Constructor -> initialize state

Statjc getDerivedStateFromProps(prevState, nextProps) -> returns new state or null (static bcs user will not have access to this keyword meaning he cannot execute this func and it is only owned by Class itself)

Render() -> mandatory it draws jsx

ComponentDidMount -> DOM is ready now and we can do some changes or async calls

Re-render – Lifecycles

Static getDerivedStateFromProps()

shouldComponentUpdate(nextProps, nextState) -> returns true or false. Sometimes re-render does not affect overall picture of component in this cases we can avoid re-rendering using shouldComponentUpdate

getSnapshotBeforeUpdate(prevProps, prevState) ->

componentDidUpdate() -> same with componentDidMount

componentWillUnmount() -> just before component will be destroyed from memory

Stopping from re-rendering React.PureComponent, React.Memo (for functional components)

Error Boundary Component -> 2 lifecycle methods **static getDerivedStateFromError**, **componentDidCatch**.

React Patterns -> Context-api pattern – access props in child components without too much nesting via Provider and Consumer

Render Props -> using children Components as functions inside of class Component (just rendering functions ex. When you map an array you use a render props func)

Presentation Component -> component which is used only for displaying “stateless”

ARROW FUNCTIONS USAGE

Why arrow functions are used in Classes, each function has it is own this keyword which belongs to its scope, when use regular funcs in Class we do not have access to this.setState() the solution is to use binding of this keyword or we can use arrow functions, they do not have this keyword and they take this keyword of immediate block outside of their scope.

Code Splitting – React.Lazyload

React introduced Lazyloading the main idea is that now it is possible to separate components into separate bundlers. Instead of loading one single huge bundle of js we can separate them to different bundlers, by this we can reduce loading of components for a client. Client will only load components which he or she needs. It is achieved by <Suspense></Suspense> and React.Lazyload

Reduxt Thunk – Redux Saga

Saga uses generator functions:

function\* gen() {yield doFirst(); doSecond();}

Javascript Prototype

All objects have prototype which can be variable or method, when we equal new created object to anther object. New object will use properties and methods of inherited object. It works just like inheritance but in implicit way.

Function expressions are not hoisted, but function declarations are hoisted.

Promises are used to simplify async awaits because they get too nested.

Javascript uses lexical scoping, variables outside of block are available inside of it

Closures are functions with preserved data