Lecture 1:Solid Principles and Pratices (2019-02-24 17:27)

Lecture 1: Solid Principles and Practices

I am going to discuss about current principles especially SOLID principles as well as industry practices used in Software Engineering. First we need to look at two factors that affect the code.

Coupling means the degree to which a unit is dependent on other units. **High coupling** leads to difficulty in extending functionality of a unit and increase complexity. **Low coupling** makes it easier to extend functionality of unit and reduce complexity.

Cohesion means responsibility of the class. **Low cohesion** leads to a unit having multiple responsibilities. **High cohesion** leads a unit having a single responsibility.

S.O.L.I.D Principles

It is an acronym for 5 design principles in object oriented programming. It was first introduced by Robert C. Martin in his 2000 paper "Design Principles and Design Patterns". However Michaels Feathers identified the actual SOLID acronym. These principles are used with OOP principles such as Abstraction, Encapsulation, Inheritance and Polymorphism.

S : Single Responsibility

O : Open-Closed Principle

L : Liskov Substitution

I : Interface Segregation

D : Dependency Inversion

Single Responsibility Principle

A single unit (class, module or function) should perform only one job or responsibility. Single Responsibility leads to low coupling and high cohesion. In addition due to this principle a unit would be small and maintainable. As a result a unit will have better readability and complexity would be less.

Open-Closed Principle

A unit which is a class, module or function can be extended for new methods or behaviors but closed to prevent modification of existing code. For instance existing code for a class cannot be modified but new methods can be included into the class. Strategy Pattern follows this principle.

Advantages

- Low coupling
- Improvement in Readability
- Risk of breaking functionality reduced

Liskov Substitution Principle

Idea is that objects can be replaced by instances of their child classes without affecting the functioning of the system from client's view. To achieve this one must use the base class and get the result that is anticipated. When representing an object, we should model classes based on the methods.

Liskov principle is strongly related to Open-closed principle.

Interface Segregation Principle

This principle focuses on the way we code interfaces. If an interface is getting large, we need to break interface into small interfaces that are specific. In addition interface will be defined according to the client who uses it which means interface only methods that are relevant to the client.

Dependency Inversion Principle

General idea is that high level modules which uses complex logic should be easily reusable and unaffected by changes in low level modules which provides utility features.

For instance let us look at the layered architecture. High level module such as business layer does not get affected by implementation details from presentation layer and persistence layer.

Practices

Set of guidelines that needs to be followed when developing code. Practices help team members to understand the flow of work process. As a result organization grows and accelerates delivery.

Version Control

It is a system that keeps tracks of changes made to a file or set of files overtime so that you can recall a particular version.

Version Control uses a repository which is central file storage location that stores several versions of a file. Three elements of Repository are trunk, branch and tags.

Advantages

Allows many people to work on a same project simultaneously.

- Provides a backup of previous versions as well as current version.
- Getting feedback from other people effectively and quickly.
- Tool for continuous integration.
- Tool for code review.

GIT is a distributed version control System.

Continuous Integration

A practice which allows testers or developers to integrate code into a shared repository daily.

Advantages

- Catch issues early and fix them.
- Spends less time debugging and more time developing new functionality.
- Reduce integration problem to deliver the software quickly
- Improves confidence.
- Development process is consistent.

Code Review

Assessment of computer source code and ask questions like:

- Does the code follow coding conventions?
- Does the code support functional and non-functional requirements?
- Are there any errors or bugs in coding?
- What type of architecture is used?
- Did the developer follow OOP principles?

Then reviewer gives a feedback to the developer. Reviewer must not criticize the developer. Idea is to improve the quality of code.

Advantages

.

Knowledge is shared.

.

Improvement in code readability

.

Reduction of bug in code

.

Ensuring functional and non-functional requirements are satisfied.

Code Quality

Why do we need code quality?

- Consistency of code throughout software
- Code is reusable
- Code is readable

- Code is less complex
- Code follows coding convention
- Code follows a proper architecture and design pattern according to the problem

Attributes that helps in achieving code quality (Non-Functional requirements)

- Reliability
- Efficiency
- Security
- Maintainability

Benefits of code quality

- Competitive advantage in the market.
- Less bugs in the software
- Easy to improve the code in the future or modify code as it less complex and readable.
- Ensuring functional and non-functional requirements are satisfied.
- Low cost in maintenance.

Code reviews helps you to assess code quality.

Unit Testing

It is the lowest level on software testing that tests individual unit such a class, function or a module based on the low level design such as class diagram and sequence diagram. It is performed by software developers themselves or their colleagues. Sometimes it is used by independent software testers. Unit testing uses white box testing method.

According to the low level design, test cases are designed and then executed. Then one compares expected output and actual output. If both outputs are same then test case is a success. If they are not the same, then the test case is a failure.

Conclusion

In conclusion by following practices lead to excellent team work, better development process and qualitative software. In addition by applying principles such as S.O.L.I.D improves developer thinking as well as quality of software. By using principles we try to reduce coupling and increase cohesion.

Lecture 2: Introduction of JavaScript (2019-04-07 22:39)

Lecture 2: Introduction of JavaScript

Introduction

- JavaScript is a **single threaded** Language.
- JavaScript is **asynchronous** as it does not wait for input and output operations to be completed.
- JavaScript is **dynamically typed** as type of variable is identified during run time.

Let us look at

1.

Use of var, let and const

2.

Ways to create a JavaScript object

4.

Callbacks

5.

Promises

1) Use of var, let and const

var: It is function scope and the value of variable can be changed.

let: It is block scope and the value of variable can be changed.

const: It is block scope but value of variable cannot be changed

We must understand the meaning of function scope and block scope. **Block scope** is where

variables are declared inside a block **{ }** can be accessed from outside the block. **Function scope** is where

variables are declared locally in a function.

Example of let

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>let,var,const</title>
    <script type="text/javascript">

        //declaring counter i with let
        function printIntegers() {
            for (let i=0;i<5;i++){
                console.log(i);
            }
            console.log(i);
        }
        printIntegers();
    </script>
</head>
<body>

</body>
</html>
```

At line 13 there is a reference error since variable i only accessible inside the for block not outside the for block of function printIntegers () (Block Scope).

```
0
ReferenceError: i is not defined [Learn More]
1
2
3
4
```

Example of var

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>let,var,const</title>
    <script type="text/javascript">

        //declaring counter i with var
        function printIntegers() {
            for (var i=0;i<5;i++){
                console.log(i);
            }
            console.log(i);
        }

        printIntegers();
        console.log(i);
    </script>
</head>
<body>

</body>
</html>
```

At line 13 the console log displays value of i unlike when using let. Here the variable is accessible inside the function (function scope).

At line 17 the console log shows a reference error as variable is not accessible outside the function printIntegers ()

```
0
1
2
3
4
5
ReferenceError: i is not defined [Learn More]
    <anonymous>  http://localhost:63342/Test/var.html?_ijt=ip64f95boeesmgkjtfe35o4dal:17
```

Example of const

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>const</title>
6      <script type="text/javascript">
7          //value of the variable can not be changed when declared with const
8          const firstName="Kajavathanan";
9          firstName="Samuel";
   Attempt to assign to const or readonly variable
12
13
14         var firstName="Kajavathanan"
15         firstName="Samuel";
16
17         let firstName="Kajavathanan";
18         firstName="Samuel";
19
20     </script>
21 </head>
22 <body>
23
24 </body>
25 </html>
```

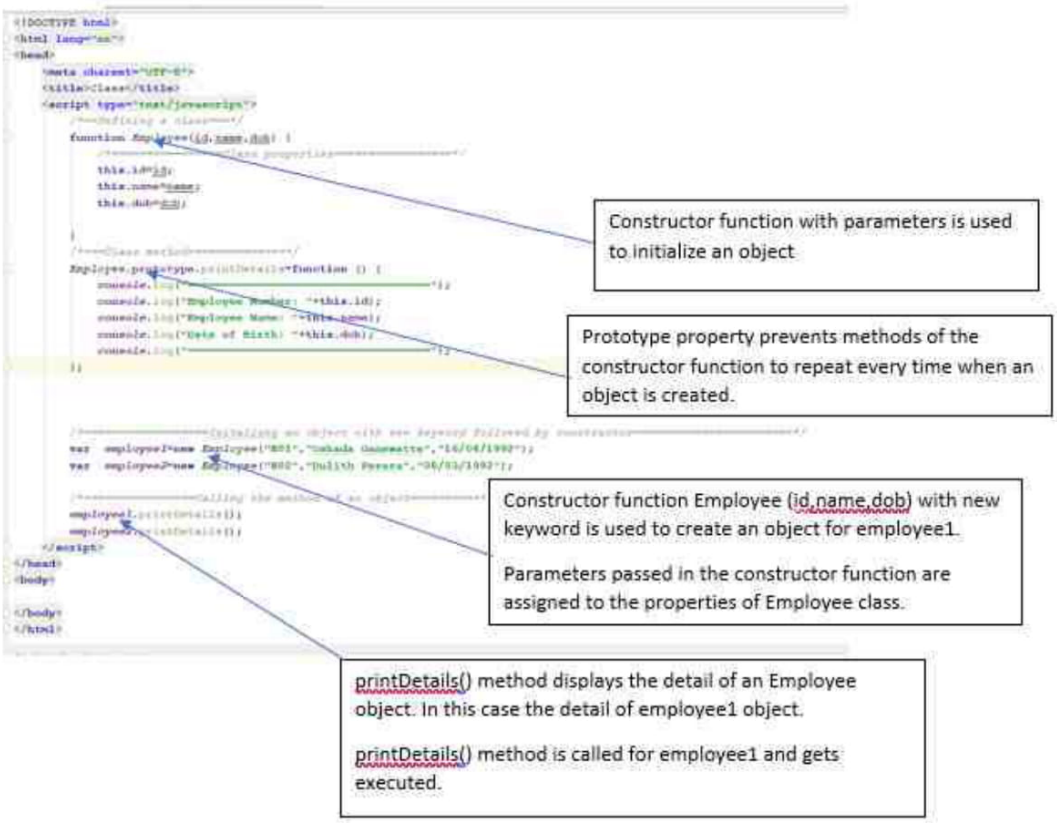
Notice variable firstName cannot be modified since it is declared with const.However variable firstName can be modified when declared with let or var as you can see in line 15 and line 18

2)

Ways to create a JavaScript object

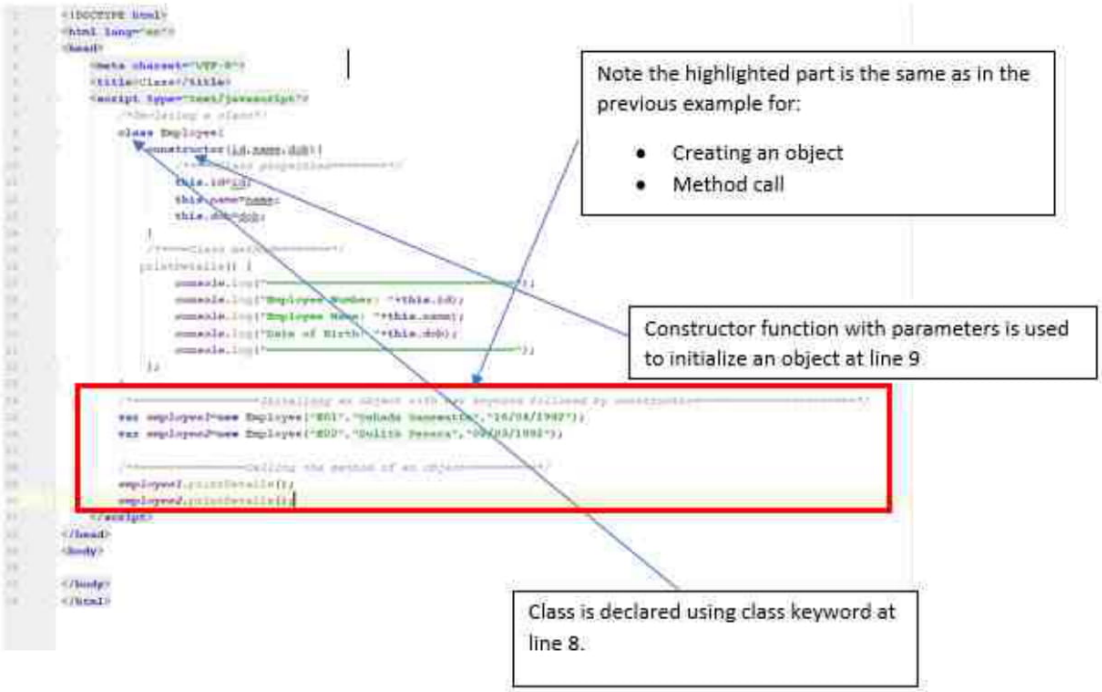
1)

Create an object using constructor function



2)

Create an object with class keyword



3) Create an object with Object Literal

Syntax for Object.create()

Object.create(prototypeObject,propertyObject)

1.

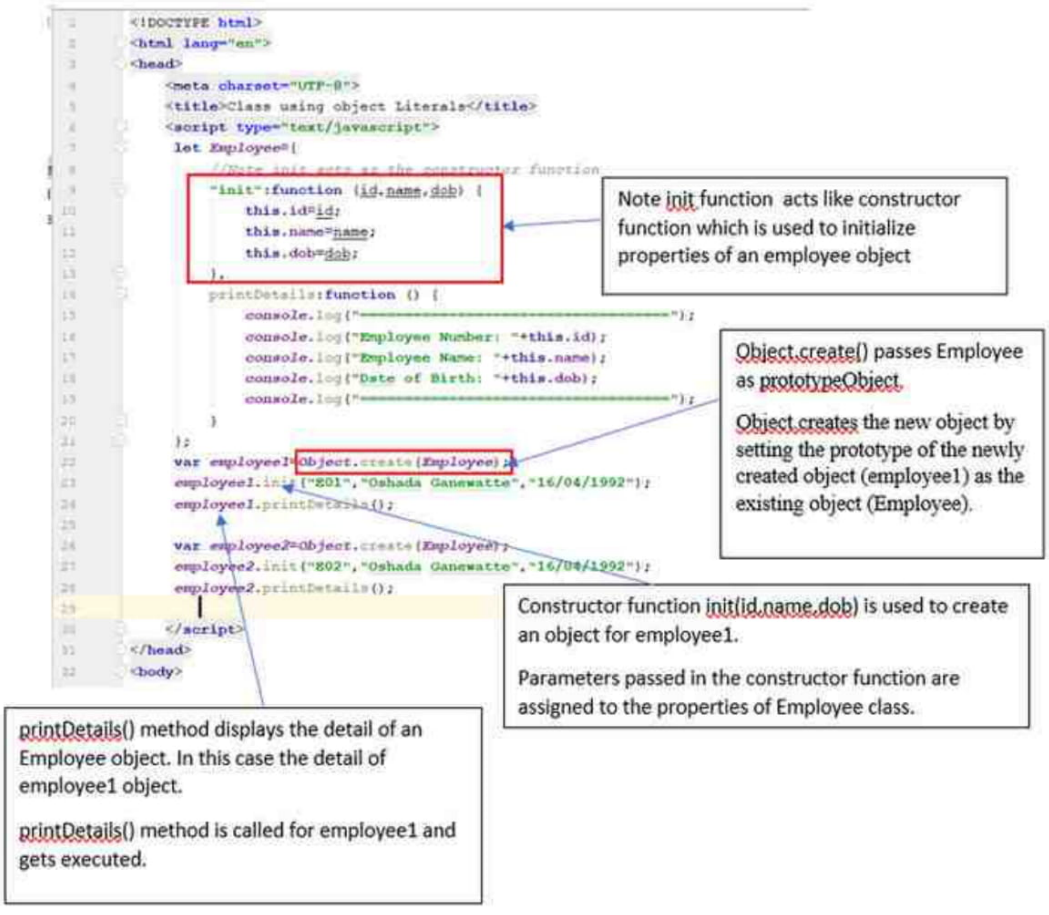
prototypeObject : Object which is going to be prototype of the newly created object.

2

.

propertyObject :

This parameter is optional. It is the property of the newly created Object



3) Callbacks

Callback is basically

a function that is to be executed in another function where its

block scope

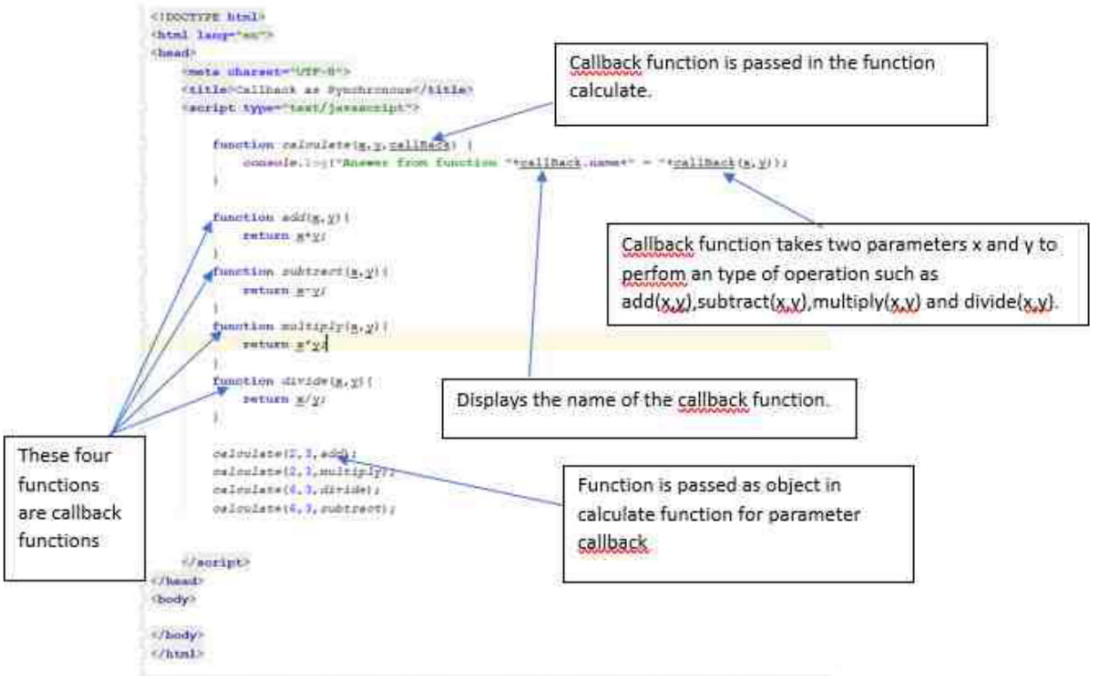
contains

callback method call. The callback functions is passed as a parameter for the another function so the

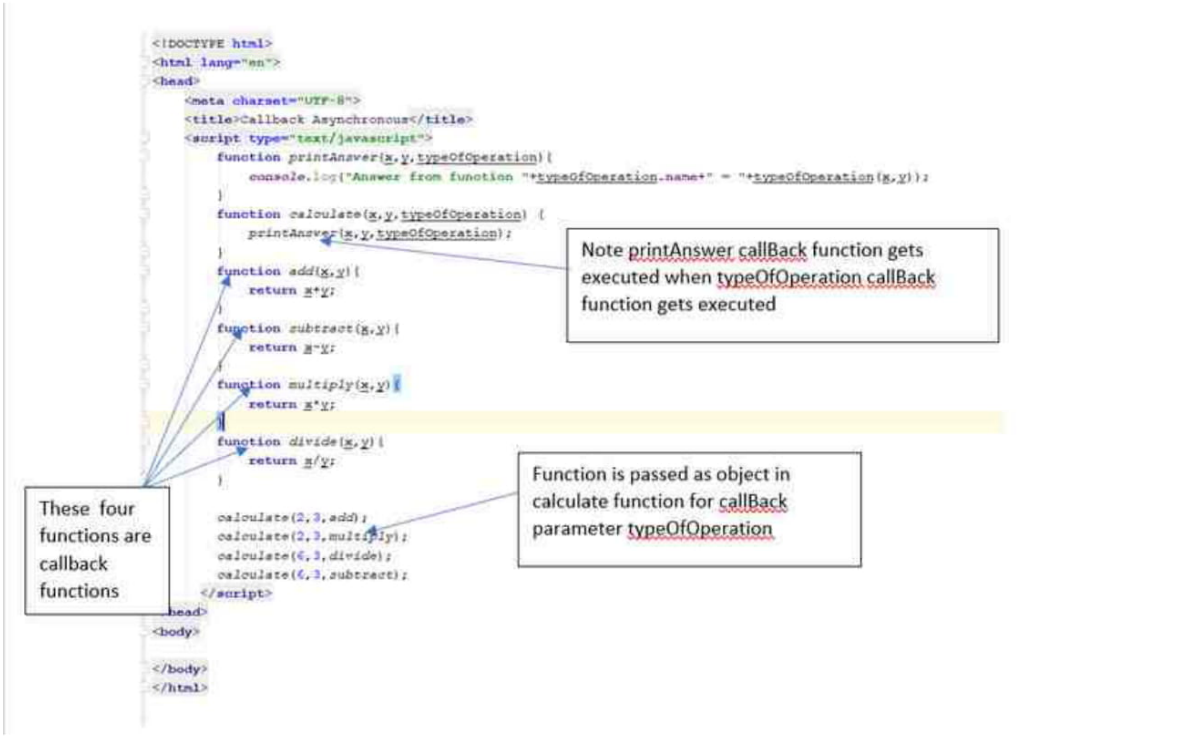
JavaScript knows what needs to be executed when the callback function is executed.

Callbacks can be used as asynchronous or synchronous.

Example of synchronous callback



Example of asynchronous callback



Note there are two callback functions in asynchronous example:

1. typeOfOperation
2. printAnswer

4) Promises

Promises are used to perform asynchronous operations in JavaScript. Promises are easier to manage

multiple asynchronous unlike callbacks which create callback hell(nested Callbacks).Since promises

manages multiple asynchronous operations, the code is readable and less complex unlike callbacks.

Further error handling is easier when using promises.

Promise has two internal properties:

1.

state:Behaviour of a promise in its response to series of events in its lifecyle.

2.

result: A value of your choosing. Initially it is undefined.

There are four types of state:

1)fulfilled: The state when a promise has been resolved

2)rejected: The state when a promise has been rejected.

3)pending :Initial state of a promise.

4)settled: The state when a promise that has been rejected or fulfilled.

There are two types of result:

1.

value

2.

error

We used two methods of promise:

1. then()
2. catch()

then()

then() is invoked when a promise is either resolved or rejected.

Parameters:

then() takes two functions as parameters

1.

Function which is executed when a promise is resolved and result is set to value.

2.

Function which is executed when a promise is rejected and result is set to error(Second function is optional since there is alternative method called catch() which can handle errors).

catch()

catch() is invoked when a promise is rejected or an error has occurred during execution.

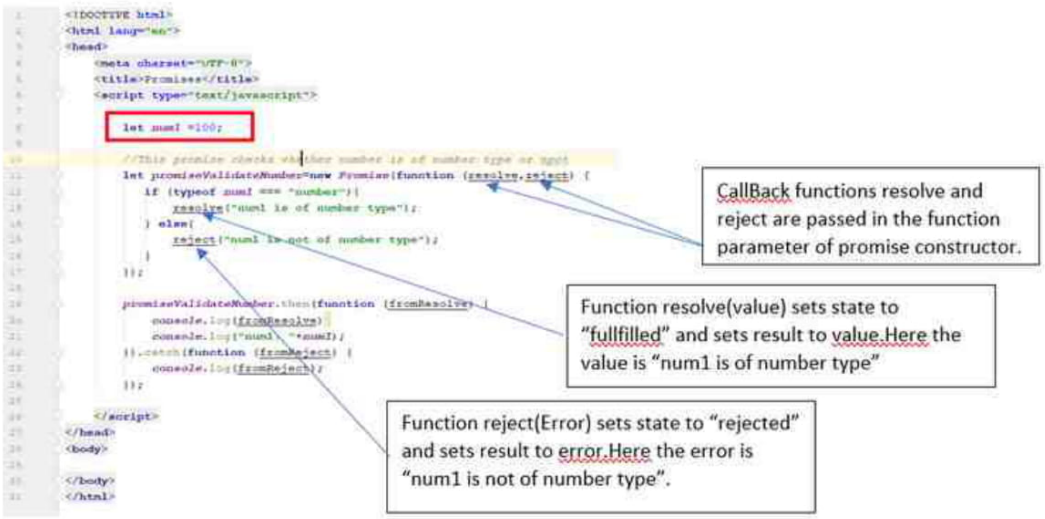
Parameters:

catch() only takes one function as a parameter

.

Function that handle errors or rejections of a promise

Example of single promise



Note the example single promise is used to validate num1 is of number type or not

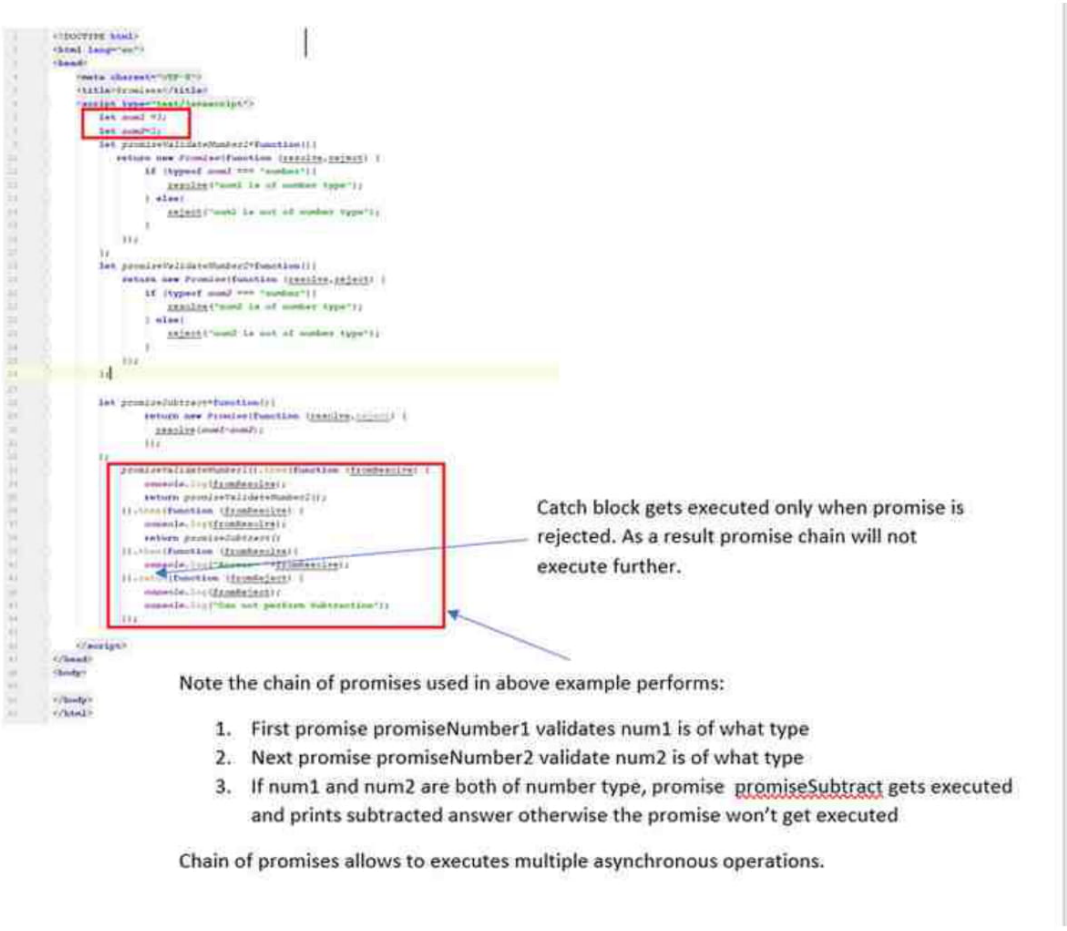
.

If num1===”number” is true then resolve function gets executed.

.

If num1===”number” is false, then reject function is executed.

Example of chain of promises



Note the chain of promises used in above example performs:

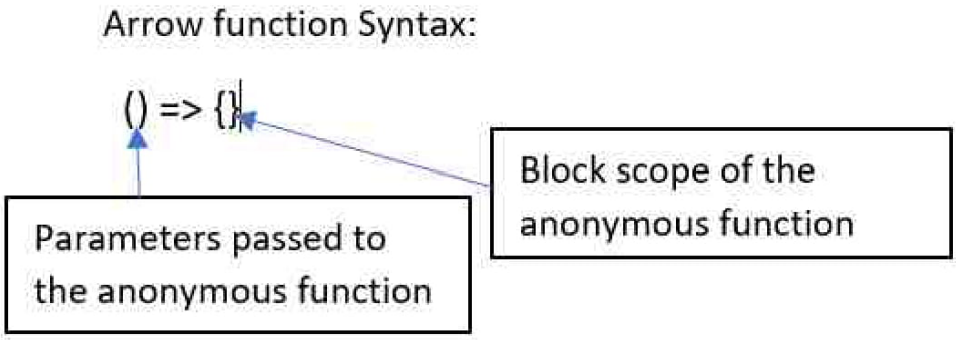
1. First promise promiseNumber1 validates num1 is of what type
2. Next promise promiseNumber2 validate num2 is of what type
3. If num1 and num2 are both of number type, promise promiseSubtract gets executed and prints subtracted answer otherwise the promise won't get executed

Chain of promises allows to executes multiple asynchronous operations.

5) Arrow functions

Arrow function was introduced in ES6/ECMAScript 2015.This is an alternative declaration to

declaration of anonymous function with function keyword.



If you have multiple statement defined for a block scope of a function, square brackets " { }" are

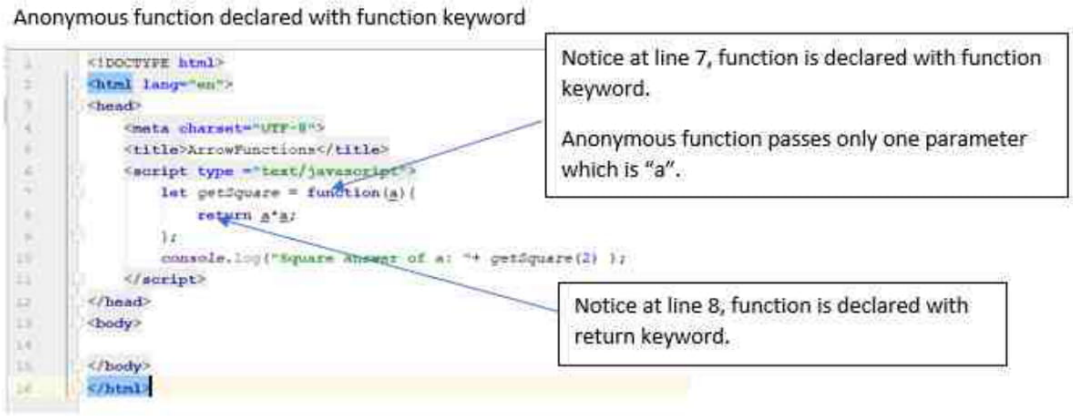
compulsory. Otherwise for a single statement, it is not necessary to use square brackets.

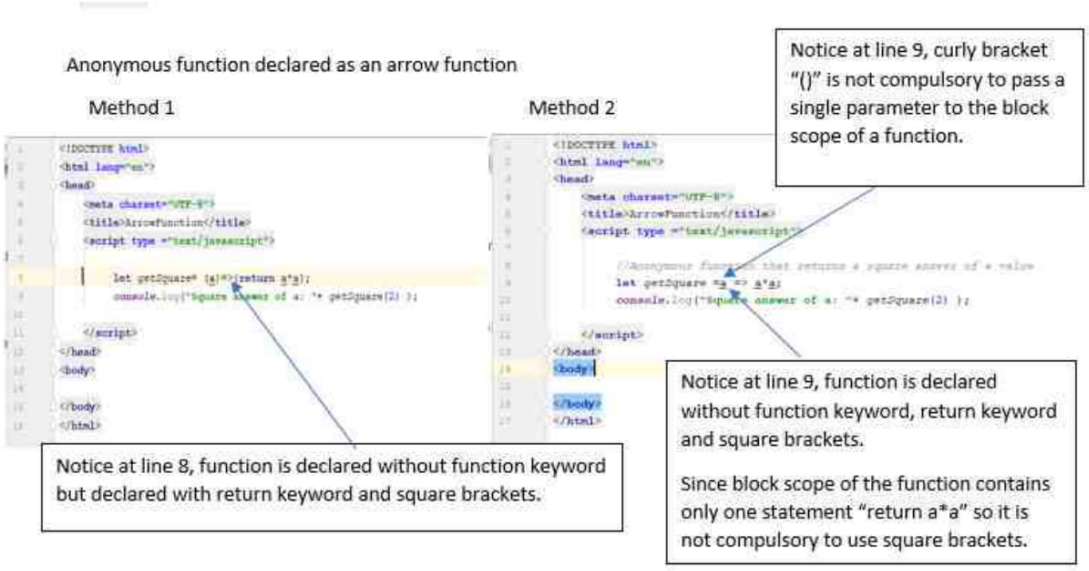
Below example compares the declaration of a anonymous function with function keyword and

declaration of a anonymous function as an arrow function.

1)

Single Parameter

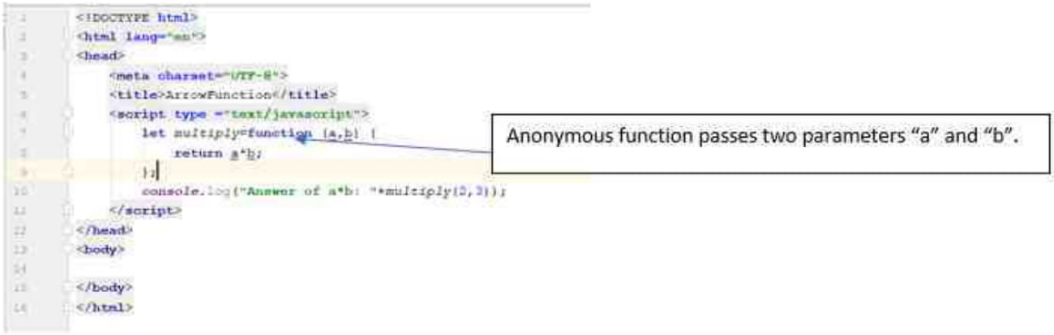




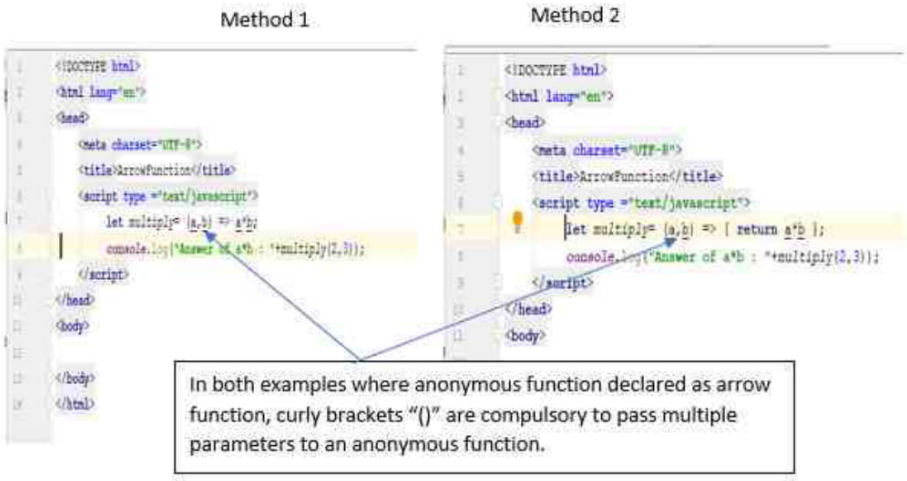
2)

Multiple Parameters

Anonymous function declared with function keyword

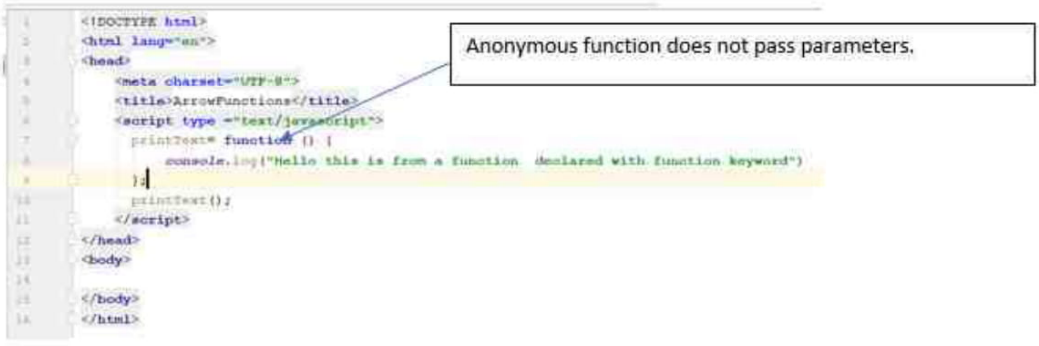


Anonymous function declared as an arrow function



3) No Parameter

Anonymous function declared with function keyword



Anonymous function declared as an arrow function

Method 1

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>ArrowFunctions</title>
    <script type ="text/javascript">
      printText= ()=>console.log("Hello this is from a arrow function");
      printText();
    </script>
</head>
<body>

</body>
</html>
```

Method 2

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>ArrowFunctions</title>
    <script type ="text/javascript">
      printText= ()=>{console.log("Hello this is from a arrow function")};
      printText();
    </script>
</head>
<body>

</body>
</html>
```

In both examples where anonymous function declared as arrow function, curly brackets "()" are compulsory when not passing a parameter or parameters to an anonymous function.

Lecture 3: React JS Features (2019-04-19 13:49)

Lecture 3: React JS Features

Introduction

.

React JS uses **component based architecture**.

.

JavaScript library for **building user interfaces.**

.

React **divides user interfaces** into **components**.

.

React **reduces boiler plate**. Boiler plate is where piece of our code is included in different sections of our project(high coupling).

Let us look at:

1.

npm commands to setup a React appliciation

2.

package.json

3.

Class components and Functional Components

4.

state and props

5.

Import and export modules

1)

npm commands to setup a React appliciation

.

npm init

Command that creates package.json file.

.

npm install parcel-bundler –save-dev

Command that installs parcel and babel related dependencies into developer environment.

.

npm install react react-dom prop-types - -save-dev

Command that installs react related dependencies into developer environment.

2) package.json

Document (type of NoSql Database) to get metadata of a package of the react application that is created from npm registry.

These the important fields in package.json:

1.

name

Sets the name of the package.

Rules

.

The name of the package can only contain lowcase letters, hypens (-) or underscores(_).

.

Length of the name must be less than 214 characters.

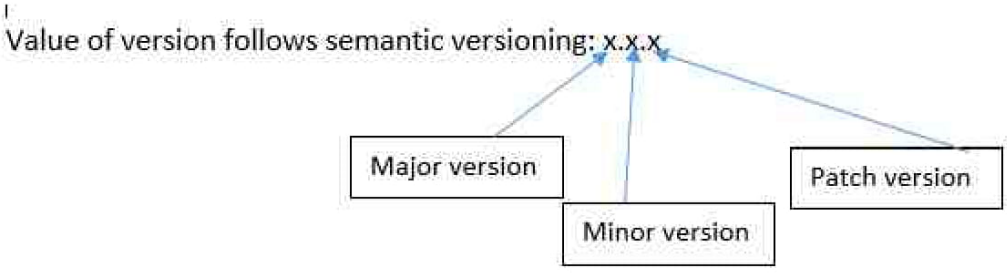
.

The name of the package cannot contain spaces .

2.

version

Indicates the current version of the package.



Major version : When you make API (Application Programming interface) changes that are incompatible.

Minor version : When you add functionality in backwards-compatible manner.

Patch version : When you make bug fixes which are backwards-compatible

Rules in using Semantic versioning

.

: This notation is used when you update patch releases.

Eg: 0.12.0, 0.12.1 is valid version but not 0.15.0

.

^ :This notation is when you want update patch and minor release

Eg: ^0.12.0, 0.12.1 and 0.15.0 are valid versions

.

* : Notation that indicates the package all updates including major version upgrades.

.

> : Accepting any version which is higher than the one you mention.

.

> = : Accepting any version which is equal to or higher than the one you mention.

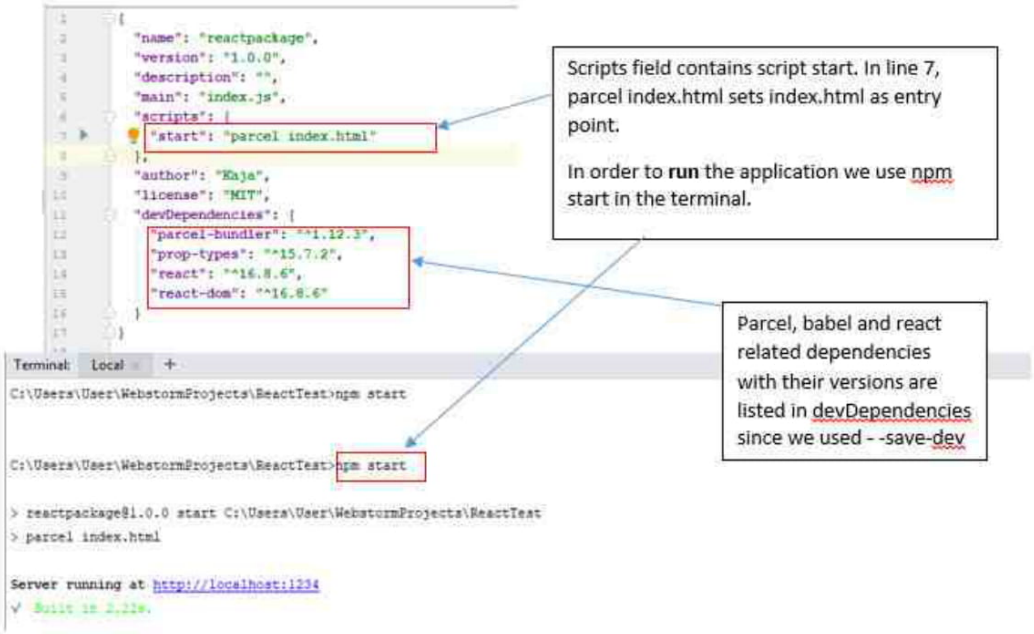
.

< : Accepting any version which is lower than the one you mention.

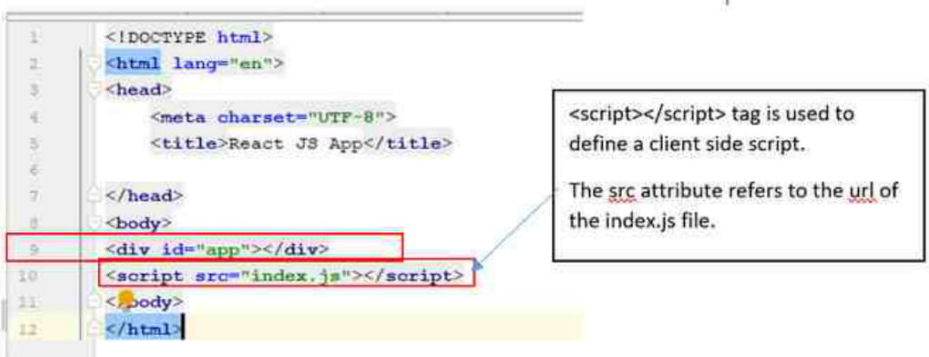
.

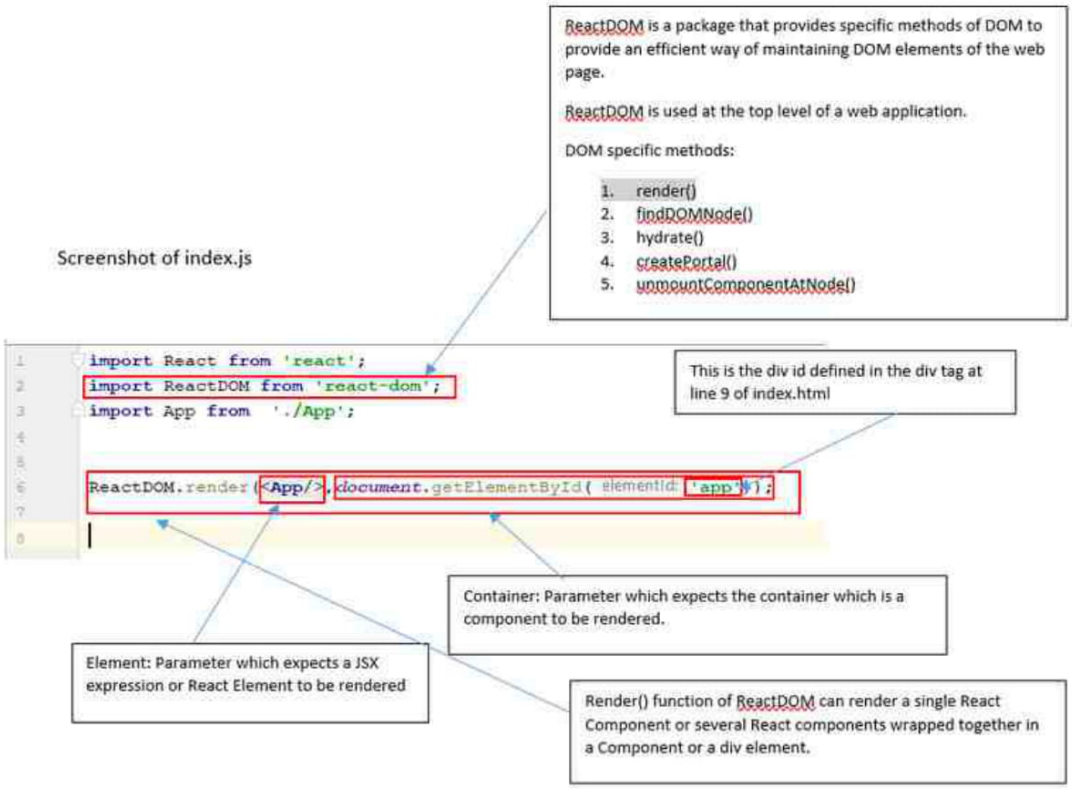
< =: Accepting any version which is equal to or lower than the one you mention.

Example of package.json



Screenshot of index.html





3) Class components and Functional Components

Functional Components

.

Function that returns a value in JSX format.

.

They are stateless components.

.

Lifecycle methods cannot be used such as render(),componentWillMount(),

.

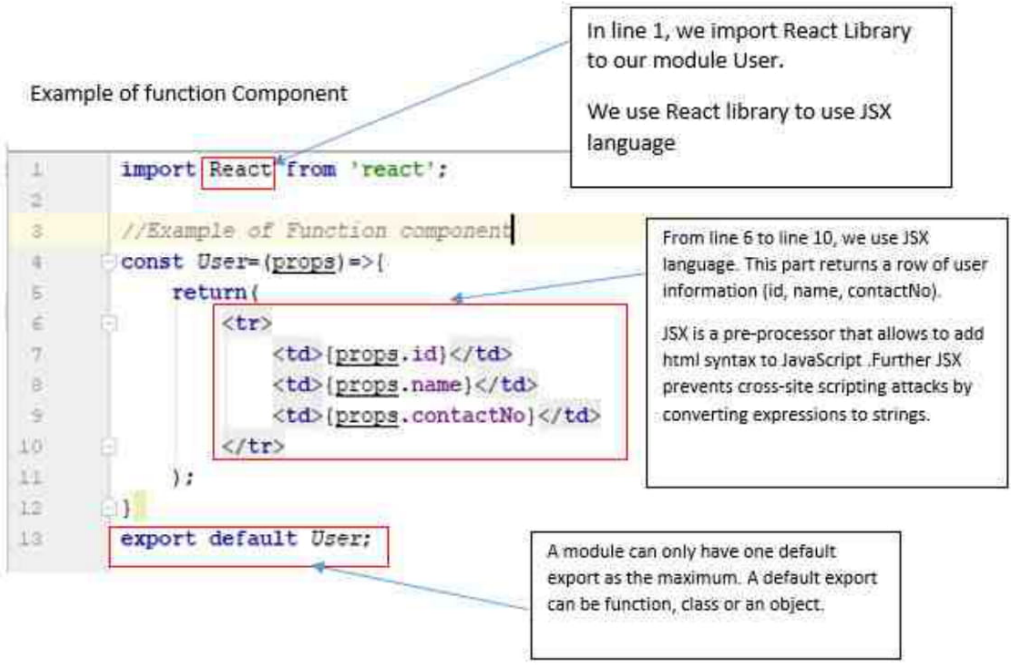
componentShouldUpdate() and etc.

.

Keyword

this

cannot be used when using Functional Components.



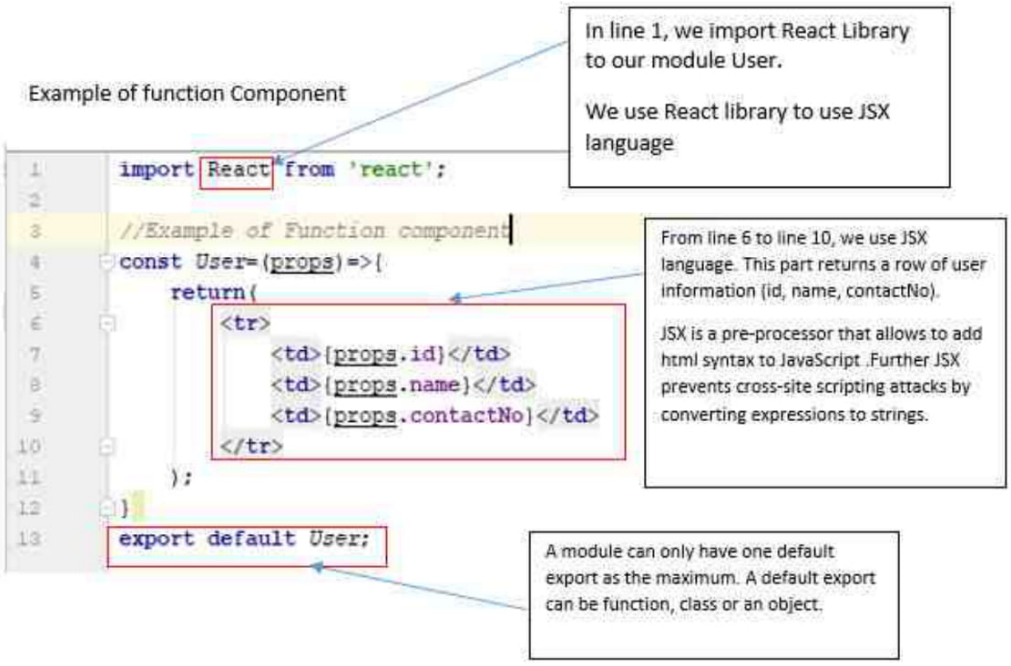
Benefits of using Functional components

.

Code becomes simple, readable and usable

.

Easy to test function components as they pure functions.



Class Components

.

Uses ES6 class Syntax.

.

They are stateful components.

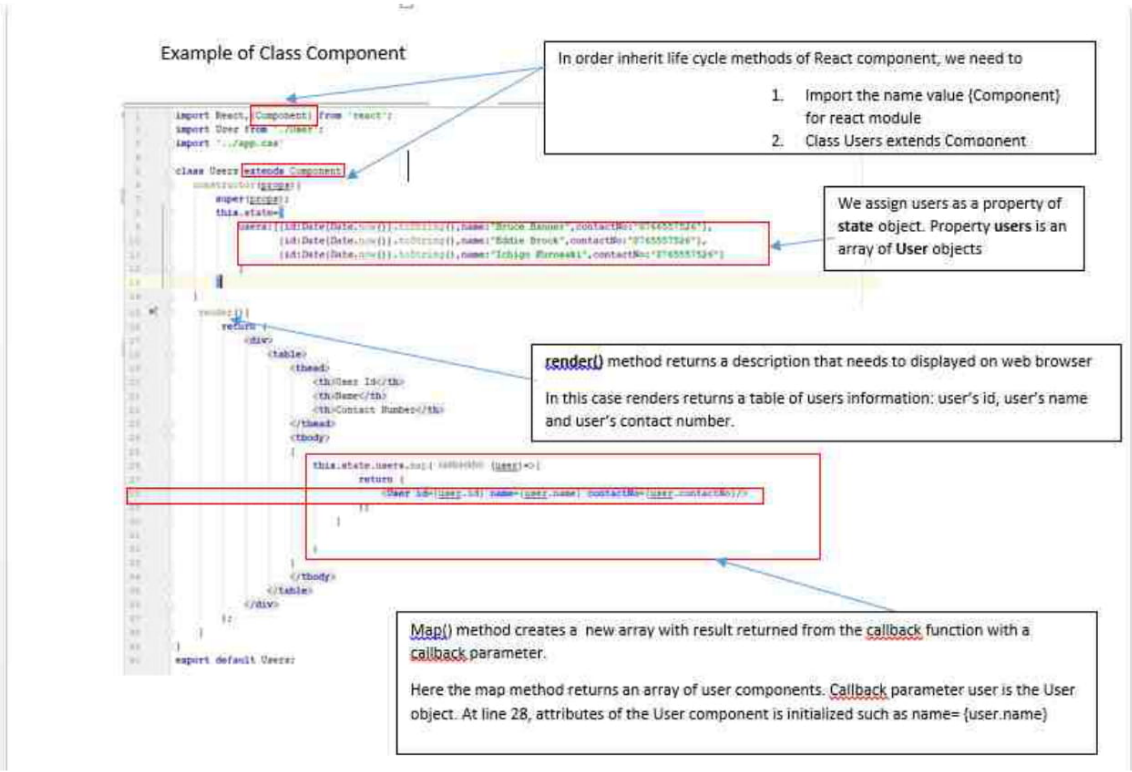
.

Class components can use life cycle methods such as constructor(),render() and

componentWillMount()

.

The class component extends *React.Component* or *Component.*



4) state and props

Props

.

Props is an object that pass data and methods from parent component to child component.

.

Props are immutable or read only.

.

Props can be used in both function components and class components.

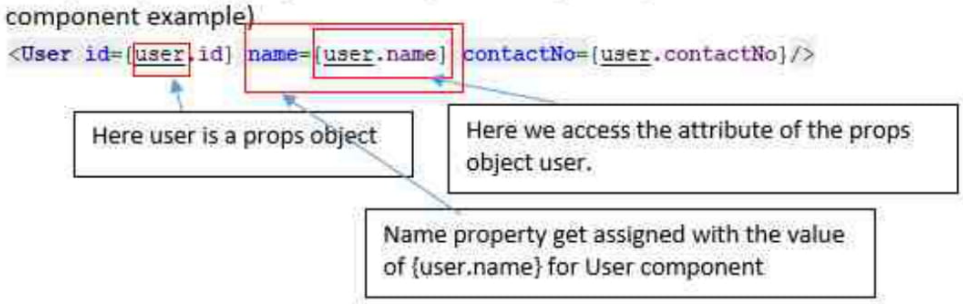
.

Props can be passed to any component as attributes are declared in the html tag of the component.

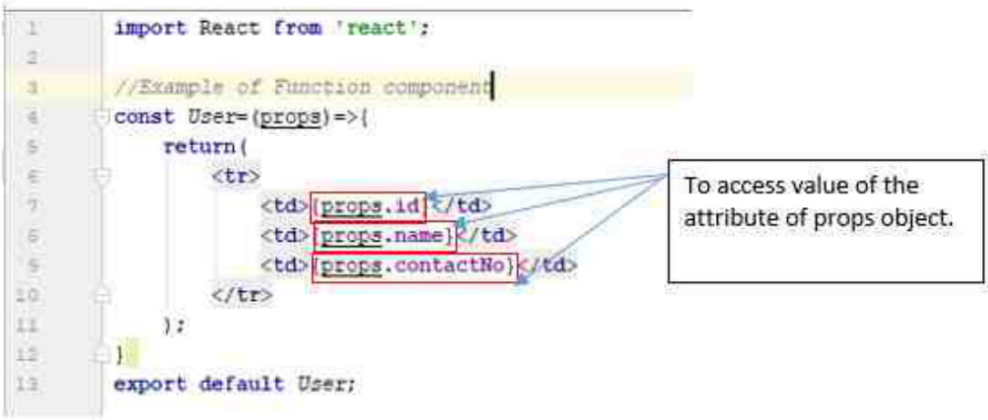
.

Value of attribute of props object can be accessed from a component.

Example of passing props in html tag of a component (Line 28 from class component example)



Example of accessing an attribute of prop in a component (Example from function component)



State

.

State is an object that keeps data which can be updated over time and controls the behaviour of a component after each update.

.

State can only be used in class Components.

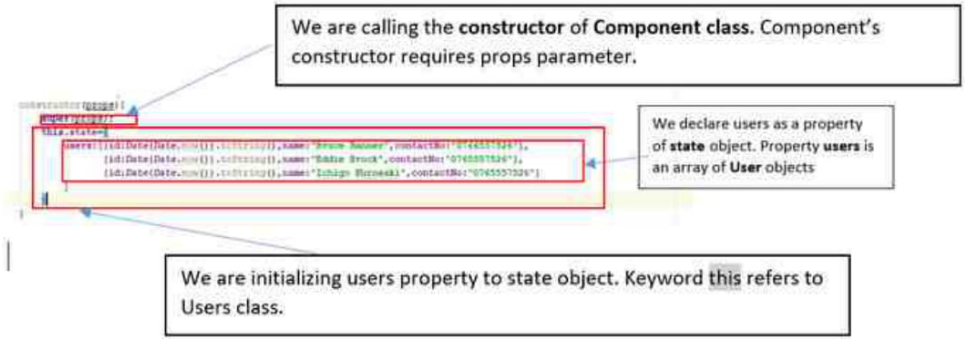
.

State are mutable.

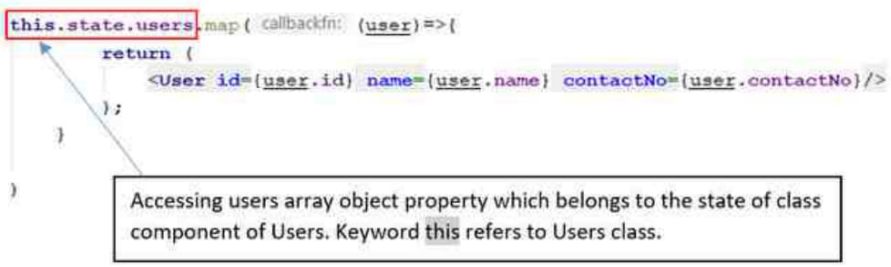
.

When updating a state,use setState({ }) method.

Example of initializing state



Example of performing an action by accessing property of state



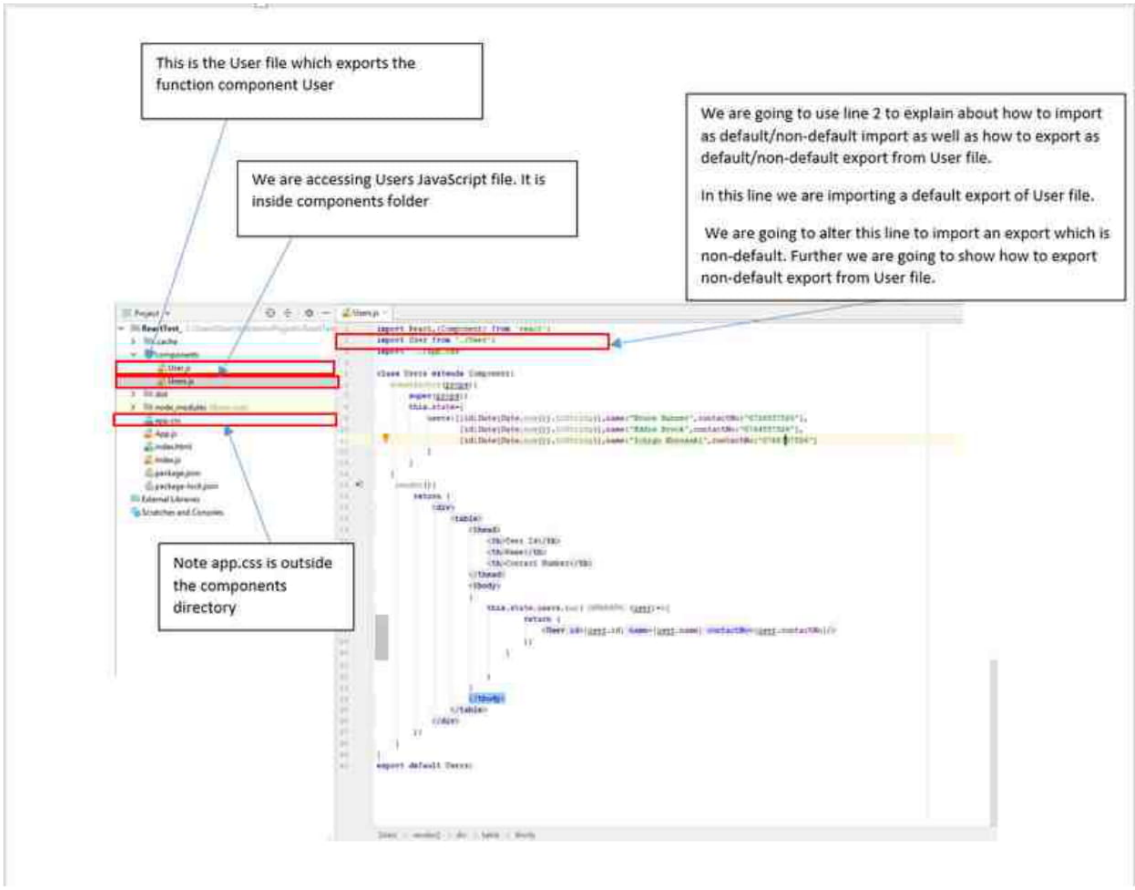
5) Import and Export Modules

In Node JS we use
'require'
to import and
'module.exports'
to export.

We use these statements for React JS when using ES5 syntax. When ES6 syntax is

introduced we use
'import'
and
'export'
statements as replacement in React JS.

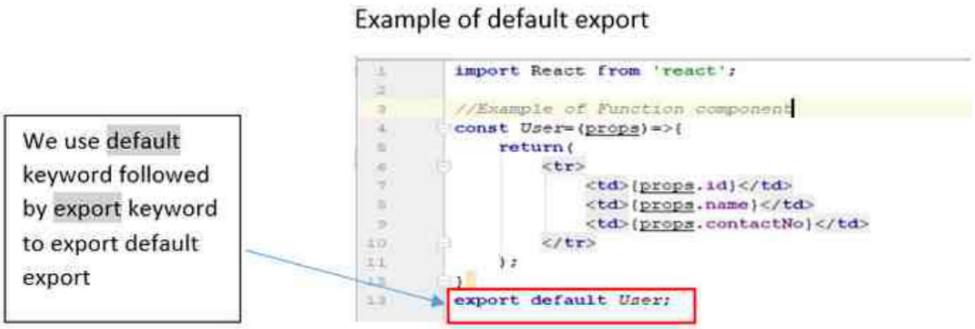
Let us consider Users file code with its project structure



Default export:

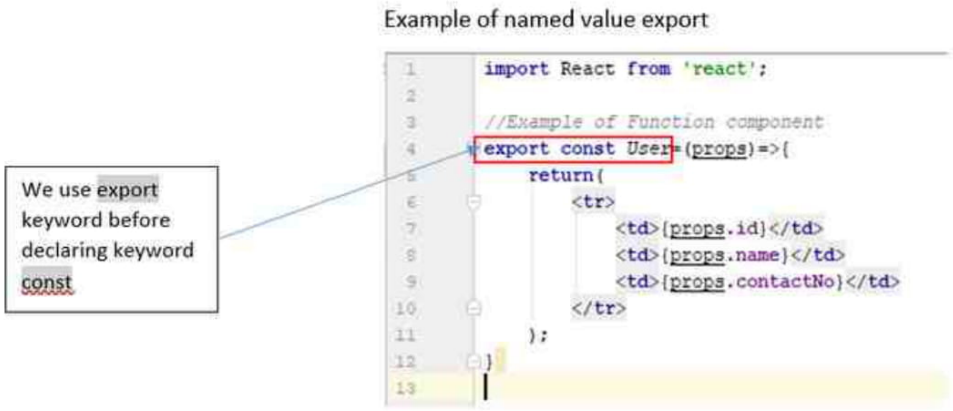
There can be only one default export for a file. A default export. Default export can be function, class or an object.

Example of default export



Named Value Export: We can export multiple name values from a single file to the file that

requires the export.



Rules when importing files from our project

When you want to import the module from another file into the file which requires the imported module:

.

If the module that needs to be imported is within the current directory, we use "./"

```
import User from './User';
```

.

If the module that that needs to be imported is in the previous directory, we use "../"

```
import '../app.css'
```

.

When importing a default export, we do need to use default brackets.

```
import User from './User';
```

.

When importing a named value export ,it is compulsory to use name brackets.

```
import {User} from './User';
```

Lecture 4: Mongo DB (2019-05-26 23:43)

Lecture 4: Mongo DB

First let us understand what is NOSQL database. NOSQL database is a class of database management systems that does not follow all the rules of relational database DBMS (Database Management System) as well as can not use SQL to query data.

There are 4 types of NOSQL database:

1)

Key value : Data is store as key-value pairs

2)

Document : Data is stored as documents which can be of t document ype such as JSON, BSON and XML in maps or collections

3) Column Family : Data is stored in column families as rows are associated with many

columns.

4) Graph : Stores entities and relationships between them and represent it in a graph.

Node represents an entity while edge represents a relationship.

Now let us understand about Mongo DB

- Mongo DB is NOSQL document database.
- Mongo DB uses SpiderMonkey JavaScript Engine.
- Further it uses strong query capabilities with aggregations using JavaScript.
- Furthermore it has an in built storage called **Grid File System.** Refer [1]https://docs.mongodb.com/manual/core/gridfs/ to understand Gride File System.
- Mongo DB uses collections to store documents. Collection in NoSQL is similar to tables in RDBMS.

There are two ways to use Mongo DB

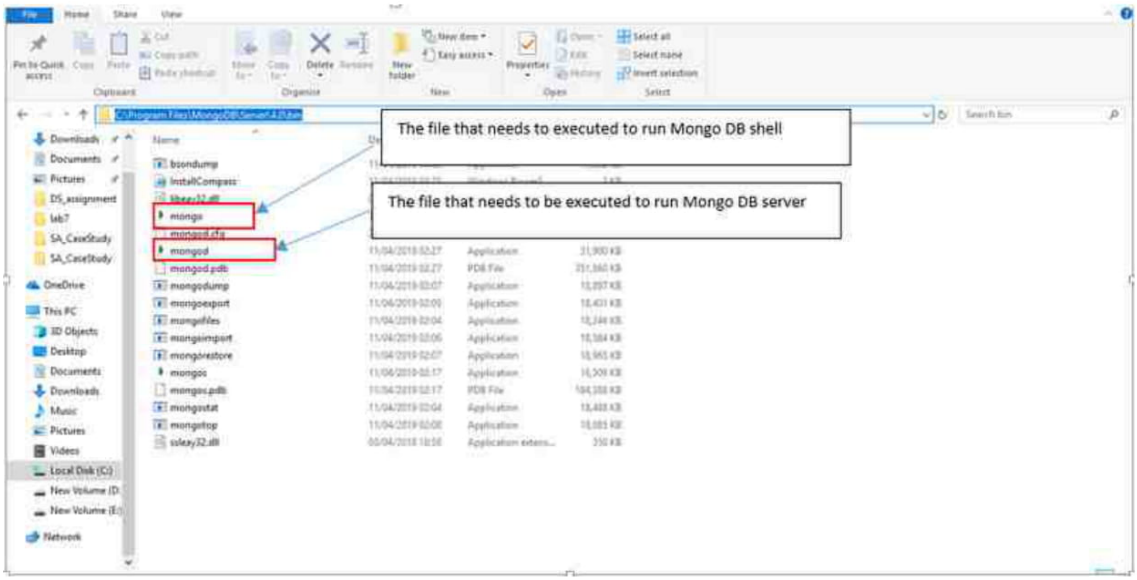
1) Command Line interface such as Mongo DB shell

2) Graphical user interface tool such Mongo community compass

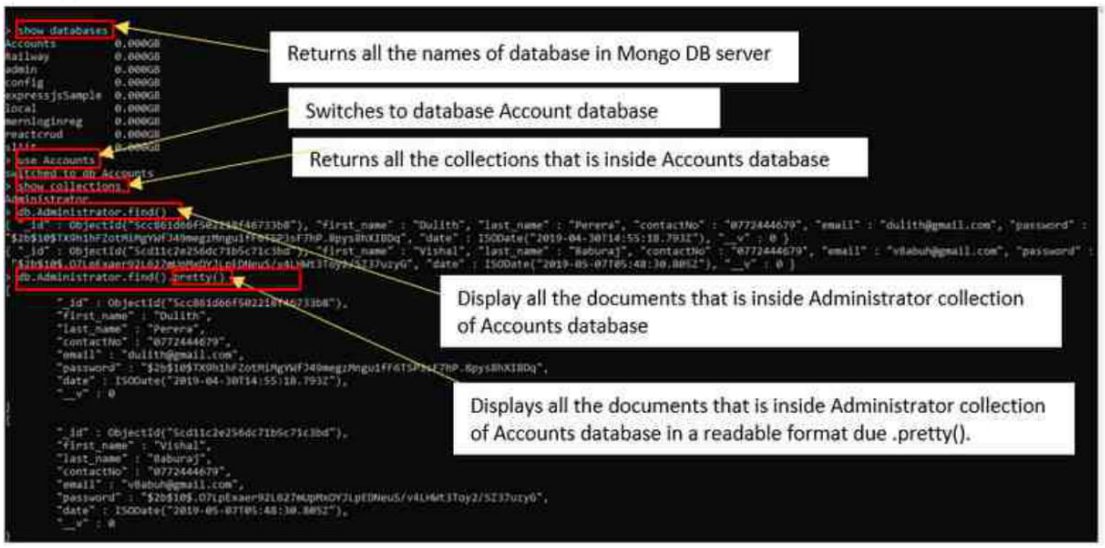
1) Mongo DB Shell

To run Mongo DB shell

1) Double click on mongod file to start the server
2) Double click on mongo file to start the shell



Mongo Shell



Mongo shell Command Helpers

1) help : Displays help.

2) db.help() : Displays help for database methods.

3) show dbs : Prints a list of database which is on the server.

4) use <db> : Switch current database to <db>. The mongo shell variable db is set to the current database.

5) show collections: List of collections for the current database is printed.

6) show databases : List of all available databases are printed

Mongo shell Javascript Operations

1) db.collection.find():

S

earch for a document according to a query

R

efer

[2]https://docs.mongodb.com/manual/reference/method/db.collection.find/

Eg:

```
db.student.find({"name":"John"})
```

2) db.collection.insert() : Inserts document or documents to a collection

Refer

[3]https://docs.mongodb.com/manual/reference/method/db.collection.insert/

Eg:

```
db.student.insert({"name":"John","dateOfBirth":"1990-01-01T00:00:00z","subjects":["Application Frameworks","Computer Architecture"]})
```

3)

db.collection.update()

: Updates a specific documents in a collection

Refer

[4]https://docs.mongodb.com/manual/reference/method/db.collection.update/

Eg:

```
db.student.update({"name":"Smith",isActive:true},{$push:{"subjects":"Distributed Computing"}})
```

The $push operator appends a specified value to an array.

4) db.collection.remove() : Removes a document

Refer [5]https://docs.mongodb.com/manual/reference/method/db.collection.remove/

Eg:

```
db.student.remove({"name":"John"})
```

5) cursor.pretty() : Formats the document into a readable format

Refer

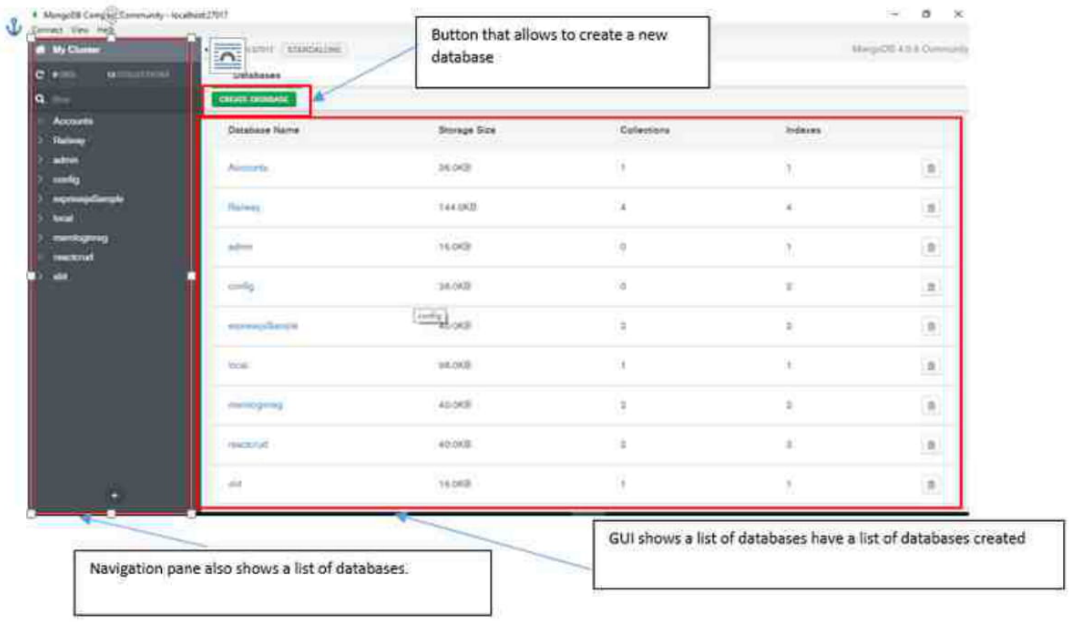
[6]https://docs.mongodb.com/manual/reference/method/cursor.pretty/

Eg:

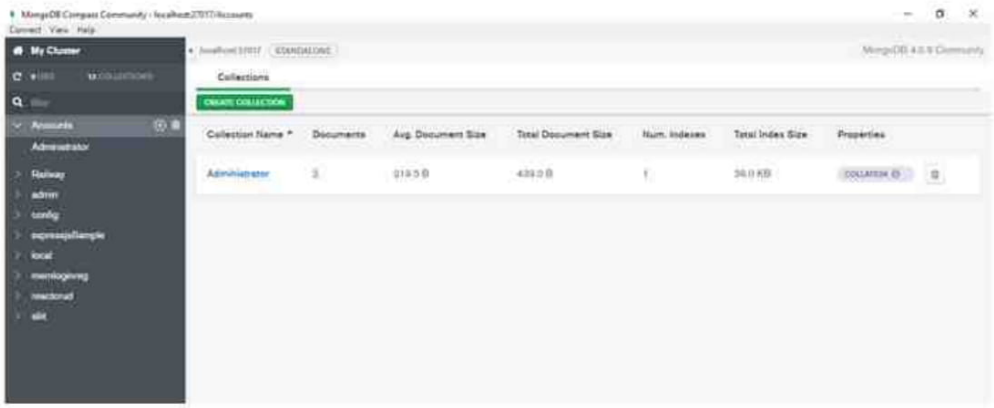
```
db.student.find().pretty()
```

2) Mongo DB Compass Community

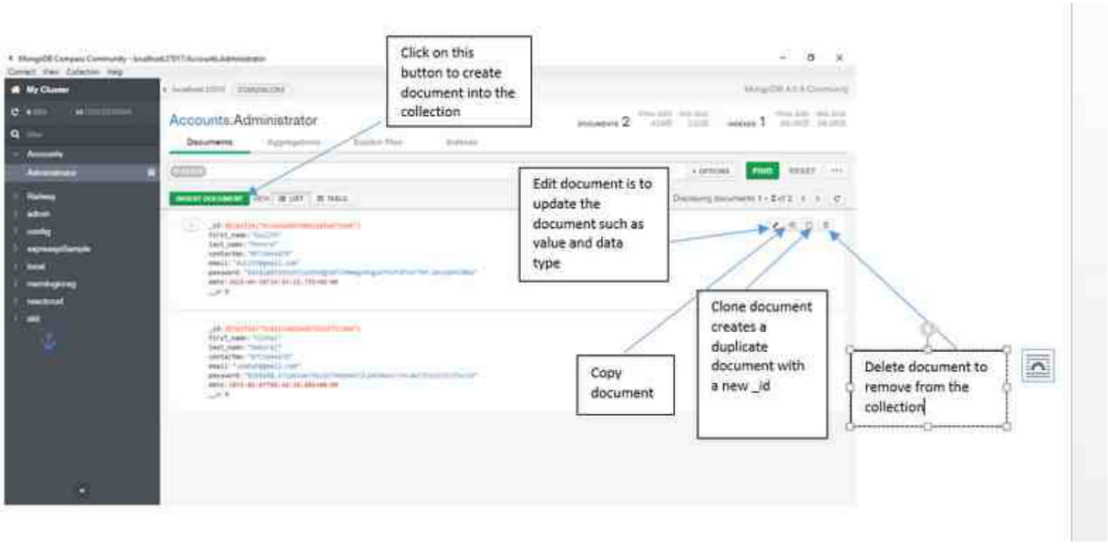
MongoDB Compass community is GUI tool allows us to do db queries without the need of query commands.



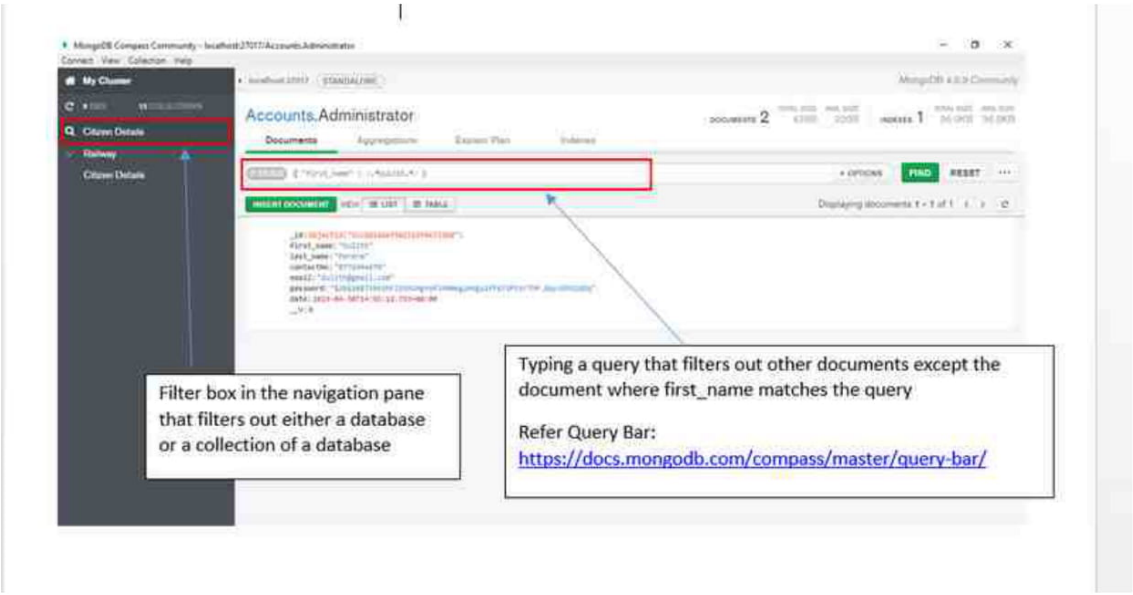
When you click on a database it will display a list of collections



When you click on Administrator link from previous image you go to collections page



This screenshot explains how to filter produces result similar to find in Mongo shell



1. `https://docs.mongodb.com/manual/core/gridfs/`
2. `https://docs.mongodb.com/manual/reference/method/db.collection.find/`
3. `https://docs.mongodb.com/manual/reference/method/db.collection.insert/`
4. `https://docs.mongodb.com/manual/reference/method/db.collection.update/`
5. `https://docs.mongodb.com/manual/reference/method/db.collection.remove/`
6. `https://docs.mongodb.com/manual/reference/method/cursor.pretty/`

Lecture 5:Node js (2019-05-28 15:21)

Lecture 5: Node js

Introduction

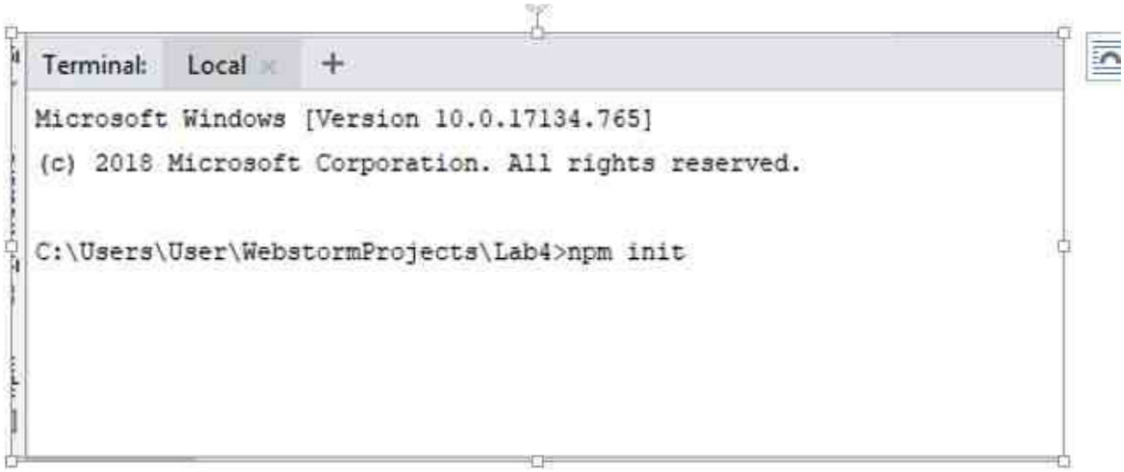
- Node js is open source as well as cross platform run time environment.
- It is used for server side and networking applications (Back end).
- It uses non-blocking an event driven model which makes it asynchronous as it does not wait for i/o operations to completed.

In this post we are going to learn

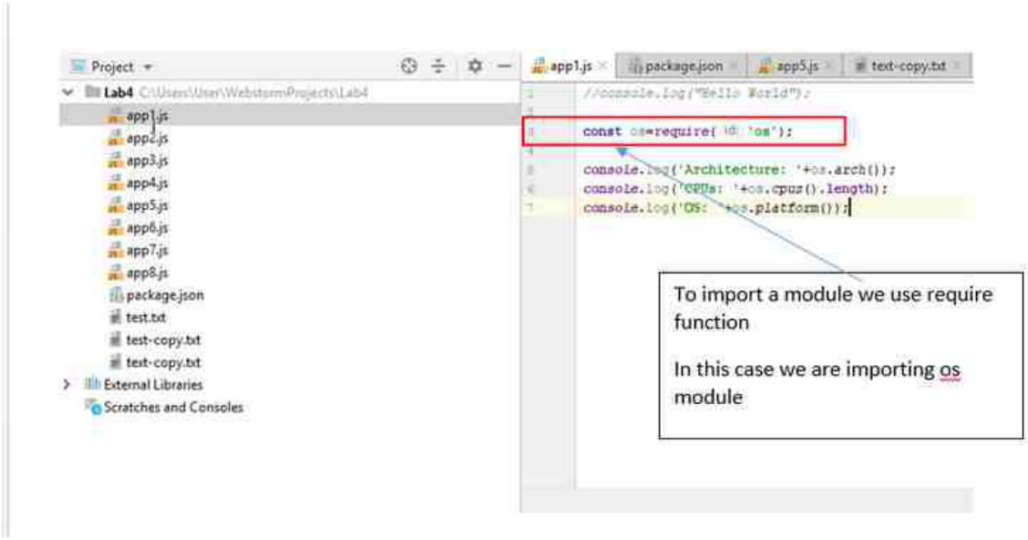
1. Node js tutorial
2. require and module.exports
3. Module os
4. Module http

1)Node js Tutorial

1) First you need to create a package .json. This can be done by typing npm init in the terminal where the path is project directory.



2) Create a node js file



3) Run a node js file using command node < node js file> in command prompt.

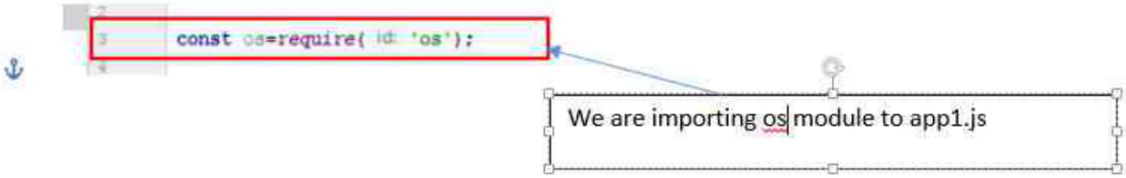
In this case

node js file

is app1.js

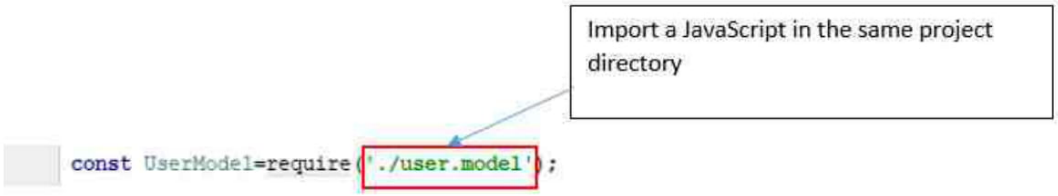
2) require and module.exports

require() is a function that is used to import or consume a module into node.js file After we import the module, we can use any function that is related to that module.For instance we can use functions that are related to os module.



module.exports are instructions that tell Node.js which bits of code thats needs to export from a given file so other files are allowed to access the exported code.

To import javaScript from module.exports we use require()



3) Module os

The

os

module provides operating system-related utility methods

This is screenshot printing outputs from some of the

os related utility methods.

```
1  //console.log("Hello World");
2
3  const os=require( id: 'os');
4
5  console.log('Architecture: '+os.arch());
6  console.log('CPUs: '+os.cpus().length);
7  console.log('OS: '+os.platform());
8  console.log("Host name: "+ os.hostname());
9  console.log("Total memory: "+os.totalmem());
10 console.log("Free memory: "+os.freemem());
```

Note we use these methods from os module

- os.arch() : Returns a string indicating CPU architecture of the operating system. In my computer it was x64.
- os.cpus(): Returns an array contains array of objects.Each object contains information about the core such as models,speed and times .

- os.platform() : Returns a String indicating platform of the operating system. In my computer it was win32.

- os.hostname(): Returns a String which is the hostname of the operating system. In my computer it was

  DESKTOP-68EBO3S

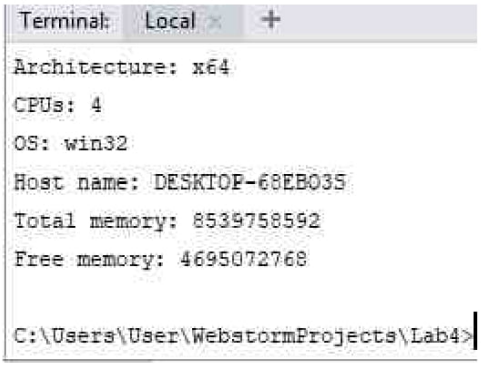
- os.totalmem() : Returns the total amount of system memory in bytes which is an Integer. In my computer it was 8539758592

- os.freemem():Return the amount of memory that is remaining as free in bytes which is an integer. In my computer it was 469507268

Notice in line 6 os.cpus().length returns a string about the length of the array

```
4    console.log('CPUs: '+os.cpus().length);
```

This is the output from the above node js file that uses the os module



3) Module http

Module required in order to use http client and server.

```
const http=require( id: 'http');

http.createServer( requestListener: (req,res)=>{
    res.setHeader( name: 'Content-type', value: 'text/html');

    switch (req.method) {
        case 'GET':
            res.write( chunk: '<h1>Hello World</h1>');
            res.end();
            break;

        case 'POST':
                req.on( event: 'data', listener: (data)=>{
                    res.write( chunk: '<h1>Hello '+data+'</h1>');
                    res.end();
                });
                break;
    }
}).listen( port: 3000, listeningListener: (err)=>{
    console.log('Server is listening to port 3000.');
});
```

Notice in the above node js file that uses http module uses two methods.

- http.createServer() : Returns an instance of server.It uses two parameters:

1) [options]

2) requestListener which is a callback function that

takes parameters request

and response to implement operations when invoking http methods such as POST and GET .

- server.listen() :

  Http server gets started when listening for connection s.

Further these two methods are used as method chaining.

Switch method is to executes the block operations depending on the

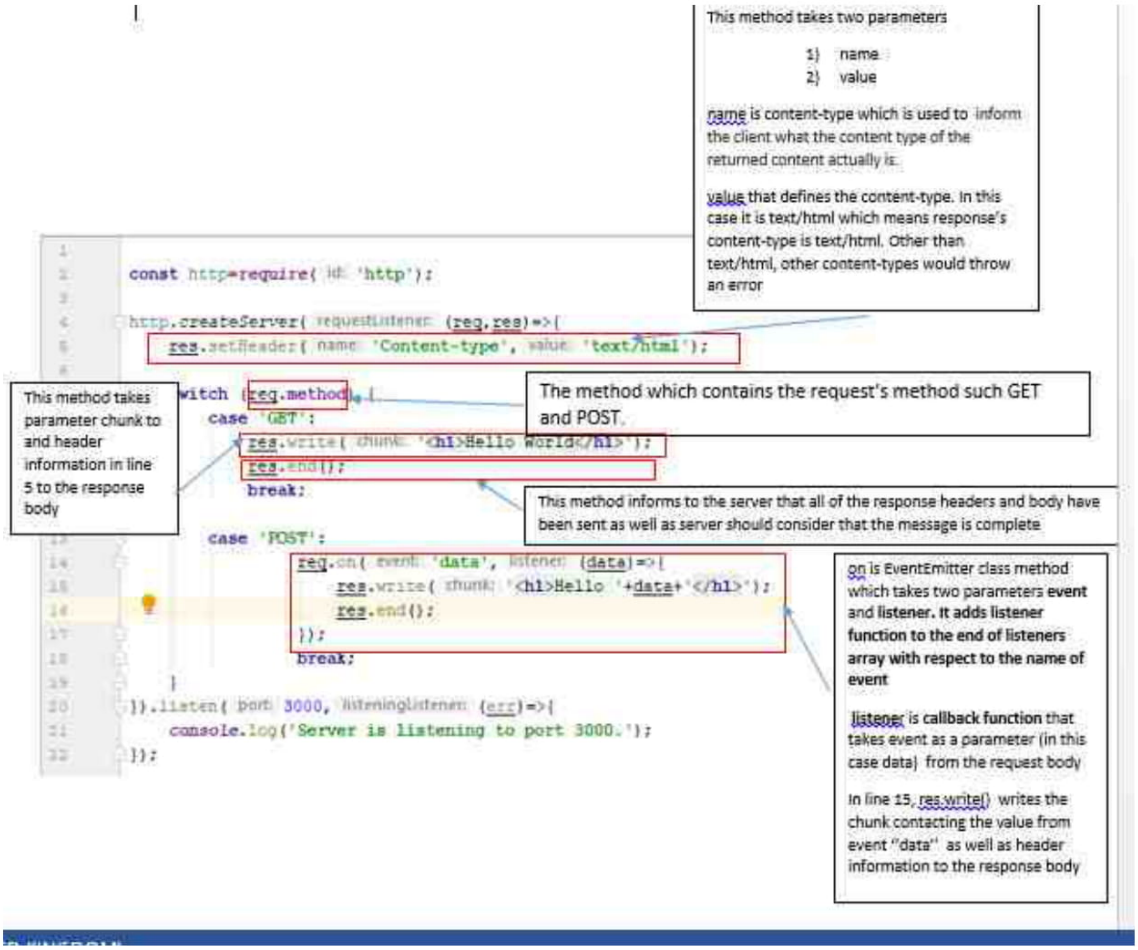
case

of

r

equest method. For instance if req.method is GET,it will execute methods according to case 'GET' .

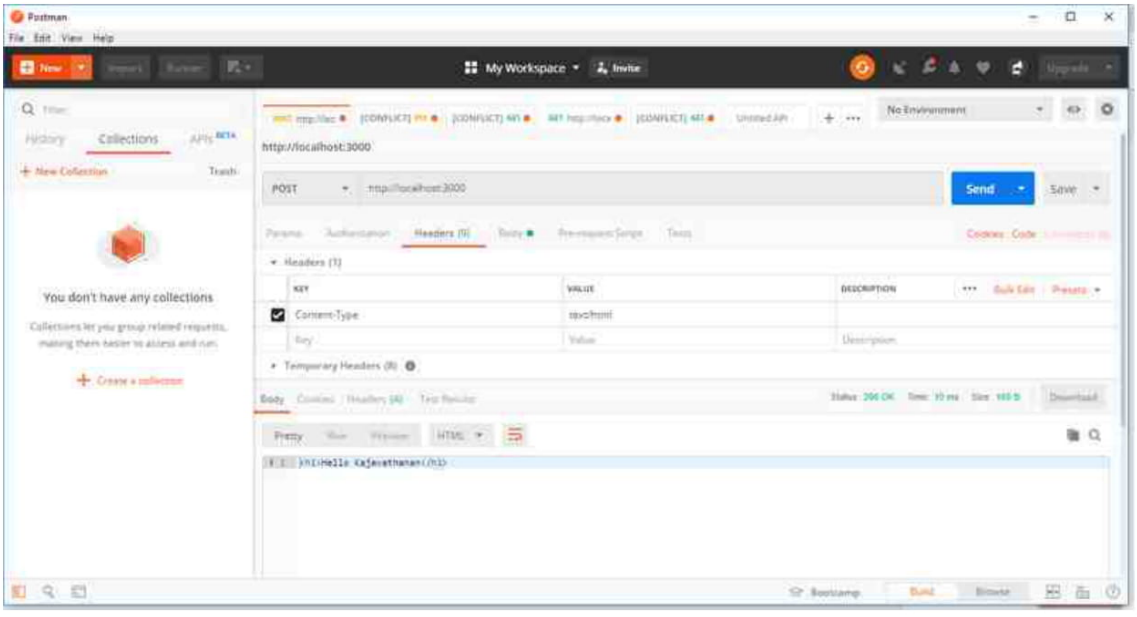
This screenshot explains methods used by req(request) and res(response)



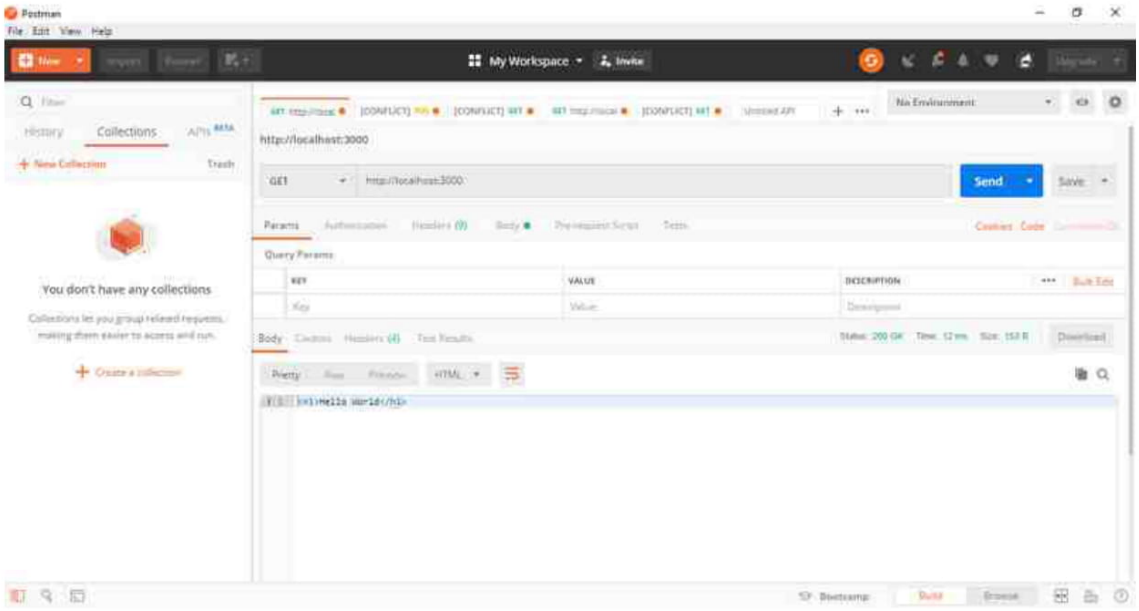
To test rest service we can use Postman or RestClient

The test services are tested in Postman

Set content-type as text/html in Headers tabs as shown in screenshot



Screenshot when you send get Request



Screenshot when you enter data in the body and send post Request