1. Library itself and small learning curve
2. Everything we build is component where we club different component and build entire page

Header component, footer component, other controls

It’s a component based development.

1. Faster in rendering speed.
2. Strong community comes with huge resources.

Setting up the environment:

IDE: Visual studio code

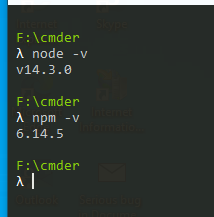
Node js (JavaScript engine on the server)

Npm is package manager helps in installing various dependency’s

Yarn

You can find it in **yarnpkg.org**

**Yarn is package manager used by the react community**



Anything we install globally can be accessed via command line.

Install yarn from npm

$ npm install -g yarn

Learn with app

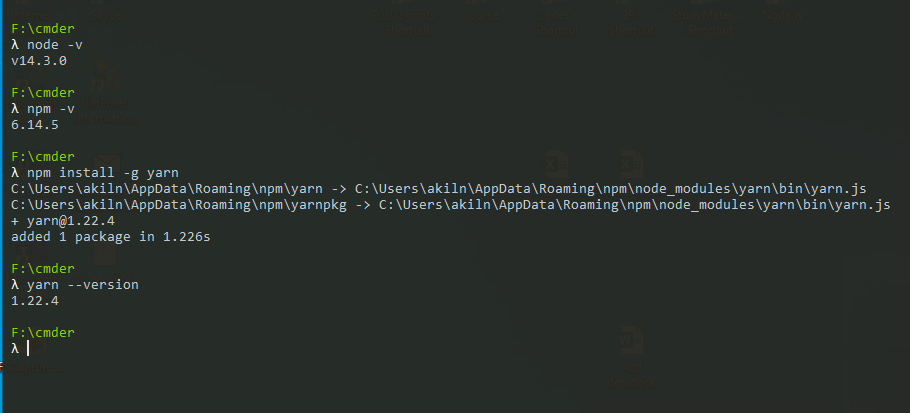
Indecision app:

JSX

Javascript xml: templating syntax that react provides to build and structure our application.

Emmet plugin to support auto complete

Yarn global add live-server



React is a library so to use it we need load react library before using it

1. React library
2. React dom library (companion library)

React vr

React native

First load react then load companion

//CDN1

   <script src="https://unpkg.com/react@15/dist/react.js"></script>

    <script src="https://unpkg.com/react-dom@15/dist/react-dom.js"></script>

//CDN2

    <script src="https://unpkg.com/react@16.0.0/umd/react.development.js" />

    <script src="https://unpkg.com/react-dom@16.0.0/umd/react-dom.development.js"/>

JSX – javascript XML define template and inject data into that template

It’s a language extension for the javascript

1. In react is template based development framework where every component in the page is considered as template

Eg: Header, footer, section, article etc

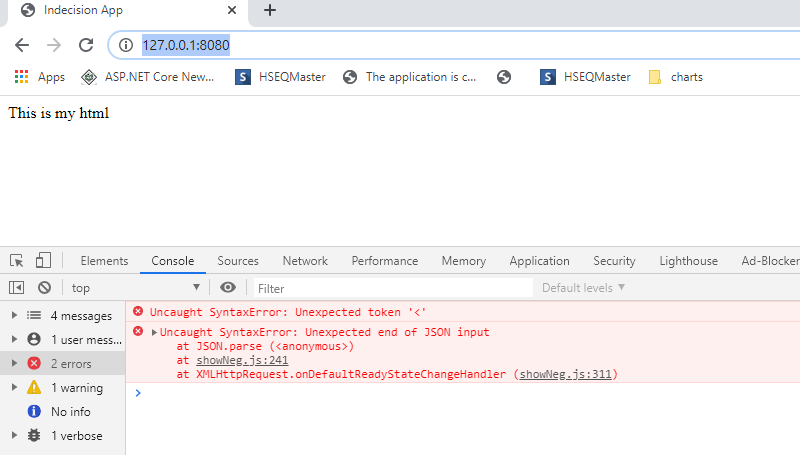
1. Using Jsx we define those template (eg inject data and build the html for it)
2. Then comes the rendering part where in we use the react Dom object to ReactDOM

Use the method render to render that defined template

So in the render method we need to define template and HTM dom element that renders this template.

ReatDOM.render(template, domObject);

OS this is a problem we have with js output because browser is not aware of the extension to JavaScript we need some kind of mediator between application and browser to make browser to understand thing written in the javascript extension



Jsx in javascript extension where all the DOM template is defined.

So we will use Babel.js javascript compiler that compiles code and makes sure the Js code is supported in most of the browsers including the old browser. So wee can develop our js using new cool feature and syntax defined by Es6/E7

That converts all javascript extension to normal javascript files.

Babel basically takes code written in ES6/Es7 and compiles down to ES5

Babel setup:

Local installation: along with babel we need to install set of presets nothing but the plugin to babel which tells what type of JS extension need to be compiled

Standard preset is env which includes Es5,Es6,E7 compilation

React preset jsx copilation preset

Steps:

1. **Babel-cli:** install it globally so that we get command line interface support for all babel command

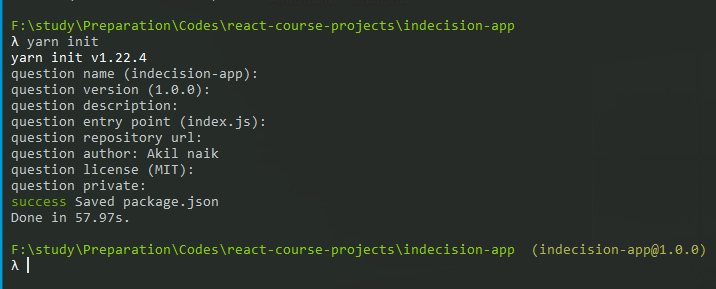
**$ yarn global add** [**babel-cli@6.24.1**](mailto:babel-cli@6.24.1)

**$npm install -g** [**babel-cli@6.24.1**](mailto:babel-cli@6.24.1)

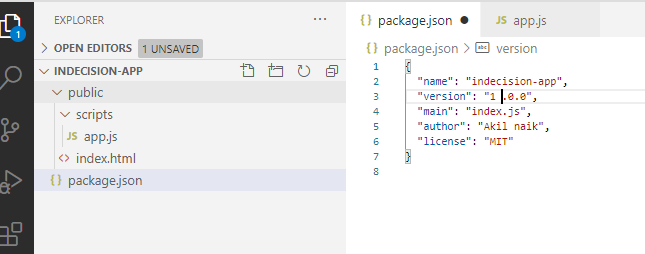
**Test: babel –help**

1. Initialize the project to manage local dependency’s

**$yarn init**



This will be used to manage all packages in application. Adds a file called package.json inside your project

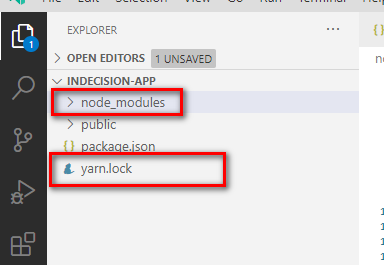


Add’s above file package.json

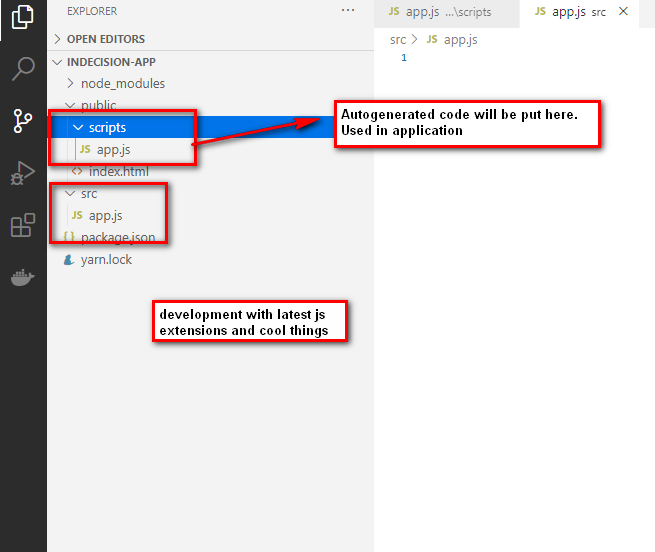
1. Add presets

**$yarn add** [**babel-preset-react@6.24.1**](mailto:babel-preset-react@6.24.1)**,**

**$yarn add** [**babel-preset-env@1.5.2**](mailto:babel-preset-env@1.5.2)



Each node module has local dependency and add its own package file



Run a babel command to generate compiled file along with presets to be used

**$babel src/app.js –out-file=public/scripts/app.js --presets=env,react**

We can setup live watching so that any changes can be automatically updated

**$babel src/app.js --out-file=public/scripts/app.js --presets=env,react –watch**

Yarn command to install all the dependencies or node modules.

**$yarn install.**

**Jsx:**

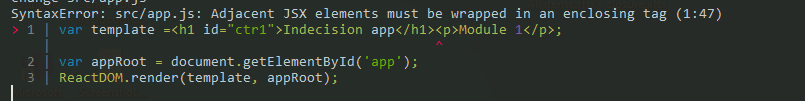
Making template from all the way dynamic, nesting, data from db

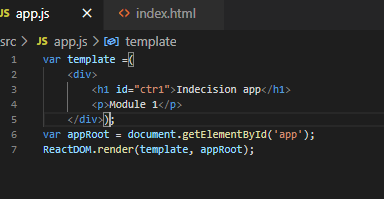
VS Code Add an extension for the better development

Babel ES6/ES7

Path intelisense

1. While making html template it can have only one root element so always have enclosing tag.

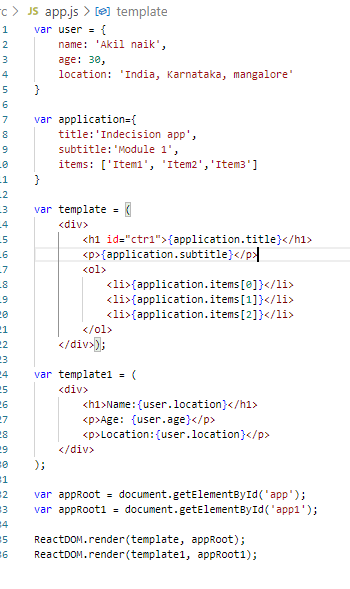




1. In JSX template definition any JS expression should go inside {} any

Object cannot be rendered in the react because react doesn’t have idea about what the object is.

But we can use object properties



Conditional rendering:

1. We cannot have any JavaScript processing logic inside the template definition

Call some javascript method to perform the processing logic.

1. We can write expression inside the {} like logical and ternary as below
2. Some attributes are renamed like for class we use className JSX. Basically one with reserved key words.

ES6:

**Var:**

Can have multiple variable defined with same name

Can access name outside the scope if not found any defines new one at the global scopes

**Let & const:**

Cannot have multiple definition

Only access variable that are defined and which are in the lexical scoping

Bit of defensive programming.

**Arrow function:**

Brand new way of creating function. Expression syntax for the arrow function that eliminates the used of old way of writing function with key function.

1. Arguments object is not bound with arrow function (receive all passed parameter under single name)
2. This key word no more bound to the arrow function so we cannot use arrow function with the class methods. So basically it wont bind the ownness of the object

Any child implementation can receives this of the parent

**Event binding:**

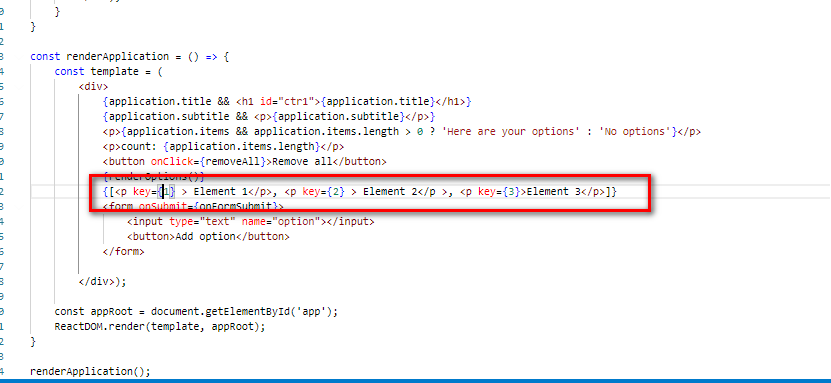
Methods are camel cased. Defined separately and then mapped to the events.

Virtual DOM algorithm run on the back round to decide what changes in the page and retenders only those components

Arrays in JSX:

We can directly render the array as part of the JSX

Also we can put JSX inside another JSX.



Kind of list item we can put it inside the array and directly render that array on the screen.

**Generating random number in JS:**

Use Math.Random() it generates value between 0 - 0.999999

Multiply this value with any number between which you want to generate value

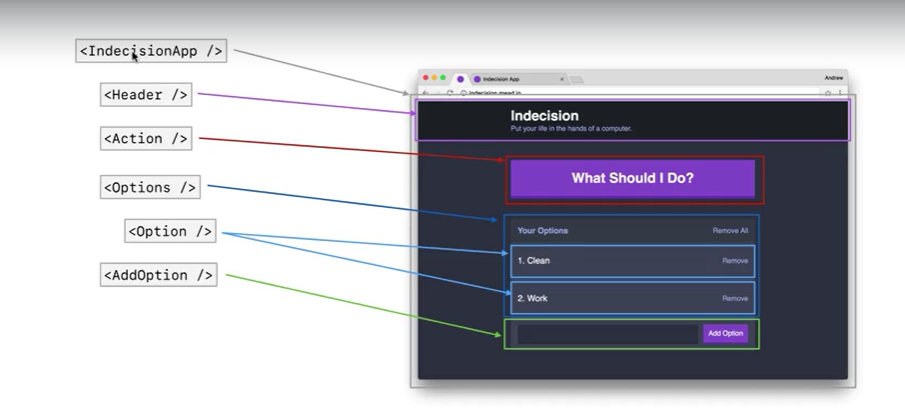
Use Math.Floor to get the whole number.

**Components in react:**

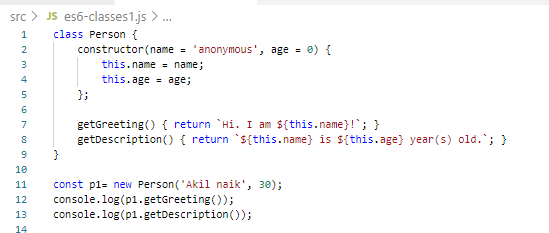
As said react is component based development so here each jsx build one component

For example namely header, footer, profile, login form

And each component is completely responsible for performing all the action related to each components.



**ES6 class:**



Use key word: class

For constructor function use Constructor(list of parameter)

Assign values down in the constructor functions.

Class properties we use this key word to specify the class properties. Define class method directly without any function key word

No arrow function is allowed.

**Overloading and inheritance:**

Inheritance an be achieved with extends key word

Can call parent class constructor method inside the child class using the keyword

Super(parameter list)

Inside the child class define method with names same as parent its called function overloading.

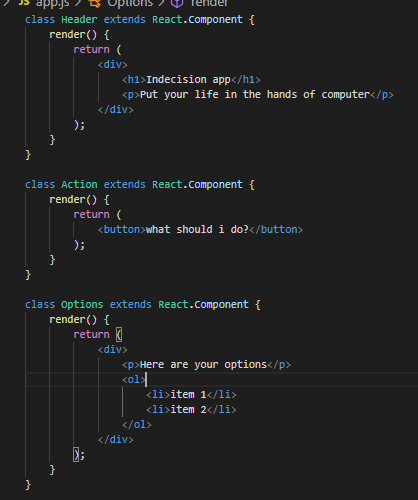
Inside the child class suppose you want to call parent class method use super key word

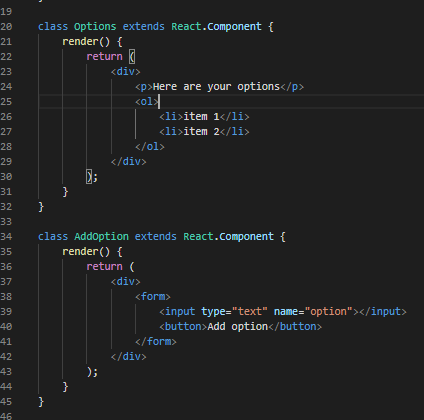
Super.ParentMethodname();

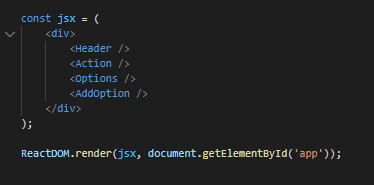


Building a component:

1. Each element in the page is one component
2. Component is a ES6 class file that extends React.Components base class
3. Overrides the render() method that us responsible for returning the html related to that component.
4. While rendering actual html make JSX tag with component name as XML elements







1. Finally use ReactDOM s render method to render final JSX

**Nesting of components:**



In JSX x called xml because we use component class as Xml element

**How components communicate with each other:**

**Component props:**