How React Works Behind the Scenes

# Component vs Instance vs Element

A screen shot of a computer

Description automatically generated

* Component is the actual function where we define what the component does
  + Function Component(){}
* Component Instance is when the component is Called into the App.js or another Component
  + <Component />
* React Element is what the component Instance returns
* DOM Element(HTML) is the actual representation of the React Element into the DOM

# React Rendering – Overview

A diagram of a component

Description automatically generated

* **Rendering is Triggered** by updating the State somewhere in the application
* **Render Phase**
  + React calls component function and figures out how DOM should be Updated
* **Commit Phase**
  + React actually writes to the DOM, updating, inserting, and deleting elements
* **Browser Paint**
  + The visual change happens

## Triggering the Render

* The render is triggered in 2 situations:
  + Initial Render of the application
  + State update happening in one or more component instances
    - The render process is triggered for the entire application but not the ENTIRE DOM is UPDATED
    - Renders are not triggered IMMEDIATELY, but SCHEDULED for when the JS engine has some “free time”.
    - It’s also batching multiple setState calls in event handlers

## Render Phase

* React Renders (calls the component instances)
* The React Elements are created
* All React Elements make up the **VIRTUAL DOM**
* The Render Phase will spit out the **LIST OF DOM UPDATES**

### A diagram of a tree Description automatically generatedVirtual DOM

* It’s a tree of all React Elements created from all instances in the component tree
* Cheap and Fast to create multiple trees
* It’s a simple JS object
* A Re-Render is triggered
* The component is Re Rendered
* The **Virtual Dom** is Updated and will force all the children components of the RERENDERED component to be RERENDERED

A diagram of a tree

Description automatically generated

* The Virtual Dom will be **RECONCILIED + Diffing**  with the current Fiber Tree
  + The reconciliation is made by the Reconcilier named **FIBER**
* An Updated Fiber Tree is created

### Reconciliation

* React uses as much of the existing DOM as possible
* Updating the whole DOM is very EXPENSIVE and SLOW
* React uses **Reconciliation** 
  + To decide which DOM elements actually need to be
    - Inserted
    - Deleted
    - Updated
* **Reconciliation allow us to never touch the DOM directly.**

### Fiber Reconcilier

* Take the Virtual DOM and build another Tree – the **Fiber Tree**

A diagram of a computer

Description automatically generated

**Reconciliation Steps – see ss below**

1. State is updated
2. App is ReRendered 🡺 All it’s children are Rerendered
3. New React Elements are created for each component instance
4. A new Virtual DOM is created
5. The New Virtual DOM is compared with the Current Fiber Tree using **Reconciliation and Diffing**
6. The Fiber DOM is updated with the necessary changes
   1. A diagram of a computer program

      Description automatically generatedBtn 🡪 DOM Update (text change)
   2. Modal and it’s Childrens 🡪 DOM Deletions (will disappear from screen)
   3. Video Component is Rerendered but it’s not changing so it’s not Affected

## Commit Phase

* Is where React will update the DOM
  + Insert
  + Delete
  + Update (list of DOM updates)
* All COMIT PHASE is **Synchronous**
  + It happens in ONE GO
  + It cannot be interrupted
  + This MUST HAPPEN so that the DOM is not showing partial results
* After Commit Phase, the **WorkInProgress Fiber Tree** becomes the current tree and it’s ready for the next Cycle
* Is handled by the **ReactDOM library for Browsers**
* For mobile, we are using **React Native**
  + React Native still uses **REACT** but it will update and commit for IOS and Android
* For Videos, we are using **REMOTION**
* There are many **RENDERERS**
  + Word, PDF, Figma Designs and so on
  + A diagram of a process

    Description automatically generated

## Browser Paint

* Will update the UI on Screen
  + It’s very browser dependent and it’s not what React is handling

# RECAP

A diagram of a process

Description automatically generated

# Diffing

## How Diffing Works?

* Is based on 2 assumptions
  + Two elements of different types will produce different trees
  + Elements with a **stable key** stay the same across renders

## Diffing Rules

1. Same Component in the Same position
   1. The state is Preserved across Renders

## The Key Prop

* A way to tell the **DIFFING** algorithm that an element is **UNIQUE**
* This will allow React to difstinguish between multiple Instances of the same Component type
* When a key **stays the same across renders,**  the element will be kept in the DOM
  + **Using key in Lists**
* When a key **changes between renders, the** element will be destroyed and a new one will be created (event if the position in the tree is the same as before)
  + **Using key to reset state**

A computer screen shot of a code

Description automatically generated

* We have added the Key={} prop with an **UNIQUE value** to the TabContent Component so that React will know that each of the Instances are a different component, so we use the **Key** to **RESET THE STATE**

# Render Logic

* Code that lives at the top level of the component function
* Participates in describing how the Component view looks like
* Ex
  + The states
  + Any JSX that is inside a return function

### Components must be PURE when it comes to RENDER LOGIC:

* Given the same props (input), a component should always return the same JSX(output)

### Render Logic is not allowed to produce any side effects

* + No interaction with the Outside world is allowed
  + No network requests
  + No start timers
  + No Use directly the DOM API
  + No mutate objects or variables outside of the function scope
  + No STATE UPDATE 🡪 infinite loops

# Event handler Logic

* Executes as a **CONSEQUENCE** of the event that the handler is listening for

A screenshot of a computer

Description automatically generated

# State Update Batching

* React batches multiple STATE UPDATES into 1 RENDER + COMMIT
* A diagram of a computer program

  Description automatically generated
* It’s a Performance Optimization
* Updating the state in React is **ASYNCHRONOUS**

# Events in React

* React does not attach the events to any of the specific elements
* React actually attaches all events to the **ROOT ElementA screenshot of a computer

  Description automatically generated**
* React performs event Delegation behind the scenes.

## Synthetic Events

* The ‘e’ is a wrapper around the DOM’s native event object
* Has the same interface
  + stopPropagation()
  + preventDefault()
* Fixes some browser inconsistencies 🡪 all events works the same ACROSS ALL BROWSERS

# Libraries vs Frameworks

## Framework (Angular, Vue, Swelte)

* All in one kit
* A complete structure that includes everything that we need to build a complete application
* Angular has everything inside
  + Routing
  + Styling
  + HTTP Requests and so on

### Library (React – view library)

* Separate ingredients
* To build an APP, we will need to add multiple Libraries for Routing, Styling, HTTP Requests and so on
* React is only the **VIEW** library
* Incredible Freedom
  + Can choose what apps you want to use
* You need to be able to use and find all those other libraries

# React Ecosystem

* Since React is so popular, there is a HUGE Ecosystem of 3rd party apps that we can use

### Routing (for SPA’s)

* React Router, React Location

### HTTP Requests

* JS Fetch(), AXIOS

### Remote management

* React Query, SWR, Apollo

### Global state management

* Context API, Redux, Zustand

### Styling

* CSS Modules, Tailwind Css

### Form Management

* React Hook Form, Formik

### Animations

* Motion, React-Spring

### UI Components

* Chakra, Mantine