Full Stack Development

UNIT -II

Redux: Application Architecture, Integrating Redux with React.

- **Testing & Deployment of React Application.**
- **Node.js:** Using Events, Listeners, Timers, and Callbacks in Node.js, Handling Data I/O in Node.js. Accessing the File System from Node.js, Implementing HTTP Services in Node.js.
-

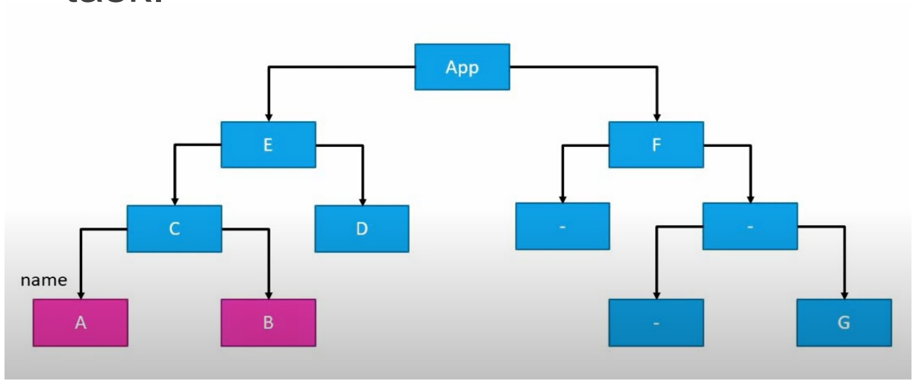
Redux
Application architecture

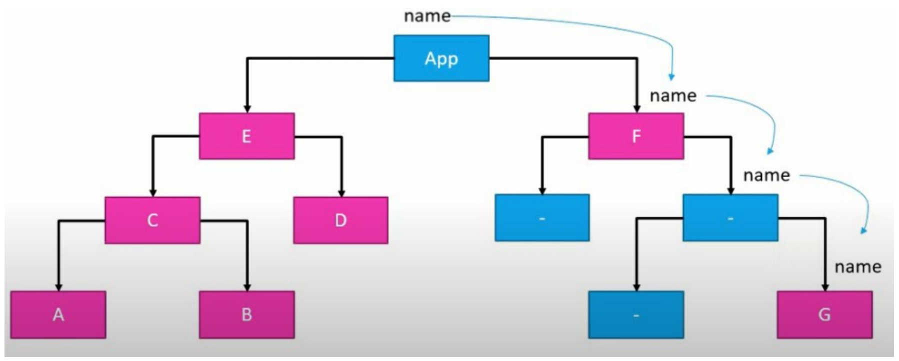
Redux

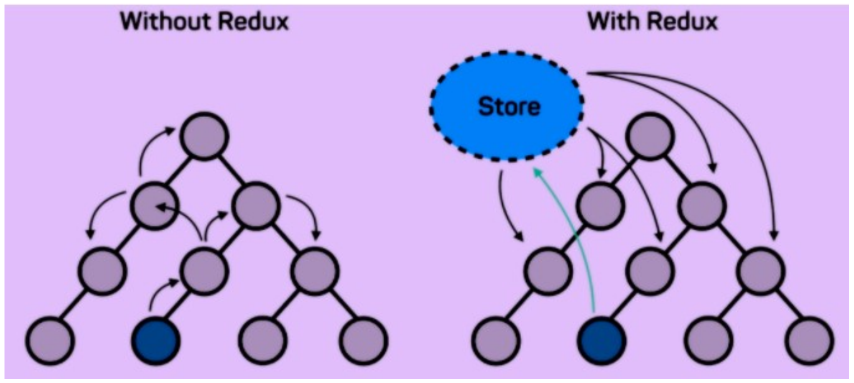
- Redux is a predictable state container for Java Script Apps (React).
- Redux is not tied to React
- Redux can be used with React, Angular, Vue or even Vanilla Java Script
- Redux is a **state management library** for JavaScript applications
- React redux chips in and helps to maintain state at the application level. React redux allows any component to access the state at any time. Also, it allows any component to change the state of the application at any time.
- Redux stores state of your application
- State of a component is:
  - in the case of ‘LoginForm’ component, username, password, submitYesNo fields
  - In the case of ‘Users’ component, usernames
- Redux basically stores and manages application state (**All components data + Application Logic**)
- React component can get the latest state from the store as well as change the state at any time. Redux provides a simple process to get and set the current state of the application
- Components in React will have their own state. Why we need to use Redux to maintain the state?
- react-redux is the official Redux UI binding library for React application

Redux

- Assume that Component A is getting username and maintaining it in its state. If another component ex. B needs to display the 'username' , state of A (name) needs to be lifted up and sends to component B.
- If G needs to display username, state needs to be transferred through unnecessary nodes and also updates needs to be reflected in several nodes. This is a tedious task.







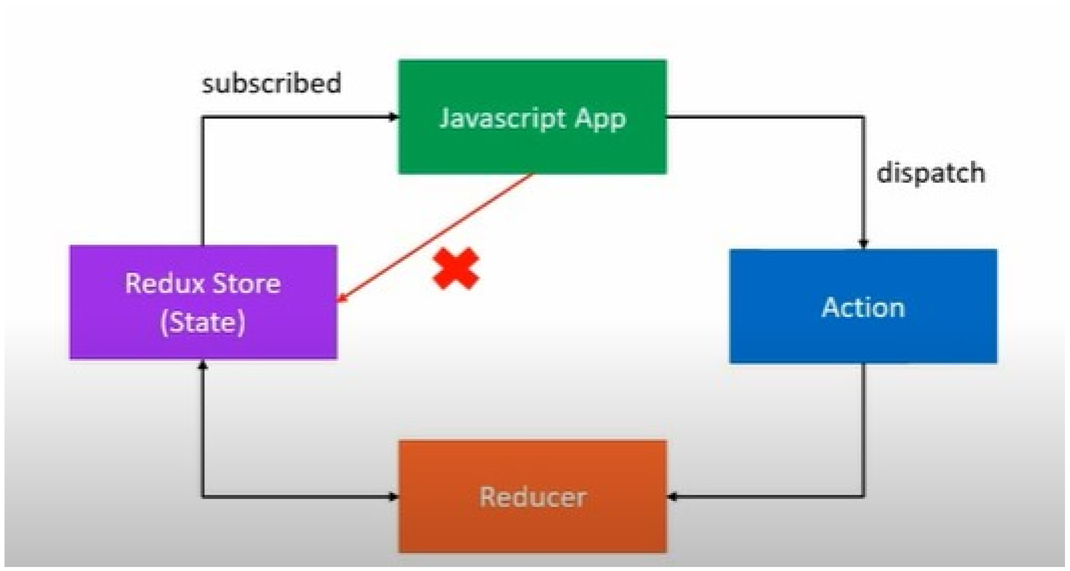
Redux

- To avoid the above problem, state is not stored with the components but they are maintained in a container accessible to all the components.
- **Store:** Holds state of the Application
- **Action:** Describes the changes in the state of the application
- **Reducer:** Actually carries out the state transition depending on the Action.

Three Principles:

1. The state of the whole application is stored in an object tree within a single object store.
2. The only way to change the state is to emit an action, an object describing what happened.
3. To specify how state tree is transformed by actions, you write pure reducers.

   Reducer – (previousState, action) => newstate



Redux – Pros & Cons

Pros:

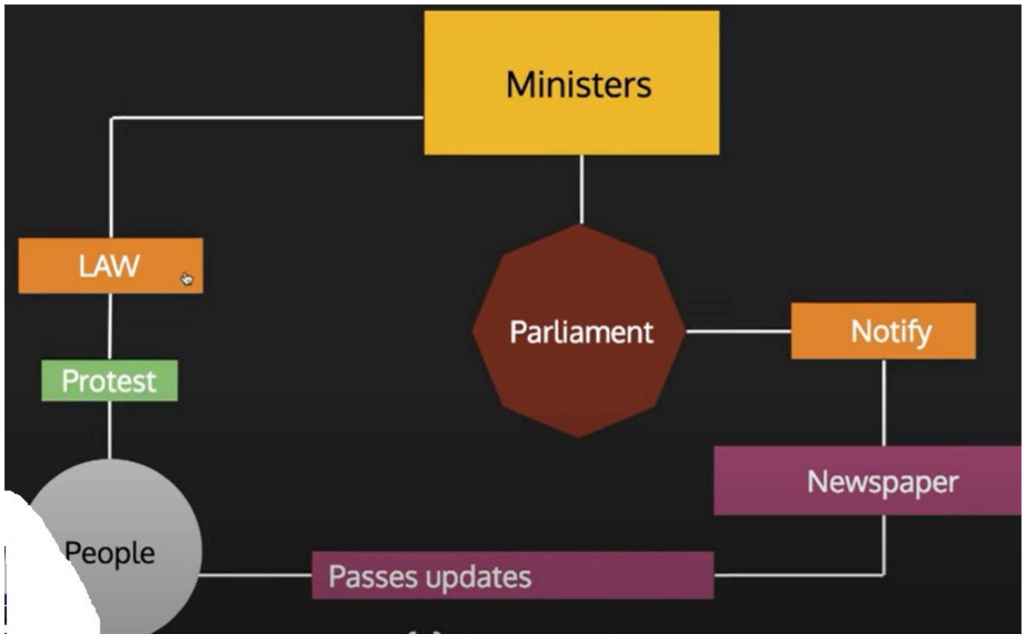
1. There is always **one source** of truth, the store. Thus, there is no confusion about how to sync the current state with actions and other parts of the application.
2. The code is **easier to maintain** because it has a predictable outcome and strict structure.
3. Redux makes **coding more consistent** due to more strict code organization procedures.
4. It's very useful, especially during the initial render, making for better user experience and **search engine optimization**.
5. Developers can **track** everything going on in the app in real-time—from actions to state changes.

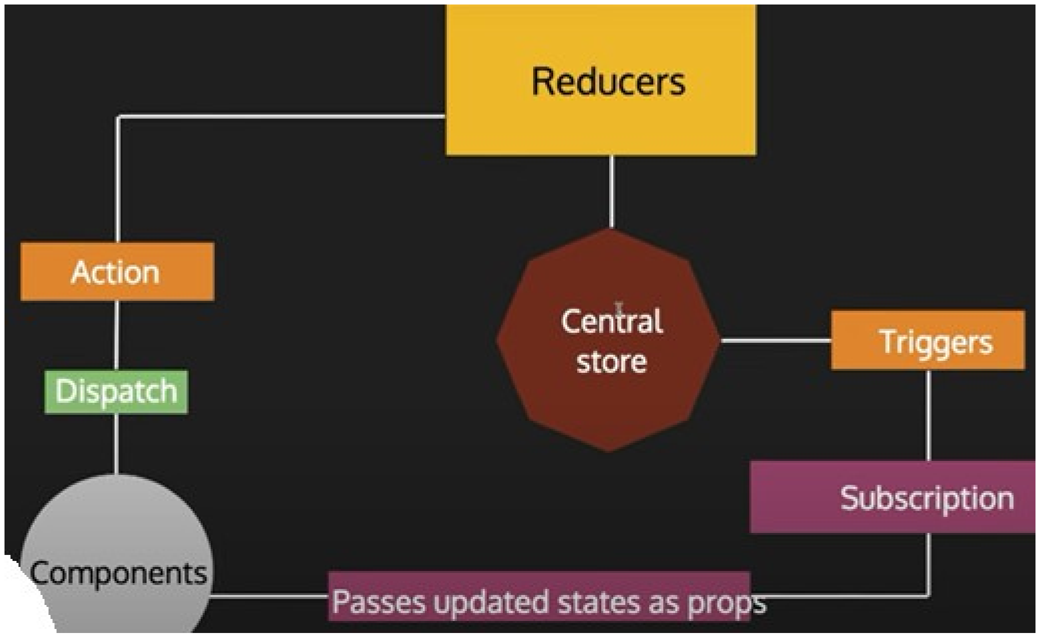
Cons:

1. Since it has no encapsulation, **any component can access data**, which may potentially cause security issues.
2. Some parts of the code are just boilerplate. However, these parts have to be incorporated with no alteration, and this **restricts the design**.
3. As the **state is immutable in Redux**, the reducer updates the state by returning a new state every time which can cause excessive use of memory.

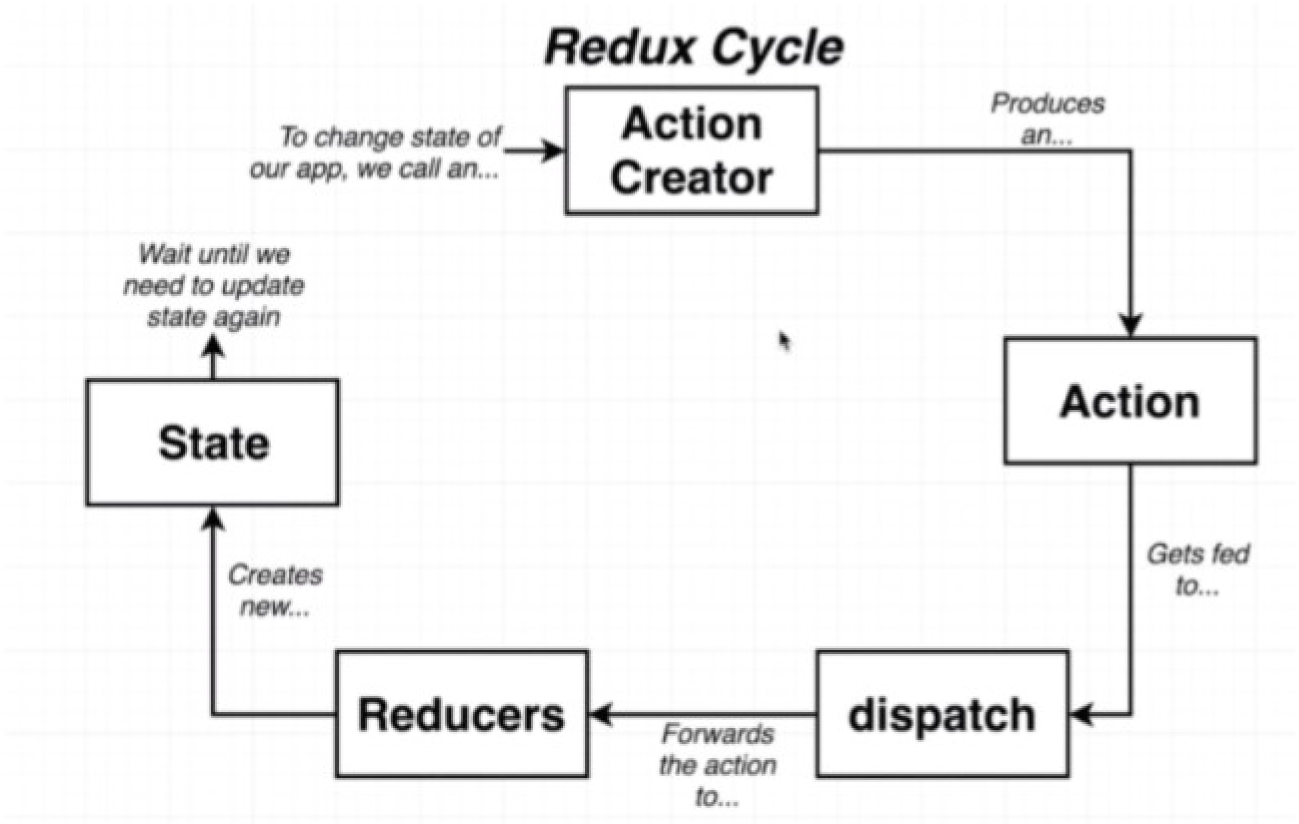
Redux

-

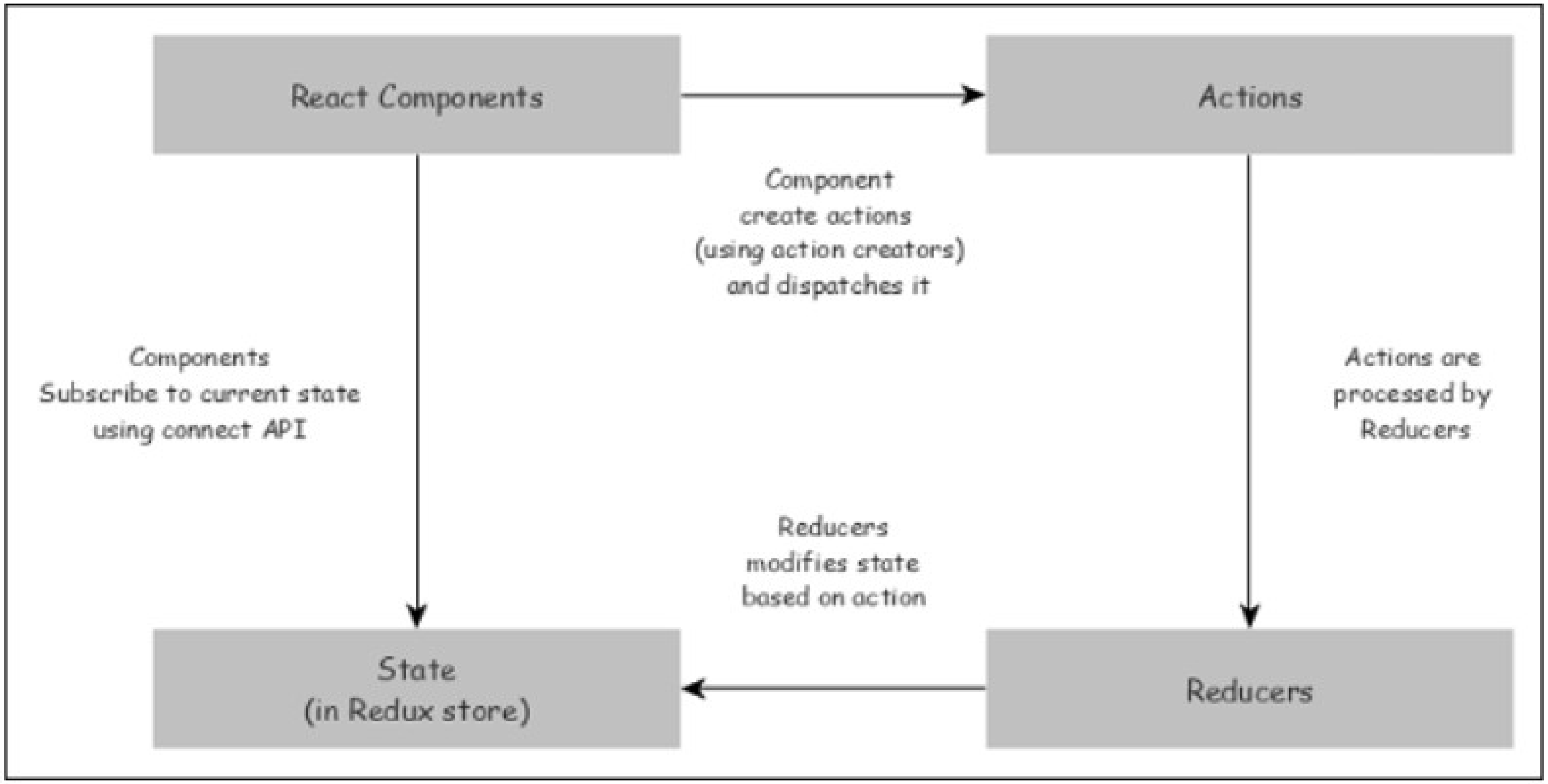

Ministers
LAW
Parliament
Notify
Protest
Newspaper
People
Passes updates


Reducers
Action
Central store
Triggers
Dispatch
Subscription
Components
Passes updated states as props

Redux


Redux Cycle
To change state of our app, we call an...
Action Creator
Produces an...
Action
Gets fed to...
dispatch
Forwards the action to...
Reducers
Creates new...
State
Wait until we need to update state again

Redux – work flow


React Components
Actions
Component create actions (using action creators) and dispatches it
Components Subscribe to current state using connect API
Actions are processed by Reducers
Reducers modifies state based on action
State (in Redux store)
Reducers

- **React component** subscribes to the store and get the latest state during initialization of the application.
- To change the state, React component creates necessary **action** and dispatches the action.
- **Reducer** creates a new state based on the action and returns it. Store updates itself with the new state.
- Once the state changes, **store** sends the updated state to all its subscribed component.

Redux

- JavaScript application is subscribed to the Store. It will not directly update the state.
- TO update the state, Application dispatches an Action.
- Reducer handles the Action and updates the state.

Action

- An object setting up information about our data moving into state.

Dispatch

- Functions that act as a bridge between our actions and reducers, sending the information over to our application.

Reducer

- A function that receives data information from our action in the form of a type and payload and modifies the state based on additional provided data.

State

- Where all the data from our reducers are passed into a central source.

Redux

When working with Redux, you will need **three main things:**

- actions: these are objects that should have two properties, one describing the **type** of action, and one describing what should be changed in the app state.
- reducers: these are functions that implement the behavior of the actions. They change the state of the app, based on the action description and the state change description.
- store: it brings the actions and reducers together, holding and changing the state for the whole app — there is only one store.

Redux - Actions

- The only way your application can interact with the store
- Carry some information from your app to the redux store
- Plain JavaScript object
- Have a **‘type’ property** is a string constant that indicates the type of action being performed

In index.js:

```
//import redux from ‘redux’;
const redux = require(‘redux’)
const createStore = redux.createStore
```

```
const BUY_CAKE = ‘BUY_CAKE’
function buyCake( ){
      return
      {
            type: BUY_CAKE,
            Info: ‘First Redux Action
      }
   }
```

Redux - Reducers

- Specify how the app's state changes in response to actions sent to the store.
- Function that accepts state and action as arguments and returns the next state of the application
  - **(prviousState, action) => newState**

```
Const initialState = {
            numOfcakes: 10
}
Const reducer = (state=initialState,action) => {

Switch(action.type){
Case BUY_CAKE: return{
                    …state,
                     numOfcakes:state.numOfcakes - 1
                    }
          default: return state
}
}
```

… is a **Spread operator** which creates a copy of the state object.

Redux - Store

- For entire store, one state
- Holds application state
- Allows access to state via getState()
- Allows state to be updated via dispatch(action)
- Registers listeners via subscribe(listener)
- Handles unregistering of listeners via the function returned by the subscribe(listener)

Installation:

```
npm install --save redux
npm install --save react-redux
npm install --save-dev redux-devtools
```

OR

```
npm install redux react-redux –save
```

- CTRL+SHIFT+P → type ‘Settings’

Redux Application – step by step

1. create react app using:

```
npx create-react-app redux-demo
```

2 install redux library:

```
npm install redux react-redux --save
```

3. Create a Redux Store:

```
import { createStore } from 'redux';
```

Redux Application – step by step

createStore( ) function:

- The createStore function accepts three arguments:
- the **first argument** is a function that is normally known as a reducer (required)
- the **second argument** is the initial value of the state (optional)
- the **third argument** is an enhancer where we can pass middleware, if any (optional)

Example:

```
import { createStore } from 'redux';
const reducer = (state, action) => {
        console.log('reducer called');
        return state;
};
const store = createStore(reducer, 0);
```

Redux Application – step by step

Example 2:

```
import { createStore } from 'redux';
const reducer = (state = 0, action) => {
    console.log('reducer called');
    return state;
};
const store = createStore(reducer);
```

Subscribe to store:

- using the subscribe function, we're registering a callback function that will be called once the store is changed. And inside the callback function, we're calling the store.getState method to get the current value of the state.

```
store.subscribe(() => {
console.log('current state', store.getState());
});
```

Testing & Deployment of React Applications

Testing

- It’s a fundamental part of developing high quality software solutions.
- Testing in software development is the process of validating assumptions.
- For example, say you’re building an application (like Medium, Ghost, or WordPress) that lets users write and create blog posts. Users pay a monthly fee and get the hosting and the tools to run their own blog. When creating the frontend of the application, there are several key things it *must* do (among others), including correctly displaying those posts and letting users edit them.
- Testing forces you to visit (or revisit) your assumptions about your software. You walk it through the different cases it can handle and ensure that it handles them all appropriately.

Few Types of Testing:

1. Unit Testing
2. Service
3. Integration

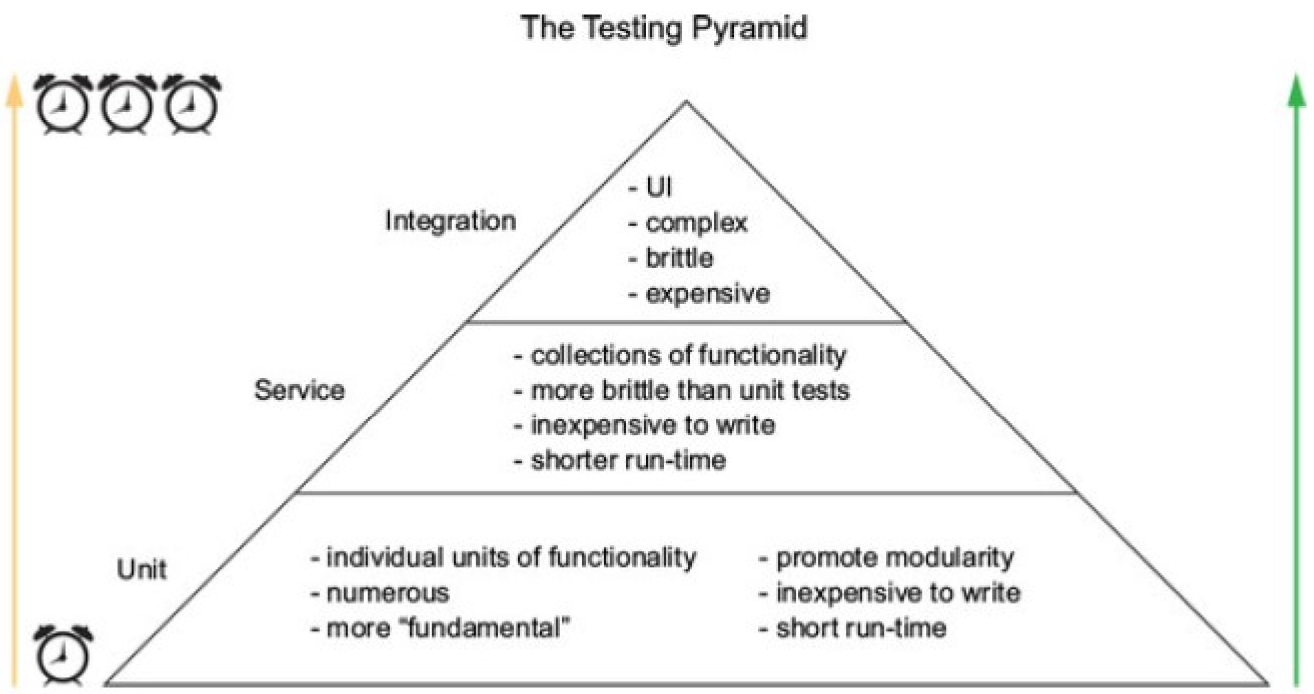
Unit Testing— Unit tests focus on individual units of functionality. For example, say you have a utility method for fetching new posts from the server. A unit test will focus only on that one function.

Service Testing— Service tests focus on bundles of functionality. This part of the “testing spectrum” can include a variety of granularities and focuses. The point, though, is that you’re testing things that aren’t at the highest level (see integration tests, next) or the lowest levels of functionality.

Testing

Integration Testing:

- Integration tests focus on an even higher level of testing: the integration of various parts of an application. They test the way that services and lower level functionality come together. Typically, these tests test an application through its user interface, not through the individual code behind the user interface.



Libraries for React Testing

- *Test runner* → The test runner allows you to run functional tests and export results.
- *Test doubles (Mocks)*
- *Assertion libraries*
- *Environment helpers*
- *Framework specific libraries*
- *Coverage tools*

Test runner:

The test runner allows you to run functional tests and export results.

Test Doubles:

In automated testing it is common to use objects that look and behave like their production equivalents, but are actually simplified. This reduces complexity, allows to verify code independently from the rest of the system and sometimes it is even necessary to execute self validating tests at all. A Test Double is a generic term used for these objects.

Libraries for React Testing

Assertion libraries: You can use JavaScript to make assertions about your code (for example, does *X* equal *Y*?), but there are plenty of edge cases that you'll need to account for. Developers have created solutions to make writing assertions about your code easier. Jest comes with assertion methods built in

***Environment helpers*:**

- You'll be using **Enzyme** and the **React Test Renderer** to aid in testing your React components.
- **Enzyme** makes testing React components easier. It provides a robust API that lets you query for different types of components and HTML elements, set and get props of components, inspect and set component state, and more.
- **The React test renderer** does similar things and can also generate snapshots of your components.

Framework specific libraries— There are libraries specifically made for React (or other frameworks) that make writing tests for a particular framework easier. These abstractions are usually developed to aid in the testing of a library or framework and handle setting up anything needed by the framework

Libraries for React Testing

Coverage tools— we can determine which parts of your code are “covered” by tests.

- 100% code coverage doesn’t mean you can’t have bugs), but it can guide how you test your code.
- You’ll use Jest’s built in coverage tool, which utilizes a popular tool called **Istanbul**

Testing React applications using JEST, ENZYME, AND REACTTEST-RENDERER

Testing using Jest

npm test → *allows to run the test script.*

Deployment of React.js applications

Deployment of React.js applications

- **npm run build** creates a build directory with a production build of your app. Set up your favorite HTTP server so that a visitor to your site is served index.html
- **npm install –g serve** installs 'serve' for creating static server to serve the page requests

  serve -s build

✓ Static server serves the page at port 3000 by default. You can change this using:

serve –s build –l 4000 → serves requests at port no 4000

serve –h → gives list of all options of the 'serve' command.

- You don't necessarily need a static server in order to run a Create React App project in production. It also works well when integrated into an existing server side app.

Fixing error: running scripts is disabled on this system.

1. Open powershell in adminstrator mode
2. Issue command,

   ```
   Set-ExecutionPolicy -ExecutionPolicy RemoteSigned
   ```

3. Give **serve –s build** command from VS Code terminal.

   3.1 use http://localhost:3000 to get Website homepassge

   3.2 even port number can be changed using:

   ```
   serve –s build –l 4000
   ```

Deployment of React.js applications

npm run build → creates production ready build.

Hosting providers for deployment:

Netlify: Offers easy deployment with continuous integration and hosting.

Vercel (formerly Zeit Now): Provides serverless deployment with great performance.

GitHub Pages: Host your React app directly from a GitHub repository.

Heroku: A flexible platform that allows you to deploy various web applications.

AWS (Amazon Web Services): Offers various services for hosting React apps, including S3 for static sites and Amplify for serverless applications.

Firebase: Google's hosting platform, suitable for web applications with authentication and real-time database needs.

Deployment procedure:

Deployment methods may vary depending on your hosting provider, but generally, you will need to:

1) Create an account or log in to your hosting provider.
2) Follow their instructions for deploying a new project.
3) Configure build settings, such as the build folder path.

Deployment of React.js applications

✓ npm install -g server	installation of server package
• Serve –s build	to start appl.

Node.js

Node.js

Node.js:

- ✓ As an asynchronous event-driven JavaScript runtime, Node.js is designed to build fast, scalable network applications.
- ✓ It’s NOT a web framework, and it’s also NOT a language
- ✓ Open source server environment. Runs Java Script
- ✓ Node.js is JavaScript running on the server-side (SSJS -> Server-Side JavaScript)
- ✓ Created by Ryan Dahl in 2009
- Based on Google V8 Engine. Dahl created Node.js **on top of V8 Java Script Engine** as a server-side environment that matched the client-side environment in the browser.
- ✓ +99 000 packages.
- ✓ Server Side technologies include PHP, JSP, ASP.NET, Rails, Django, and Yes, also Node.js
- ✓ Written in Java Script.
- ✓ Use the **same language for client side and server side.** The fact that Node.js is written in JavaScript allows developers to easily navigate back and forth between client and server code and even reuse code between the two environments.
- ✓ Single Thread with Event Loop
- ✓ Runs on Windows, Linux, Mac
- ✓ Easy to implement and Highly Scalable
- ✓ Active community
- ✓ Node.js environment is clean and easy to install, configure, and deploy
- ✓ Can handle thousands of Concurrent connections with Minimal overhead (cpu/memory) on a single process
- ✓ Non Blocking I/O. Uses Asynchronous programming

Node.js

- Literally in only an hour or two you can have a Node.js webserver up and running.
- Node.js quickly gained popularity among a wide variety of companies. These companies use Node.js first and foremost for scalability but also for ease of maintenance and faster development.
  - **Ex:** LinkedIn, eBay, Yahoo, Microsoft, NewYork Times

Usage:

- To handle HTTP traffic, the most common use is **as a webserver**.
- However, Node.js can also be used for a variety of other **web services** such as:
  - ✓ **Web services APIs such as REST**
  - ✓ Real-time **multiplayer games**
  - ✓ **Backend web services** such as cross-domain, server-side requests
  - ✓ **Web-based applications**
  - ✓ Multiclient communication such as IM (Instant Messaging)

What can node.js do?

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database
- Provides built in modules or user created modules

Node.js modules built in

Node.js modules: Node.js comes with many built-in modules available right out of the box.

- ✓ **Assertion testing:** Allows you to test functionality within your code.
- ✓ **Buffer:** Enables interaction with TCP streams and file system operations.
- ✓ **C/C++ add-ons:** Allows for C or C++ code to be used just like any other Node.js module.
- ✓ **Child processes:** Allows you to create child processes.
- ✓ **Cluster:** Enables the use of multicore systems.
- ✓ **Command line options:** Gives you Node.js commands to use from a terminal.
- ✓ **Console:** Gives the user a debugging console.
- ✓ **Crypto:** Allows for the creation of custom encryption.
- ✓ **Debugger:** Allows debugging of a Node.js file.
- ✓ **DNS:** Allows connections to DNS servers.
- ✓ **Errors:** Allows for the handling of errors.
- ✓ **Events:** Enables the handling of asynchronous events.

Node.js modules built in

- ✓ **File system:** Allows for file I/O with both synchronous and asynchronous methods.
- ✓ **Globals**: Makes frequently used modules available without having to include them first.
- ✓ **HTTP:** Enables support for many HTTP features
- ✓ **HTTPS:** Enables HTTP over the TLS/SSL
- ✓ **Modules:** Provides the module loading system for Node.js.
- ✓ **Network Appl.:** Allows the creation of servers and clients.
- ✓ **OS:** Allows access to the operating system that Node.js is running on.
- ✓ **Path:** Enables access to file and directory paths.
- ✓ **Process:** Provides information and allows control over the current Node.js
- ✓ process.
- ✓ **Query strings:** Allows for parsing and formatting URL queries.
- ✓ **Readline:** Enables an interface to read from a data stream.
- ✓ **REPL (Read-Evaluate-Print-Loop):** Allows developers to create a command shell. This is used to test, evaluate, experiment, or debug code much easier and accessible way.
- ✓ **Stream:** Provides an API to build objects with the stream interface.

Node.js modules built in

- ✓ **String decoder:** Provides an API to decode buffer objects into strings.
- ✓ **Timers:** Allows for scheduling functions to be called in the future.
- ✓ **TLS/SSL:** Implements TLS and SSL protocols.
- ✓ **URL:** Enables URL resolution and parsing.
- ✓ **Utilities:** Provides support for various apps and modules.
- ✓ **V8:** Exposes APIs for the Node.js version of V8.
- ✓ **VM:** Allows for a V8 virtual machine to run and compile code.
- ✓ **ZLIB:** Enables compression using Gzip and Deflate/Inflate

Browsers with JS Engines (VM)

Google Chrome – V8 & Blink

Mozilla Firefox – SpiderMonkey

Safari **--** Webkit & Nitro

Edge **--** Chakra & EdgeHTML

Opera **--** V8 & Blink

Internet Explorer **--** Chakra & Trident

Node.js installation folder files

Path: C:\Program Files\nodejs

Files in the installation folder:

node: This file starts a Node.js JavaScript VM. If you pass in a JavaScript file location, Node.js executes that script. If no target JavaScript file is specified, then a script prompt is shown that allows you to execute JavaScript code directly from the console.

npm: This command is used to manage the Node.js packages.

node_modules: This folder contains the installed Node.js packages. These packages act as libraries that extend the capabilities of Node.js.

Check Installation:

From command prompt, enter command: **npm --version**

To start Node VM, at command prompt run: node → opens Node VM command prompt

console.log("Hello World"); →display 'Hello World' message in console screen .

Check npm installation:

npm --version

Running Node.js application

Step 1: Create a folder with the App name

Step 2: From cmd prompt, go to the project folder location, and issue command:

```
npm init                    → creates 'package.json' file
```

Step 3: Add **app.js** file and add the following code:

```
var msg = 'Hello World';
console.log(msg);
```

Step 4: Open 'New Terminal' from the VS code.

Step 5: **To run the project**, from Terminal Prompt, give command:

```
node app.js
```

node.exe → This file starts a Node.js JavaScript VM

Npm →used for managing the Node.js packages

Node.js Webserver

```
const http = require('http');       → includes ‘http’ module in the code

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

http module

http createServer(req, res) →

- **options** <*Object*>**:** It is any <Object> or JSON data which contains connection details.
- **Callback** <*Function*>**:** It is a callback<function> which receives the created socket.

Listen(port_no, host name, handler) →

Parameters: options → portno, hostname etc..; handler → callback function

IDEs for node.js

- ✓Eclipse
- ✓Webstorm IDE
- ✓Visual Studio Code