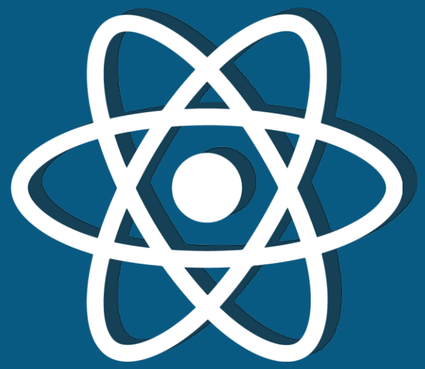


React With Redux Certification Training

COURSE OUTLINE
MODULE 01

1. **Introduction to Web Development and React**
2. Components and Styling the Application Layout
3. Handling Navigation with Routes
4. React State Management using Redux
5. Asynchronous Programming with Saga Middleware

6. React Hooks
7. Fetching Data using GraphQL
8. React Application Testing and Deployment
9. Introduction to React Native
10. Building React Native Applications with APIs

Topics

Following are the topics covered in this module:

- Building blocks of web application development
- Single-page and Multi-page applications
- Different client side technologies
- MVC architecture
- Introduction to React
- NPM modules
- “create-react-app” package
- Installation of React
- JSX and its use case
- DOM
- Virtual DOM and its working
- ECMAScript
- Difference between ES5 and ES6

Objectives

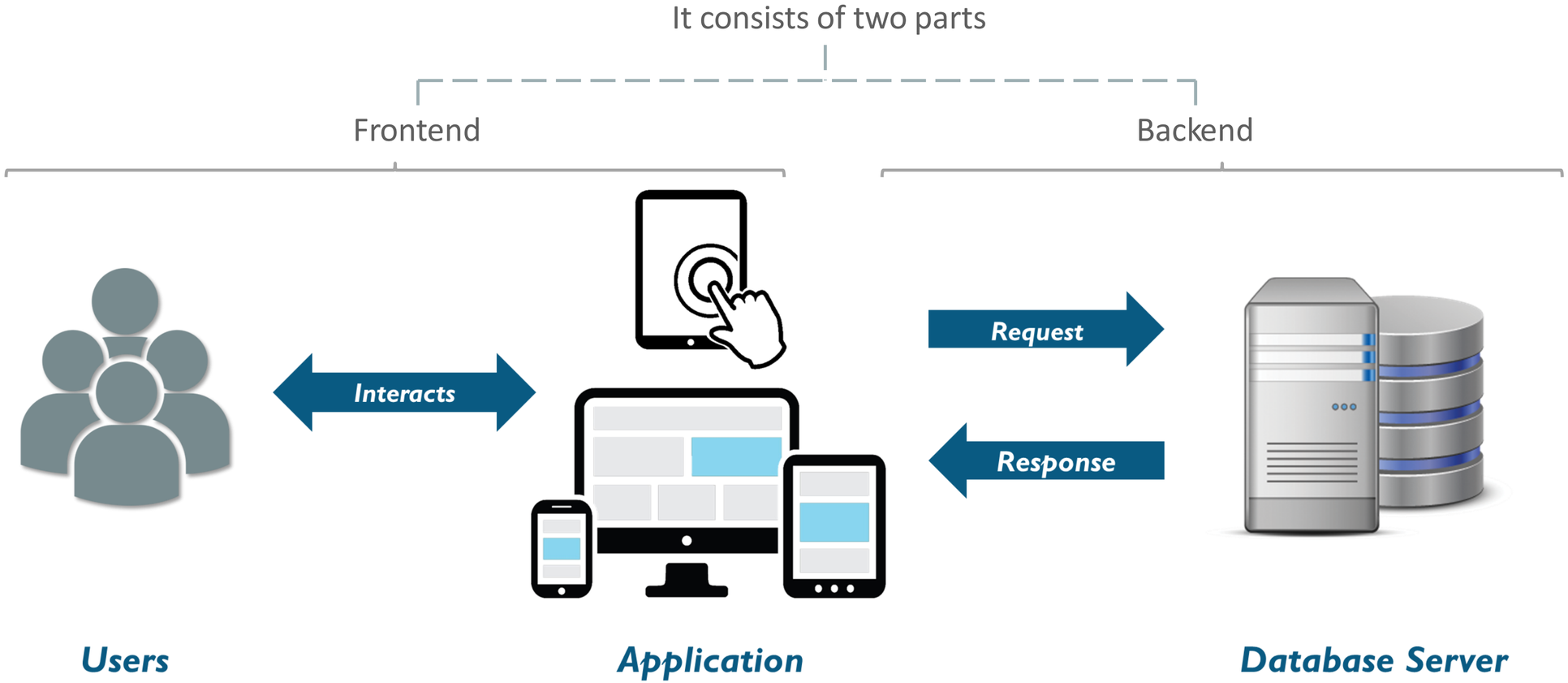
After completing this module, you should be able to:

- Understand components of Web application development
- Differentiate between Multi-page and Single-page Applications
- List different frontend technologies
- Explain MVC architecture
- Identify different NPM packages
- Install React and design your first React application
- Design React applications using JSX
- Understand working of virtual DOM
- Differentiate between ES5 and ES6

Building Blocks Of Web Application Development

Building Blocks Of Web Application Development

Web Application Development is the creation of an application program that reside on remote servers and are delivered to the user's device, over the internet through browser interface.



Frontend Development And Backend Development

Frontend Development refers to constructing what a user sees when they load a web application – the content, design and how you interact with it.

01

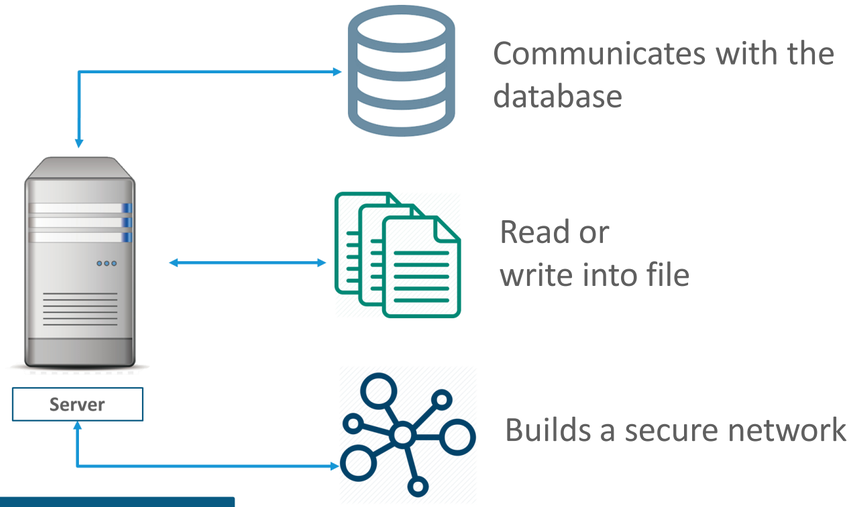
Components of Frontend

HTML

JavaScript

CSS

Components of Backend



Communicates with the database

Read or write into file

Builds a secure network

02

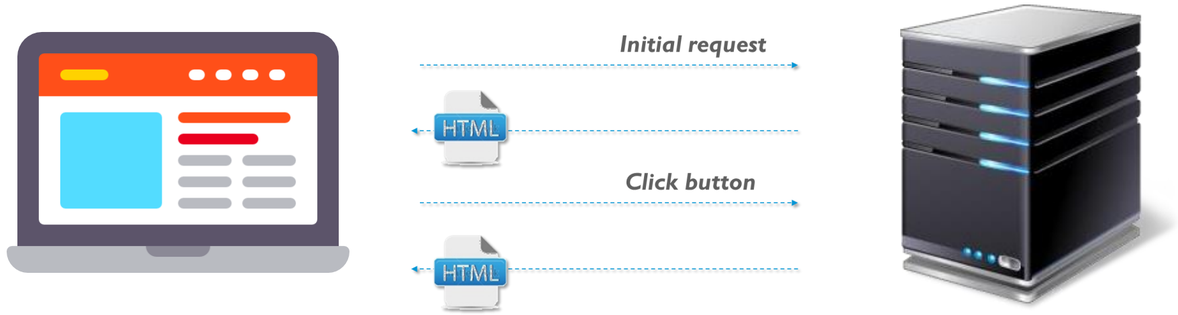
Backend Development refers to the server side of an application, where you have to develop a business logic to manage the content, along with security and structure, in order to ensure that application remains operational.

Ways Of Developing Web Application:

1. Multi-page Application
2. Single-page Application

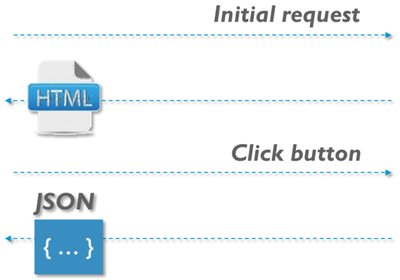
Multi-page Application (MPA)

- In ***multi-page application*** each time you click on a link or interact with the application, a new page is downloaded from the server and then rendered in the web browser
- Here content is organized on individual pages that are usually **static**
- They do not change in response to user's action
- A **brand new page** with its own static content is served when a user clicks any button

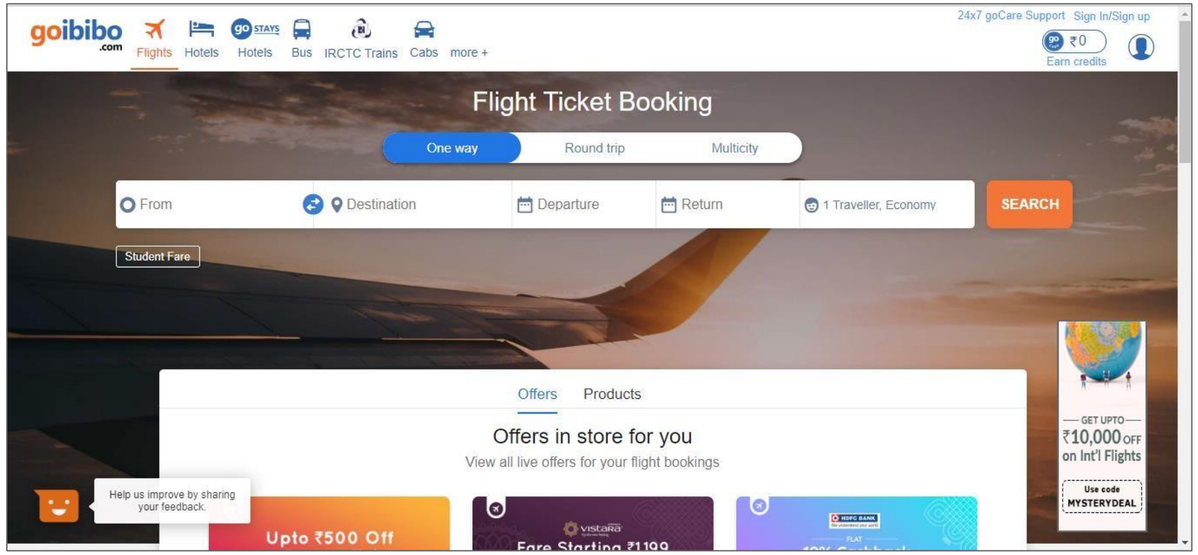


Single-page Application (SPA)

- A ***single-page application*** is an application that works inside a browser and does not require page reloading during use
- Instead of serving a brand new page to the user, SPA swaps out the old content for new in case of any user interaction
- SPA is **faster**, more **responsive**, **compact, easy to develop and deploy**
- SPA and all its content is **only loaded once**, when the user first interacts with the web page

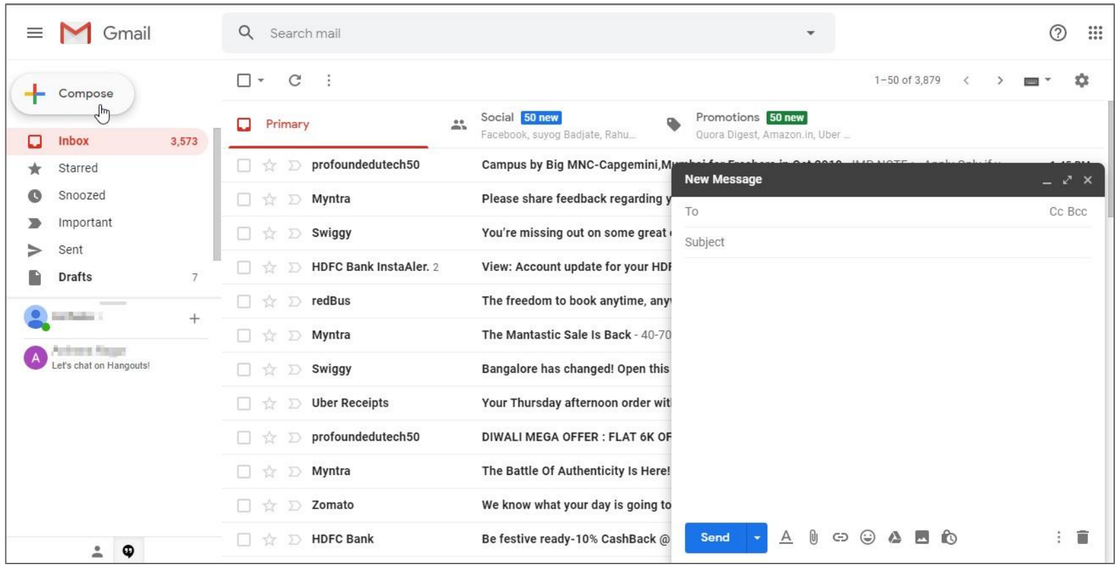


Examples



Multi-page Web Application

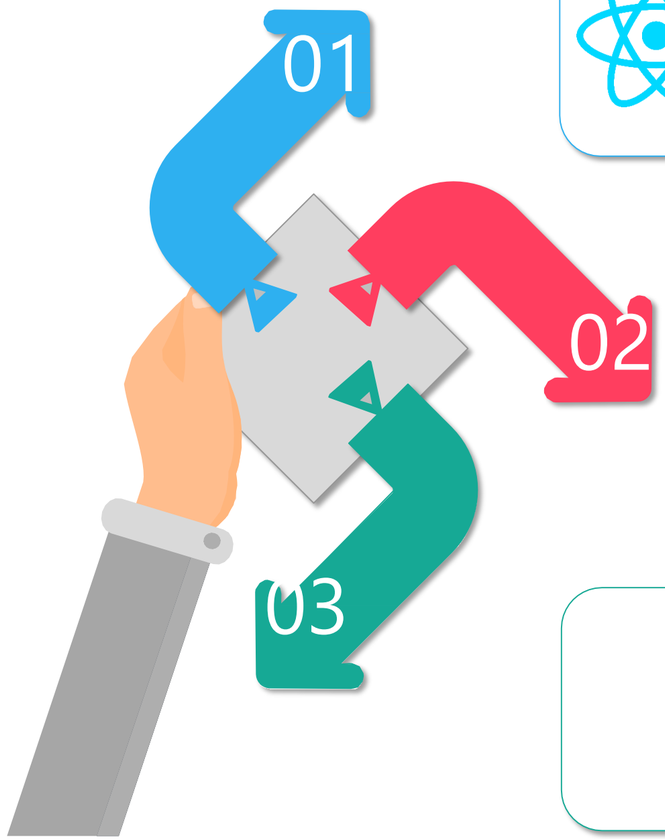
Single-page Web Application



Different Client Side Technologies

Different Client Side Technologies

The most commonly used frontend (client side) technologies are:



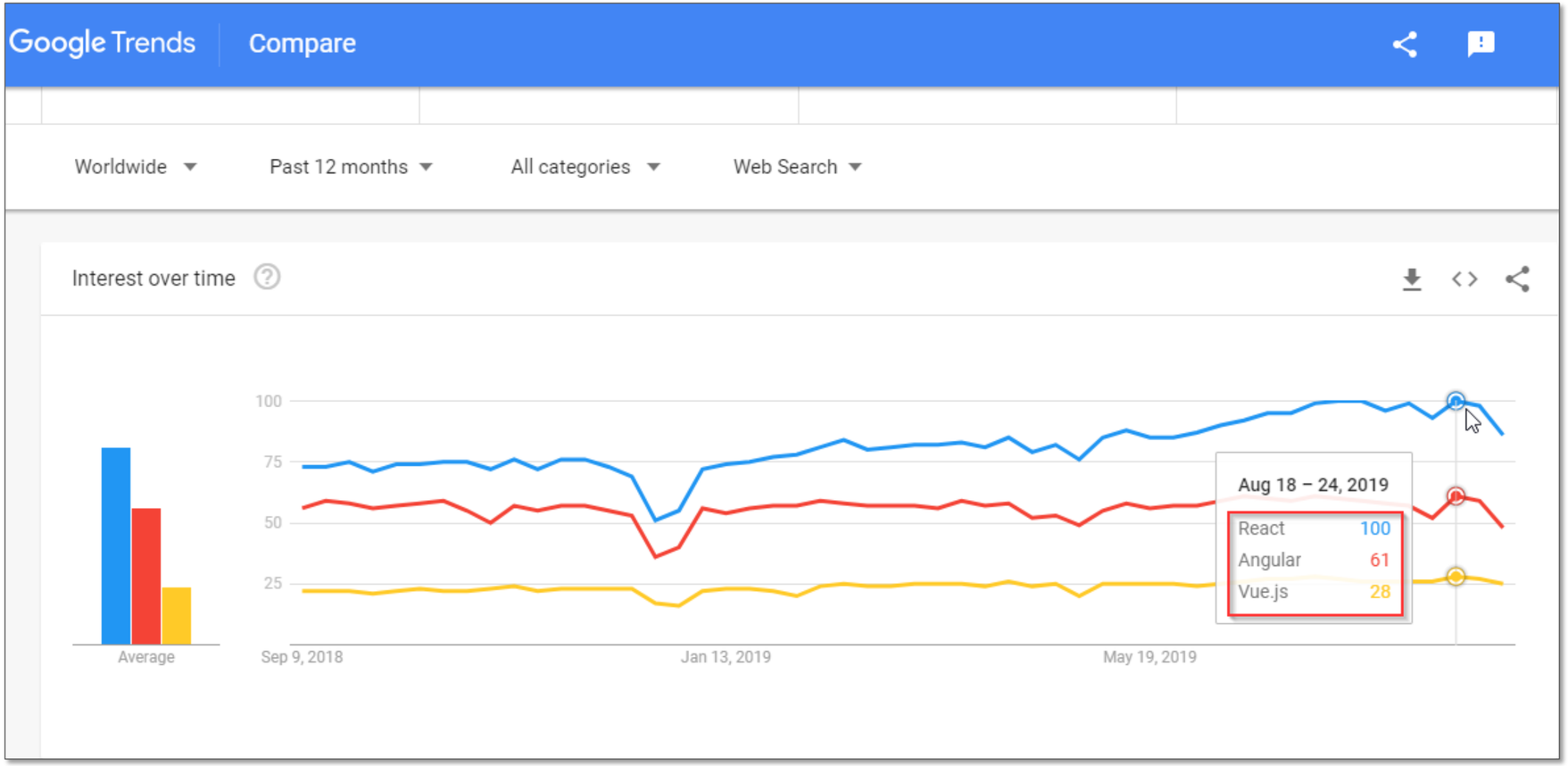
React

ANGULAR

Vue.js

Market Trends Of Frontend Technologies

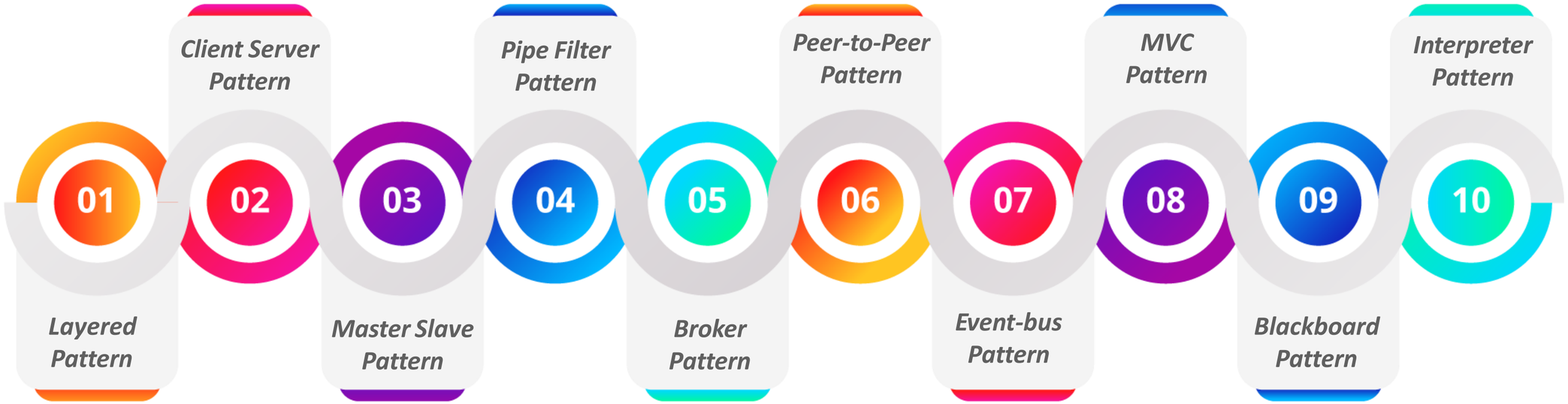
As per ***Google*** Trends React attracts major market crowd than any other client side technology.



Enterprise scale systems are designed with lots of pre planning.
Before starting major software development we have to choose ***a suitable architecture*** that will provide us desired functionality and quality attributes.

Web Application Architecture Patterns

Architectural pattern is a general reusable solution to a commonly occurring problem in web development with a given context. Given below is a list of 10 commonly used architectural patterns :



Usually ***MVC pattern*** is the opted architectural pattern by ***frontend technologies*** to develop web applications.

Model-View-Controller Architecture

Model-View-Controller (MVC) Architecture

Model–View–Controller is an architectural pattern commonly used for developing user interfaces that divides an application into three interconnected parts- Model, View and Controller

It is a software code that controls the interactions between the Model and View.

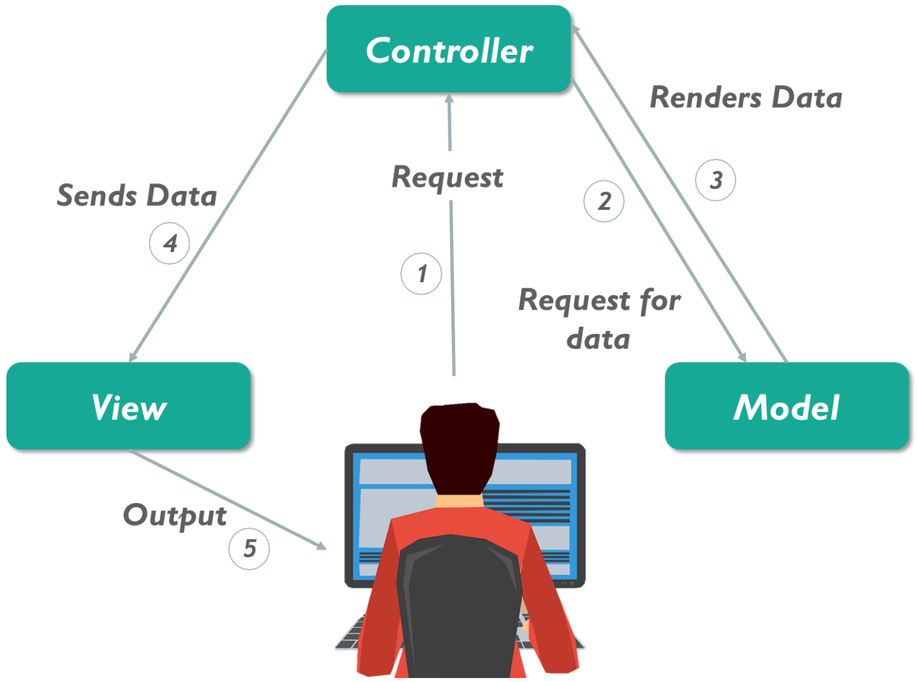
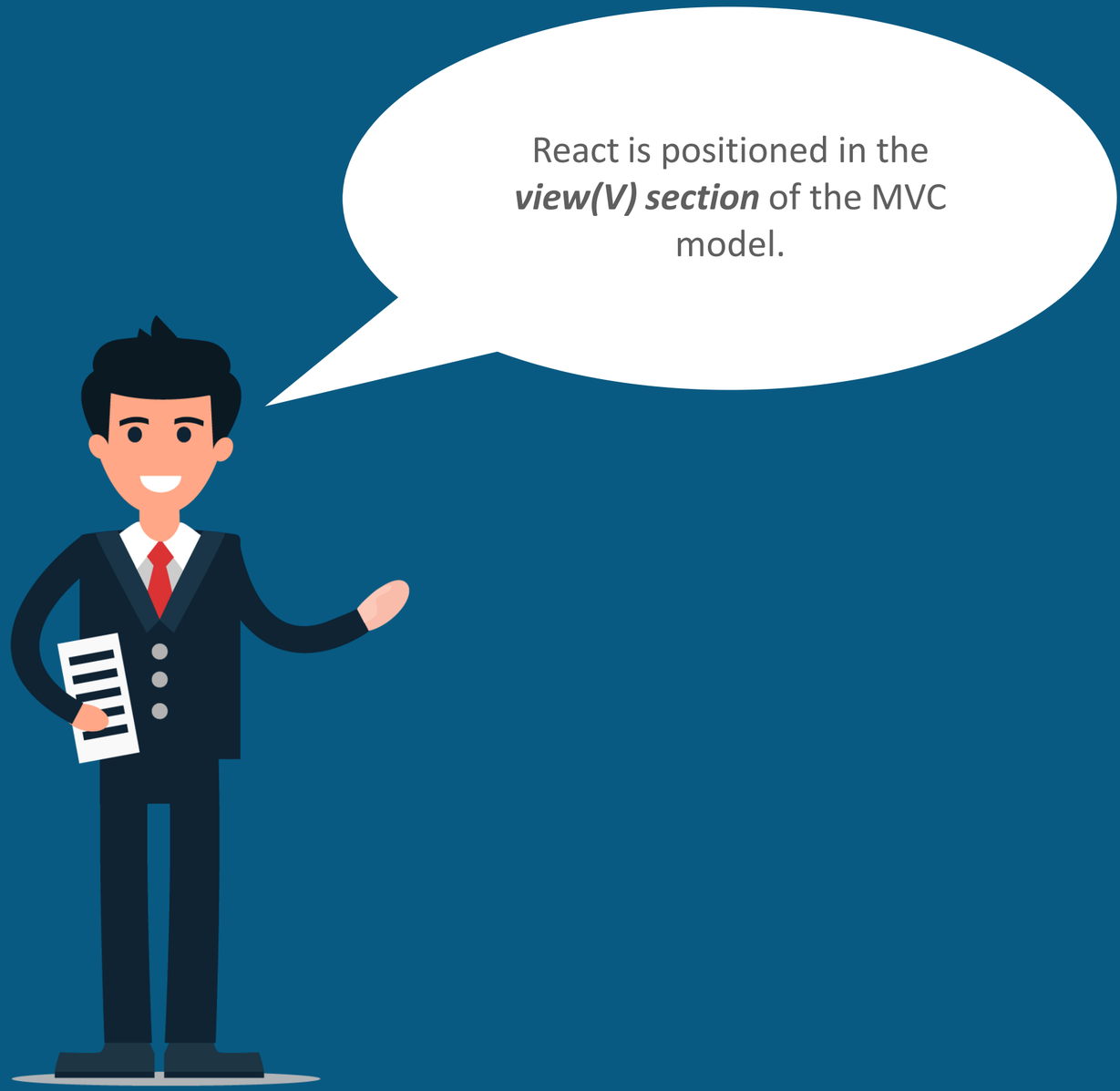


Fig: Model View Controller

A view is what you see when you visit a site.
Eg: blog post, contact form etc.
It sends the data to client.
Generally, views are HTML documents.

It represents application data.
It retrieves, stores and maintains the data of the application in a database.

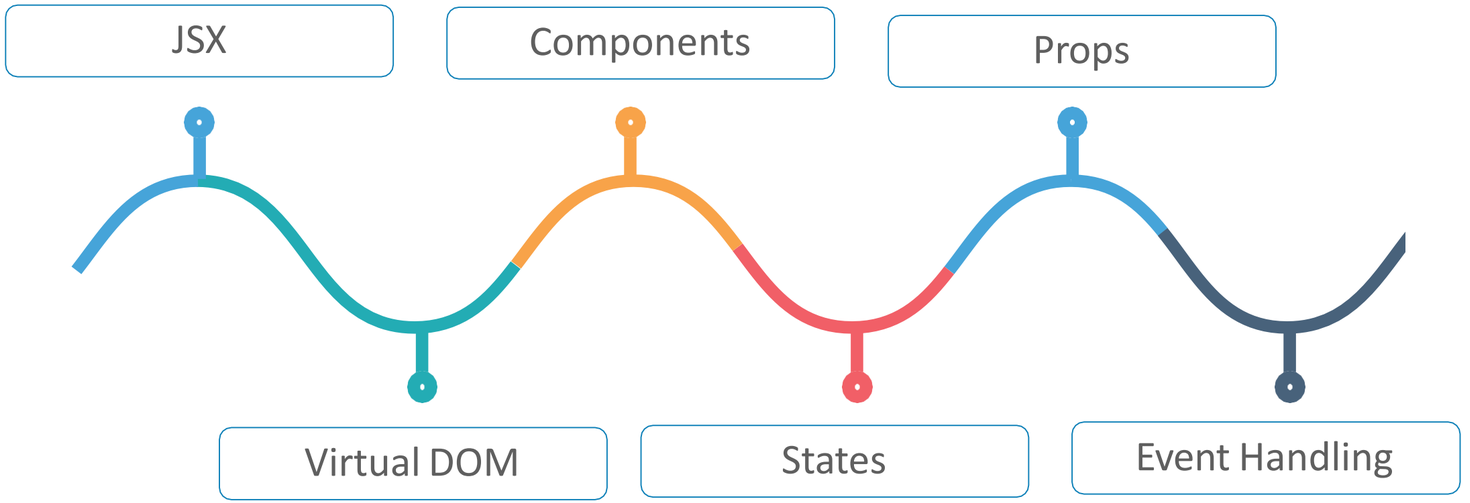

React is positioned in the
view(V) section of the MVC
model.

What Is React?

React (also known as React.js or ReactJS) is a JavaScript library for building user interfaces.

React is used in the development of single-page web application or mobile applications, as it is optimal for fetching rapidly changing data that needs to be recorded.

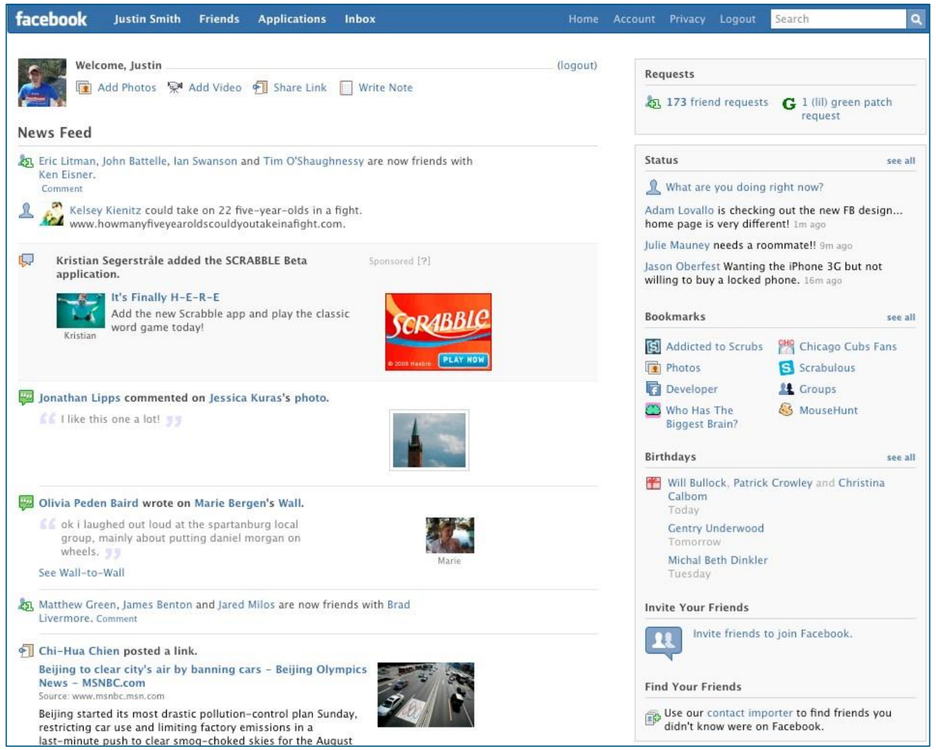
Elements of React:



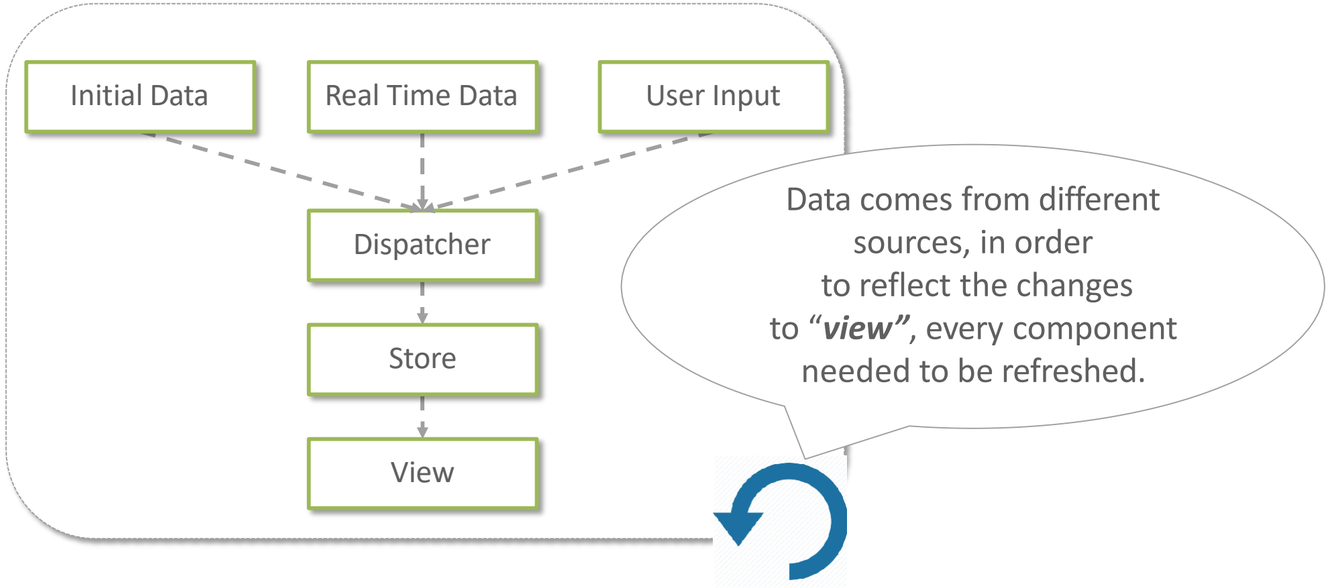
React Case-Study

Problems Faced By Facebook Before React

Before React, to know about every new newsfeed user needed to reload the complete Facebook page.



Data Flow within the website:

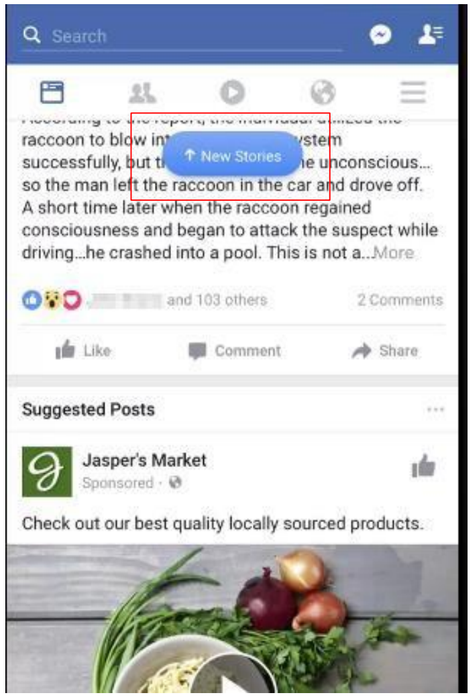


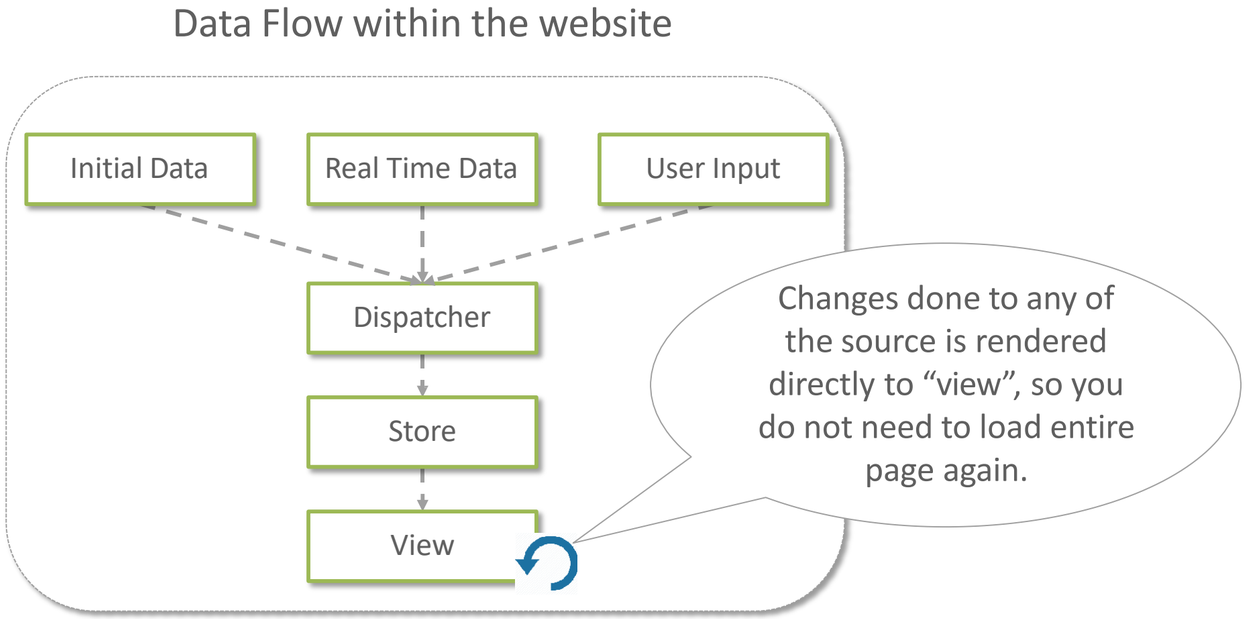
For every reload a new DOM is created which led to more memory consumption thereby slowing down the performance of the website.

Source: http://espressobin.net/wp-content/uploads/2016/11/p40-casestudy.pdf

Facebook: Solution

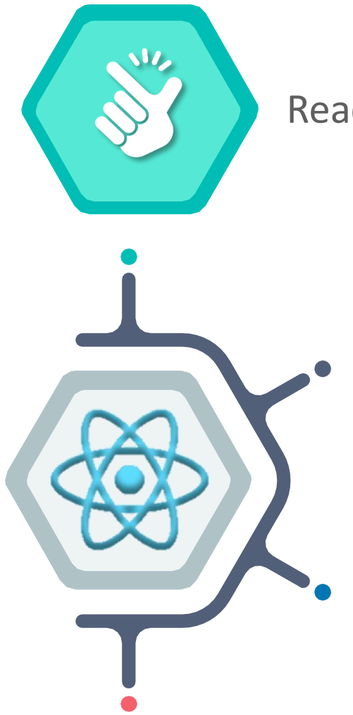
Facebook after implementation of React became more user friendly. When new data is added, a “New Stories” notification appears. Clicking it, will automatically refresh the newsfeed section





React uses ***virtual DOM*** which optimizes the memory consumption and results to faster load of web pages.

Why Should We Learn React?



React offers a ***lower learning curve,*** all you need to know is ***JavaScript and HTML***

Elements like JSX, Components, Props, State and Virtual DOM, lets React to ***develop*** application faster with ***smaller code size, easy deployment and rich user interface***

Top corporations such as ***Facebook, New York Times, Yahoo! Mail, Netflix, Instagram*** and more use it to solve their user interface related issues

As React has highest market popularity so obviously there are ***more job availabilities for React developers*** and average salary of React Developer ranges from ***$74,033*** per year for Software Engineer Intern to ***$112,143*** per year for JavaScript Developer

Source: https://www.indeed.com/salaries/Reactjs-Developer-Salaries

Node Package Manager

Node Package Manager (NPM)

1. NPM is the world's largest Software Registry with 10 lakh code packages. Developers use NPM to share software. Many organizations also use NPM to manage private development

2. NPM is free to use, you can download all NPM public software packages without any registration or login.

3. All NPM packages are defined in a file called *package.json*. At least two fields must be present in the definition ***file***: ***name*** and ***version***

4. **NPM** is installed with **Node.js.** This means, you have to install Node.js to get NPM installed in your system. Refer LMS to install Node.js

package.json

A package.json in Node.js contains all the files you need for a module, where modules are JavaScript libraries you can include in your project..
package.json is used to store the metadata associated with the project as well as to store the list of dependency packages.

The metadata can be categorised into two:

Identifying metadata: It consist of the properties to ***identify the project*** such as the name, current version of the module, license, author of the project, description about the project, and more

Functional metadata: It consist of the functional properties of the project such as the entry point of the module, dependencies in project, scripts being used, repository links of project, and more

NPM’s Local Packages vs Global Packages

Local Package

- It is installed in the directory where you run the ***npm install <package-name>***, by placing there codes under node_modules
- It is imported to your code by ***import ('package-name’)***
- Can have different versions of same package in different application
- Easy to handle when there is an upgradation of a package
- Occupies more memory

Global Package

- It is placed in a single file of your system, regardless of where you run ***npm install -g <package-name>***
- It is imported to your code by ***import('package-name’)***
- Can have only a single version of same package
- Difficult to handle when there is an upgradation of a package
- Occupies less memory

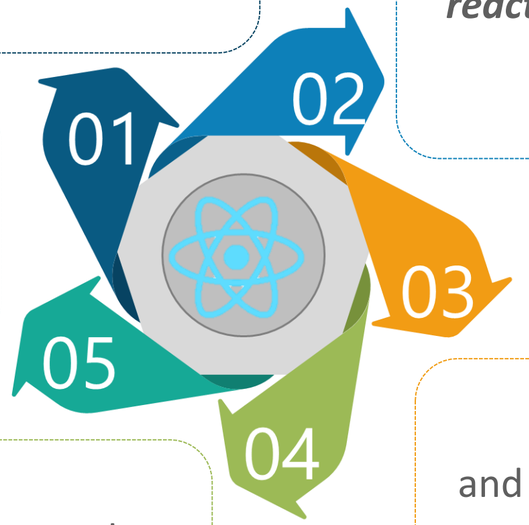
NPM Libraries Involved In React Installation

The necessary NPM libraries for starting any React application are:

react: It contains only the functionality necessary to define React components

react-dom: It serves as an entry point to the DOM

babel: It is used to convert the ES6 code to ES5 version of javaScript



webpack: It is a module bundler, and bundles JavaScript files for usage in a browser

react-scripts: It includes scripts and configurations used by ***create-react-app***

create-react-app

create-react-app is a NPM package and easiest way to start up a React application.

- It provides a ready-made React application starter, so you can dive into building your application without having to deal with ***webpack*** and ***babel*** configurations
- Create-react-app setup includes:

Provides a ***testing*** environment with Jest and allows to build the React app

Development server with quick reloading

Ready for ***ES6+ syntax***

Bundles all your JavaScript and assets

Comes with ***CSS autoprefixer***, ***SASS and CSS Modules*** support

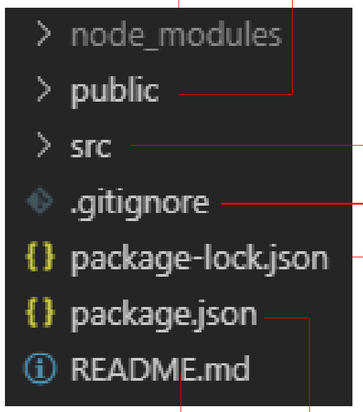
NOTE:

Install this package using the command:

npm create-react-app <app-name>

Demo: Installation Of React

Folder Structure In React



This folder contains all dependencies and sub-dependencies specified in ***package.json and*** used by React application

It contains files which don't require additional processing by ***webpack***

It contains the necessary components and application code

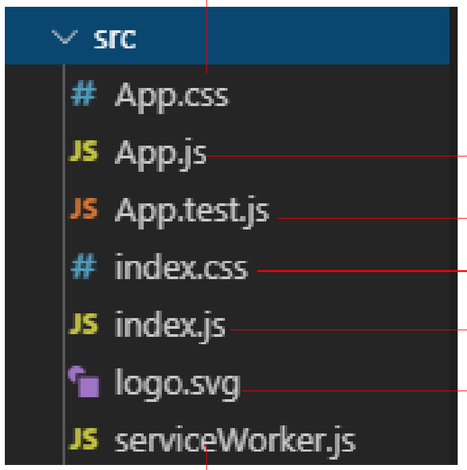
It is a standard file used by source control tool ***git*** to identify files and folders to be ignored while committing the code. Until and unless this file exists, the ***create-react-app command*** will not create a git repo in this folder

It maintains a ***history of changes*** in package.json, so that at any point of time, previous changes can be looked back in the ***package-lock.json file***

Contains ***dependencies and scripts*** required for building the application

Provides ***information*** about the application

Sub-Folder Structure In React



Designs the required stylings for code present in App.js

Contains the main application code

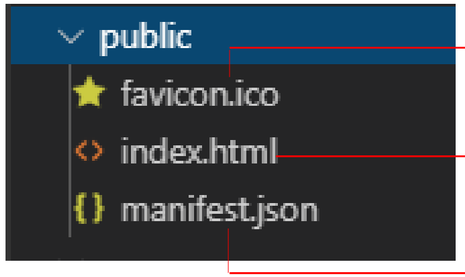
Contains the required test suite for application

Designs the required styling for code present in index.js

It provides an entry point to the web application code

It provides the React Logo

It takes care of caching and updating files for the end user. It helps in offline capability and faster page loading after the first visit



It is the header icon

It is the web template which is considered as an entry point for the web application

manifest.json file contains configurations when your application is used for Android app

JSX

JSX

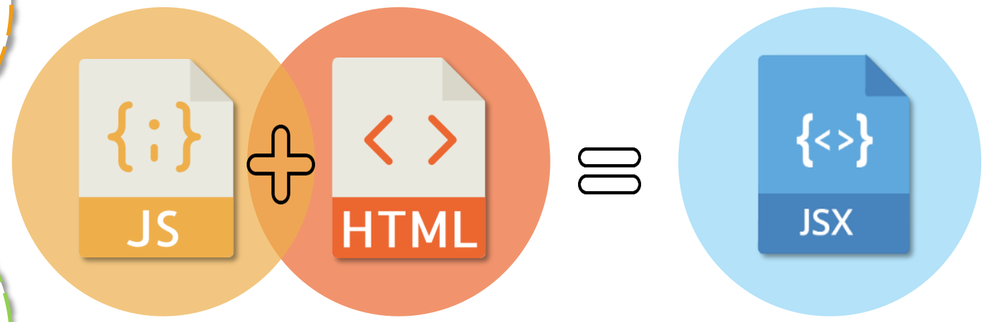
JSX stands for JavaScript XML, which lets you to write the HTML code within the JavaScript.

JSX is an extension to the JavaScript language syntax. It is easy to write JSX templates if you know the HTML

It is type-safe as most of the errors can be caught during compilation

It makes use of ***render function*** to return the single HTML element at a time

React components are typically written using JSX, although they do not have to be (components may also be written in pure JavaScript)



JSX Use Case

Regular JSX

Returns the HTML Representation

```
var MyComponent =
React.createClass({
    render : function () {
        return (
            <div>
            Hello World!!!
        </div>
    );
    });
```

Specifying Attribute

```
var styles={ backgroundcolor: 'cyan'};
var MyComponent=React.createClass({
    render : function () {
        return (
            <div style={styles}>
            <h1>Header</h1>
            </div>
    );
    }
});
```

Adding Attributes

JSX Nested Elements

Returns multiple elements

<h1>,<h2>,<p> nested inside

```
var MyComponent = React.createClass( {
    render : function () {
        return (
            <div>
            <h1>Header</h1>
            <h2>Content</h2>
            <p>This is the content!!!</p>
        </div>
    );
    }
});
```

Embedding JavaScript

```
var MyComponent = React.createClass({
    render: function () {
        return(
            <div>
                <h2> {2+4} </h2>
            </div>
        );
    }
});
```

JavaScript Expression

Document Object Model

Document Object Model

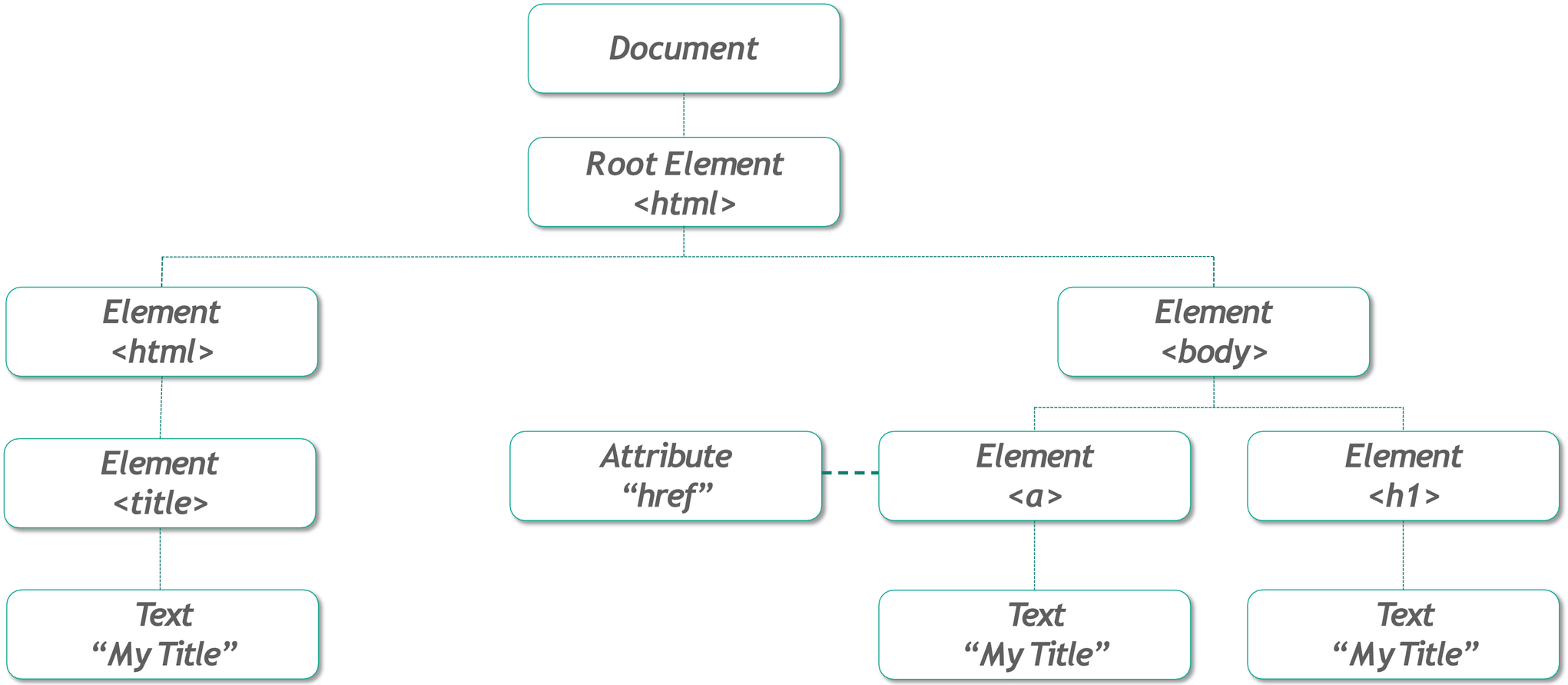
DOM is a standard logical representation of any webpage.

With the Document Object Model, programmers can ***create and build*** documents

DOM is a ***tree-like*** structure that contains all the ***elements*** and its ***properties*** as its ***nodes***

DOM provides a ***language-neutral interface*** that allows accessing and updating of the content of any element of a webpage

HTML DOM Tree Of Objects



DOM Manipulation: Disadvantage

By now we have already discussed the importance of DOM and established the fact that DOM manipulation is the heart of the modern, interactive web. Unfortunately, it does have some drawbacks

1. Let us assume that you have a list that contains ten items. You are suppose to make some changes in the fifth item. Under such circumstances, most JavaScript frameworks would rebuild *the* ***entire list again***

2. This leads to ***more work*** than necessary. Just to implement the changes of one item, even rest nine get ***rebuilt*** unnecessarily

3. Rebuilding a list is not a big deal to a web browser, but modern websites can use ***huge amounts of DOM manipulation***. To address this problem, the React popularized new aspect called ***virtual DOM***

Introduction To Virtual DOM

The virtual DOM (VDOM) is a programming concept, where “virtual” representation of a UI is kept in memory and synced with the “real” DOM by a library such as ***ReactDOM.***

In React, for every DOM object, there is a corresponding “virtual DOM object”

A virtual DOM object is a *representation* of a DOM object, like a lightweight copy

A virtual DOM object has the same properties as a real DOM object, but it lacks the power to directly change what’s on the screen

Manipulating the virtual DOM is much faster, because nothing gets drawn onscreen

Features Of Virtual DOM

1

When you render a JSX element, every single virtual DOM object gets updated.

This sounds incredibly inefficient, but the cost is insignificant because the virtual DOM updates quickly

Features Of Virtual DOM

1. **Element Rendering**

2. Once the virtual DOM is updated, React compares the virtual DOM with a virtual DOM *snapshot* that was taken right before the update

Features Of Virtual DOM

1. **Element Rendering**

2. **DOM Comparison**

3. By comparing the new virtual DOM with real DOM, React figures out *exactly which virtual DOM objects have changed.* This process is called "***diffing***." Once React knows which virtual DOM objects have changed, then React updates only those objects on the real DOM

Features Of Virtual DOM

1. **Element Rendering**
2. **DOM Comparison**
3. **DOM Update**
4. In our example from earlier, React would be smart enough to rebuild your one checked-off list-item, and leave the rest of your list alone. This makes a big difference! React can update only the necessary parts of the DOM. Reacts reputation for performance comes largely from this innovation

Features Of Virtual DOM

1. Element Rendering
2. DOM Comparison
3. DOM Update
4. Improve Performance

ECMAScript

JavaScript was standardized by ECMAScript, to foster multiple independent implementations.

ECMAScript (or ES) is a scripting-language specification standardized by ECMA International in ECMA-262 and ISO/IEC 16262.

At present many browsers support only the ***ES5 version***

But an ***updated version ES6*** with better features is available,

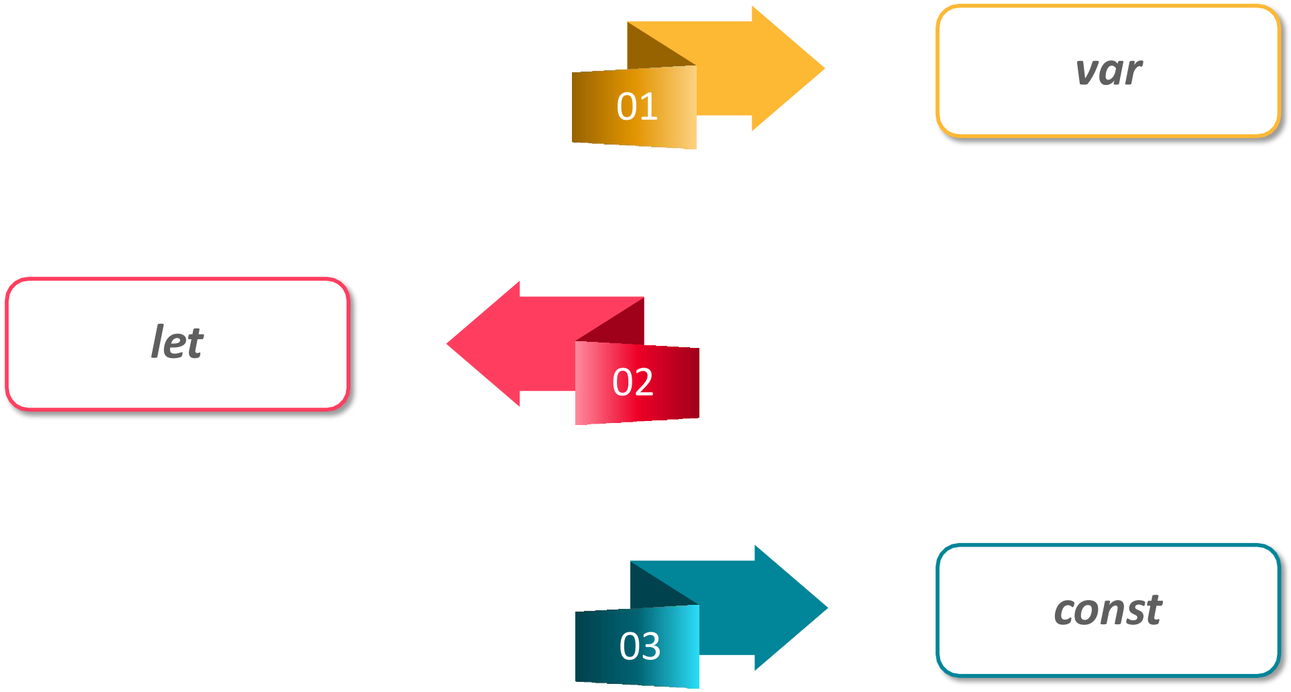
which makes it easier for developers to code.

ECMAScript Variables

ECMAScript Variables

Variables are the defined spaces in the memory, to store values in a program.

JavaScript supports ***dynamic typing***, where variables should be necessarily defined before they are used.
The majorly used ECMAScript variable types are:



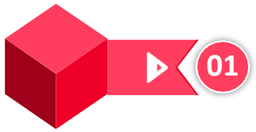
ECMAScript Variables: Var

var

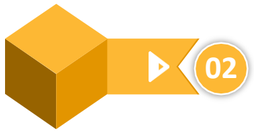
let

const

ES5 makes use of ***var*** keyword and it was the only way to declare the variables in JavaScript and it had certain limitations like:



It has a ***function scope,*** that is if you want to restrict the access of variables within the functions it should be defined inside the function, otherwise it will belong to the global scope



Variables can be ***declared more than one time*** and their ***values can be re-assigned*** anytime in the same project

ECMAScript Variables: Var (Example)

var

let

const

Input Screen:

```
var x = 10
function test() {
var x = 100
console.log("value of x as per defined function "+x)
}
console.log("value of x before declaration of function "+x)
test()
```

Declaring variable globally

Declaring variable locally

Output Screen:

```
PS C:\Users\archana\Desktop\Node.js\Node.js-Demo\firstapp> node ES.js
value of x before declaration of function 10
value of x as per defined function 100
```

ECMAScript Variables: Let

var

let

const

ES6 introduced variables ***let and const*** which provide ***Block Scope.***

- ***Block Scope*** is the area within if, switch conditions, for and while loops, where variable declared within curly braces ***{}*** remains available only in this scope

- With ***let*** you can ***re-assign*** the value of the variable any time but ***you cannot declare*** the variable more than one time

Example:

```
if(true)
{
let x= 1;
console.log(x);
}
console.log(x)
```

Prints 1

Throws reference error : x is not defined

Difference Between Var And Let

Var

Let

Const

Difference between ***var*** and ***let***

```
var x = 10;
var x = 20;
console.log(x);
```

```
PS C:\Users\archana\Desktop\Node.js\Node.js-Demo\firstapp> node ES.js
20
```

```
let x = 10;
let x = 20;
console.log(x);
```

```
PS C:\Users\archana\Desktop\Node.js\Node.js-Demo\firstapp> node ES.js
C:\Users\archana\Desktop\Node.js\Node.js-Demo\firstapp\ES.js:2
let x = 20;
    ^

SyntaxError: Identifier 'x' has already been declared
```

Hence, any variable declared using the ***let*** keyword is assigned the ***block scope.***

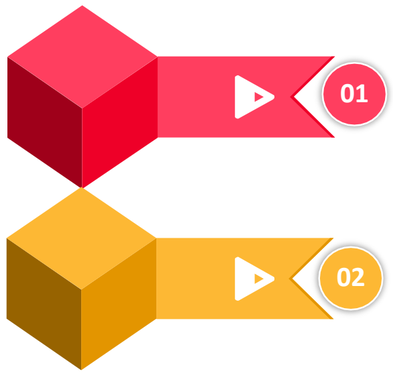
ECMAScript Variables: const

var

let

const

The value assigned to a ***const*** variable is ***immutable.***



Value must be assigned to a const variable at the time of declaration and it later ***cannot be re-assigned or re-declared*** within the same program

In case of declaring a new array using const variable, you ***can change*** the value of an element within the array

Example:

```
const a =0
const a = 1
const b = [1,2]
b.push(3);
b[3] = 4
console.log(a,b)
```

Assigns value ‘0’to constant ***a***

TypeError: Assignment to constant variable

Sets b = [1,2,3]

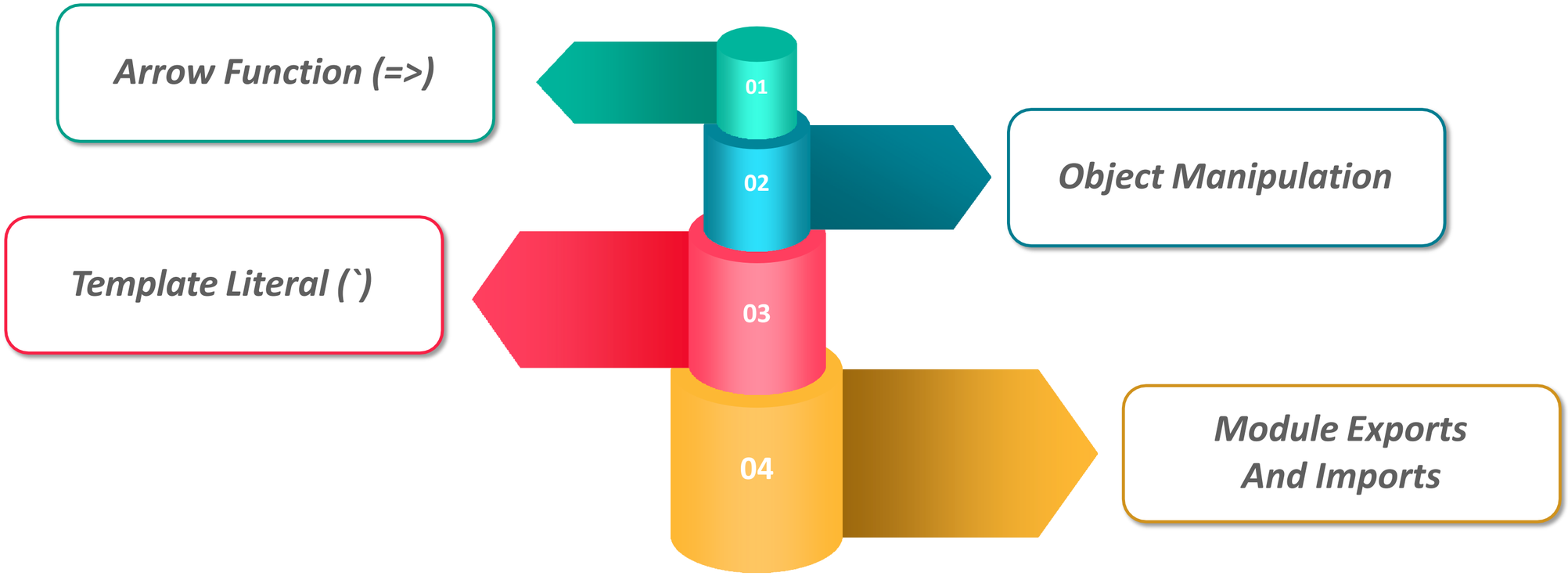
Sets b = [1,2,3,4]

Difference Between ES5 And ES6


The major difference between ES6 and ES5 is ES6 *syntaxes are shorter* compared to ES5
Lets have a look at this in upcoming slides.

Difference Between ES5 And ES6

ES6 features with shorter and better syntax than ES5 are as follows:



ES6 vs ES5: Arrow Function (=>)

- ***Arrow functions*** are the functions where you don't have to use curly brackets, neither type the **function** keyword
- They utilize a new token, '=>'
- ***Syntax: (parameters) => { statements }***

Arrow function can be majorly used in the following cases:

Return Number

Return Array

Return Number with parameter

Return Object

ES6 vs ES5: Arrow Function (=>)

Return Number

Return Array

Return Object

Return Number with parameter

ES5

```
function getNum()
{
  return 5;
}
```

ES6

```
const getNum = () =>5;
```

ES6 vs ES5: Arrow Function (=>)

Return Number

Return Array

Return Object

Return Number with parameter

ES5

```
function getArr()
{
  return [1, 2, 3];
}
```

ES6

```
const getArr = () => [1, 2, 3];
```

ES6 vs ES5: Arrow Function (=>)

Return Number

Return Array

Return Object

Return Number with parameter

ES5

```
function getObj()
 {
  return { a: 1, b: 2, c: 3 };
 }
```

ES6

```
const getObj = () => ({ a: 1, b: 2, c: 3 });
```

ES6 vs ES5: Arrow Function (=>)

Return Number

Return Array

Return Object

Return Number with parameter

ES5

```
function calcCircleArea(radius)
 {
  return Math.PI * radius * radius;
 }
```

ES6

```
const calcCircleArea = (radius) => Math.PI * radius * radius;
```

ES6 vs ES5: Object Manipulation

To merge the objects, you have to use the
Object.assign() method, which takes both objects as input and outputs the merged object. The spread operator makes merging objects a breeze for the developer.

Merge objects with the spread operator (…)

```
var obj1 = { a: 1, b: 2 };
var obj2 = { c: 3, d: 4 };

ES5: var obj3 = Object.assign(obj1, obj2);

ES6: var obj3 = { ...obj1, ...obj2 };
```

ES6 vs ES5: Template Literal (`)

- ES6 offers an alternate syntax to define strings known as ***template literals***
- This syntax uses ***back tick (`)*** rather then regular quote ***(')*** or double-quotes ***(")*** as delimiter
- The advantage of this syntax is, it can interpolate variables or expressions using ***${expression}*** inside the string

ES5

```
const x = 1;
 const y= 2;

const z = ‘value of x is ‘ + x ’ and value of y is ‘ + y’’;
```

ES6

```
const x = 1;
 const y = 2;

const z = `value of x is ${x} and value of y is ${y}`;
```

ES6 vs ES5: Module Exports And Imports

ES6 syntax is more readable and it comes up with keyword export default.

- ES6 also provides us with an ability to export and import multiple variables from a single module

- So in your module file if you export your modules :

  export const a = 1;
  export const b = 2;

- You can import them all together:
  import {a,b} from ‘./testModule’;

Export module

var testModule = { a: 1, b: 2 };

ES5: module.exports = testModule;

ES6: export default testModule;

Import module

ES5:
var testModule = require(./testModule)

ES6:
import testModule from ‘./testModule’;


Questions


Comments
Ratings
Suggestions
FEEDBACK
Survey
Likes
Ideas

Thank You

For more information please visit our website
www.edureka.co