Day 7: Frontend Development

JS and React

Fundamental of web apps

- Dev tools (console and network)
- HTML and DOM
  - concept of document, window
- JS Refresher
  - ECMAScript
  - variables
  - modules (import and export)
  - Babel
  - modern JS (ES6)

Single Page Application (SPA)

- application that loads a single HTML file and necessary assets (js/css) required for application to run
- instead of the default method of a web browser loading entire new pages, SPA rewrites current web page with new data from the web server
- Main objective: to provide native app experience to user by dynamically updating content of the page from server without loading the page
- Examples: React, Vue, Angular, Svelte, etc.

Page loading in Traditional app vs SPA

Traditional

Single Page Application

Pros

- single html file to load reducing server load
- Smooth user experience akin to native app
- Fast and responsive frontend development due to de-coupled structure of frontend and backend

Cons

- SEO
- Initial loading time

Introduction to ReactJs

- React is an open-source frontend JS library for building user interfaces with a component-based architectural approach

Project initialization and structure

- Nodejs
- NPM
- Build tools (webpack, vite, cra)

Elements

- smallest building blocks in react apps
- react use JSX which gets converted to React.createElement() function calls that evaluates to JS objects so that browser can understand
- JSX to JS object transformation is done by library like BabelJS

JSX (Javascript XML)

- JSX let you write html-like markup and JS together

Components

- contain react elements
- reusable UI elements of the react app that generally returns JSX
- returns only one root element
- to avoid extra div generated in the DOM, React.Fragment can be used

React rendering process

- Rendering is React’s process of describing a user interface based on the application’s current state and props

  Rendering can be divided into two phases:

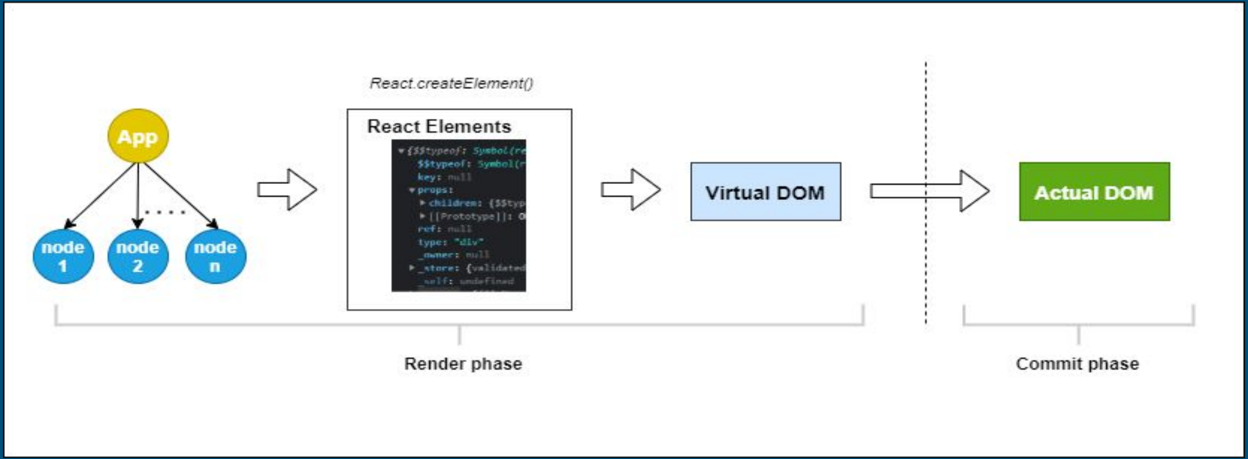
1. Render phase (VDOM render)
   - every time set function triggered, new VDOM is created
   - reconciliation happens to know what is the minimum change that need to be updated
2. Commit phase(Native DOM render)

- here the minimum change that need to be updated is reflected back in the real/actual/native DOM

DOM vs VDOM

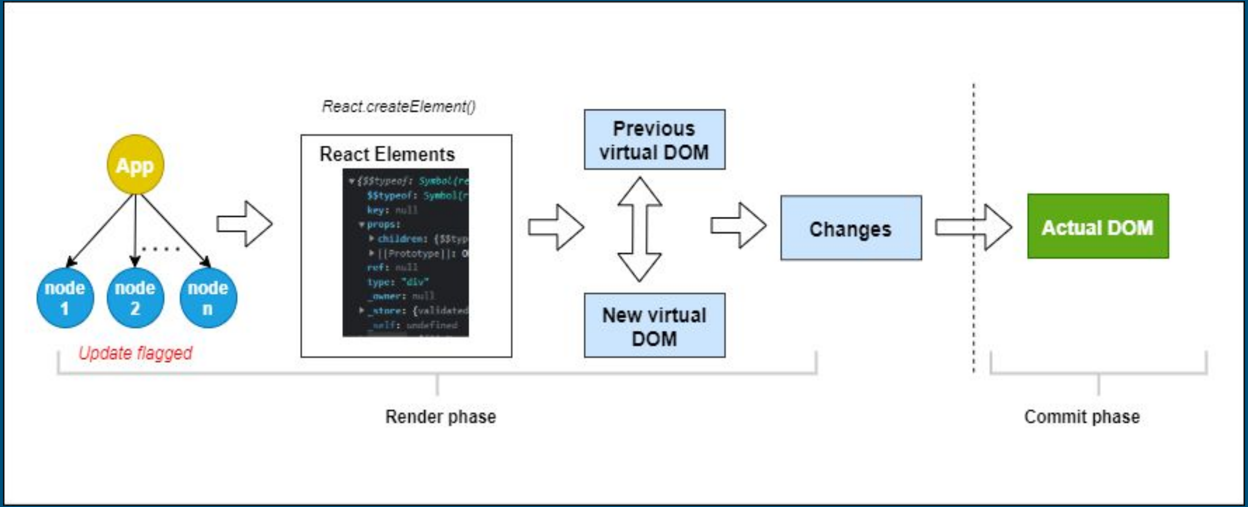
- DOM: tree structure made up of HTML markup
- VDOM: tree structure made up of React (javascript) element.

App initial render



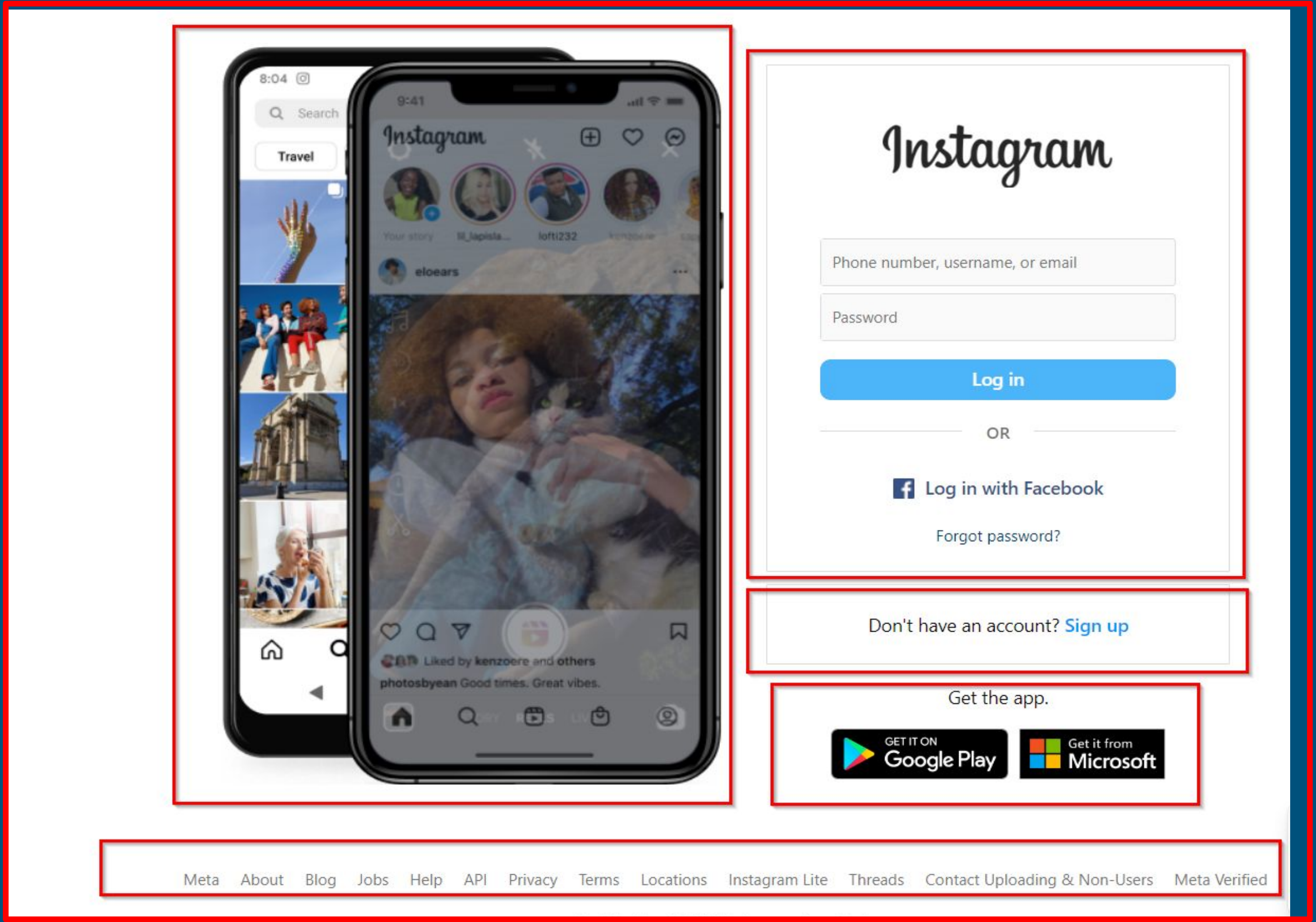
```
▾{$$typeof: Symbol(re
  $$typeof: Symbol(r
  key: null
 ▾props:
  ▸children: {$$typ
  ▸[[Prototype]]: O
  ref: null
  type: "div"
  _owner: null
 ▸_store: {validated
  _self: undefined
```

App re-render



```
▾{$$typeof: Symbol(re
  $$typeof: Symbol(r
  key: null
 ▾props:
  ▸children: {$$typ
  ▸[[Prototype]]: O
  ref: null
  type: "div"
  _owner: null
 ▸_store: {validated
  _self: undefined
```

Components



Instagram

Phone number, username, or email

Password

Log in

OR

Log in with Facebook

Forgot password?

Don't have an account? Sign up

Get the app.

GET IT ON Google Play

Get it from Microsoft

Meta About Blog Jobs Help API Privacy Terms Locations Instagram Lite Threads Contact Uploading & Non-Users Meta Verified

Props

- Read-only properties that are shared between components.
- A parent component can send data to a child component.
- <Component key=value/>

Hooks

- Special function that allows functional components to use react features
- React (v16.8)
- Not used in class components
- (useState, useEffect, useContext, useRef etc)

useState()

- A react hook that create a stateful variable and a setter function to update its value in the virtual dom.
  Like [name, setName]

useEffect()

- Performs side effects in functional components such as fetch api data.
- Tells to do something when components rerenders, when state of value changes or props changes.
- useEffect(function, [dependencies])
- useEffect(()=>{ })=> runs after every re-render
- useEffect(()=>{ }, [ ])=> runs once when mounts
- useEffect(()=>{}, [value])=> runs on mounts + when value changes