains setState function for these states and will use them to update the states accordingly. The inputHandler will updatethe current notes details as the user is typing in the form. The setState will update the notes array and add in the new note submitted by the user.

Q2

In functional programming, a functor is a container of values which can be used with the “map” function. A functor can be mapped upon by a unary function.

*[1,2,3,4].map(multiplyBy2)*

*//=> [2,4,6,8] or map(multiplyBy2, [1,2,3,4])*

*//=> [2,4,6,8] where multiplyBy2 = x => x \*2 and map = (fn, arr) => arr.map(fn)*

In the JavaScript code example above the map function takes a function multiplyBy2 and a list of values as its arguments. The multiplyBy2 function is applied to each value in the list and produces a new list with each value multiplied by 2.

Q3

**Streams**

Advantages: streams are a more declarative, expressive style. Declaring your intent in code is sometimes considered better than explaining how it is done. Streams also encourage looser coupling. Stream-handling code does not need to know the source of the stream or its eventual terminating method.

Disadvantages: Performance is one of the main disadvantages of Streams. Streams are not good for raw speed memory usage.

**Callbacks**

Advantages: Callbacks allow you to wait for a previous operation to complete before executing the next operation.

Disadvantages: Callback hell consists of nested callbacks where each function is waiting on the previous. Callback hell causes issues with readability and maintainability of code.

**Promises**

Advantages: Can handle multiple async operations in a better way. Good code readability and good error handling.

Disadvantages: More difficult to use then callbacks

Q4

Diagram

Description automatically generated

***Content area***: This area contains the body of the element. The body can include text, images videos etc. The dimension of this are consists of the content’s height and width. It can also have a background color or background image. Other properties include, width, min-width, max-width, height, min-height, and max-height.

***Padding area***: This area extends the content area to include the element’s padding. Its dimensions are the padding-box width and padding-box height. The thickness of the padding is determined by the padding-top, padding-right, padding-bottom, padding-left and a general term padding which applies to all.

***Border area***: This area extends the padding area to include the element’s borders. The thickness of the borders is determined by the border-width properties. If there is a background-color or background-image set on the box, it extends to the outer edge of the border.

***Margin area***: This area extends the border area to include an empty area called the margin area. This is used to separate an element from its neighbors. The size of the margin area is determined by the margin-top, margin-right, margin-bottom, margin-left properties. When margin collapsing occurs, the margin area is not clearly defined since margins are shared between boxes.

The code below will show an example of how properties are used in HTML

*<HTML>*

*<HEAD>*

*<TITLE>Examples of margins, padding, and borders</TITLE>*

*<STYLE type="text/css">*

*UL {*

*background: yellow;*

*margin: 12px 12px 12px 12px;*

*padding: 3px 3px 3px 3px;*

*/\* No borders set \*/*

*}*

*LI {*

*color: white; /\* text color is white \*/*

*background: blue; /\* Content, padding will be blue \*/*

*margin: 12px 12px 12px 12px;*

*padding: 12px 0px 12px 12px; /\* Note 0px padding right \*/*

*list-style: none /\* no glyphs before a list item \*/*

*/\* No borders set \*/*

*}*

*LI.withborder {*

*border-style: dashed;*

*border-width: medium; /\* sets border width on all sides \*/*

*border-color: lime;*

*}*

*</STYLE>*

*</HEAD>*

*<BODY>*

*<UL>*

*<LI>First element of list*

*<LI class="withborder">Second element of list is*

*a bit longer to illustrate wrapping.*

*</UL>*

*</BODY>*

*</HTML>*

Q5

The following will illustrate how a browser loads and bootstraps a rich web app from an initial URL:

1. User navigates to a URL in the web browser
2. Browser makes a TCP/IP connection to server IP
3. Application server accepts the connection from the browser
4. Browser sends a HTTP requests over the TCP/IP connection
5. Application server parses the request and responds to the browser over the same open TCP

Loading the web app:

1. The first HTTP request will load the initial assets from the server (HTML, JavaScript & CSS).
2. The browser parses each in turn and runs any commands.
3. For example, HTML elements would build the view of the websites and the CSS commands would style the page.
4. Some commands may cause the browser to connect back to application server to other servers to load more assets.
5. This process continues until the last asset has been fetched.