Unit-3
Angular JS & Node JS

Angular JS

Introduction

- **AngularJS** is a very powerful JavaScript Framework. It is used in Single Page Application (SPA) projects.
- It extends HTML DOM with additional attributes and makes it more responsive to user actions.
- AngularJS is open source, completely free, and used by thousands of developers around the world.
- It is licensed under the Apache license version 2.0.

Why to Learn AngularJS?

- AngularJS was originally developed in 2009 by Misko Hevery and Adam Abrons. It is now maintained by Google.
- AngularJS is a efficient framework that can create Rich Internet Applications (RIA).
- AngularJS provides developers an options to write client side applications using JavaScript in a clean Model View Controller (MVC) way.
- Applications written in AngularJS are cross-browser compliant.
- AngularJS automatically handles JavaScript code suitable for each browser.
- Overall, AngularJS is a framework to build large scale, high-performance, and easy-to-maintain web applications.

Core Features

- **Data-binding** − It is the automatic synchronization of data between model and view components.
- **Scope** − These are objects that refer to the model. They act as a glue between controller and view.
- **Controller** − These are JavaScript functions bound to a particular scope.
- **Services** − AngularJS comes with several built-in services such as $http to make a XMLHttpRequests. These are singleton objects which are instantiated only once in app.
- **Filters** − These select a subset of items from an array and returns a new array.

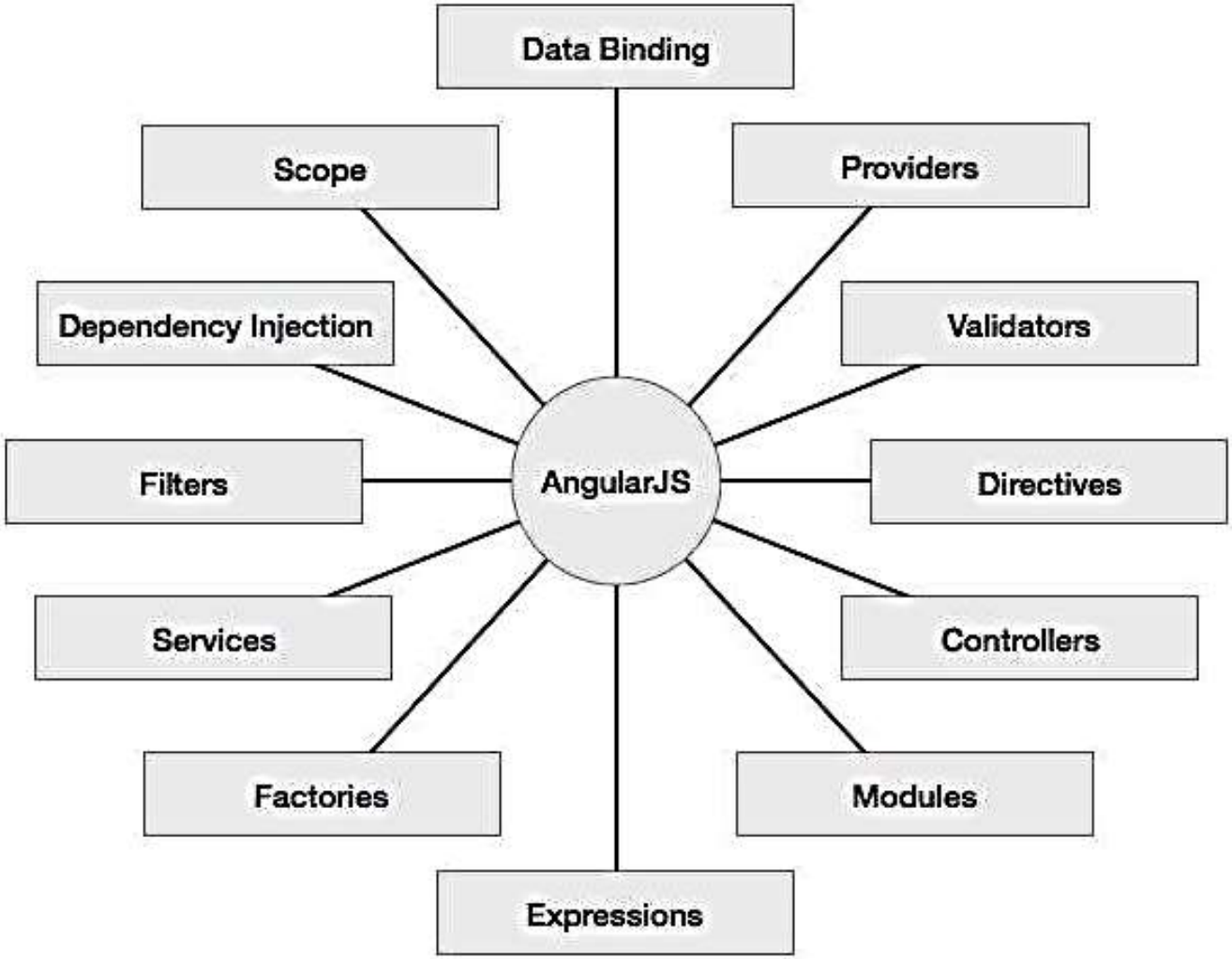
Core Features

- **Directives** – Directives are markers on DOM elements such as elements, attributes, css, and more. These can be used to create custom HTML tags that serve as new, custom widgets. AngularJS has built-in directives such as ngBind, ngModel, etc.
- **Templates** – These are the rendered view with information from the controller and model. These can be a single file (such as index.html) or multiple views in one page using *partials*.
- **Routing** – It is concept of switching views.

Core Features

- **Model View Whatever** − MVW is a design pattern for dividing an application into different parts called Model, View, and Controller, each with distinct responsibilities. AngularJS does not implement MVC in the traditional sense, but rather something closer to MVVM (Model-View-ViewModel).
- **Deep Linking** − Deep linking allows to encode the state of application in the URL so that it can be bookmarked. The application can then be restored from the URL to the same state.
- **Dependency Injection** − AngularJS has a built-in dependency injection subsystem that helps the developer to create, understand, and test the applications easily.

Concepts


Data Binding
Scope
Providers
Dependency Injection
Validators
Filters
AngularJS
Directives
Services
Controllers
Factories
Modules
Expressions

Advantages of AngularJS

- It provides the capability to create Single Page Application in a very clean and maintainable way.
- It provides data binding capability to HTML. Thus, it gives user a rich and responsive experience.
- AngularJS code is unit testable.
- AngularJS uses dependency injection and make use of separation of concerns.
- AngularJS provides reusable components.
- With AngularJS, the developers can achieve more functionality with short code.
- In AngularJS, views are pure html pages, and controllers written in JavaScript do the business processing.

Disadvantages of AngularJS

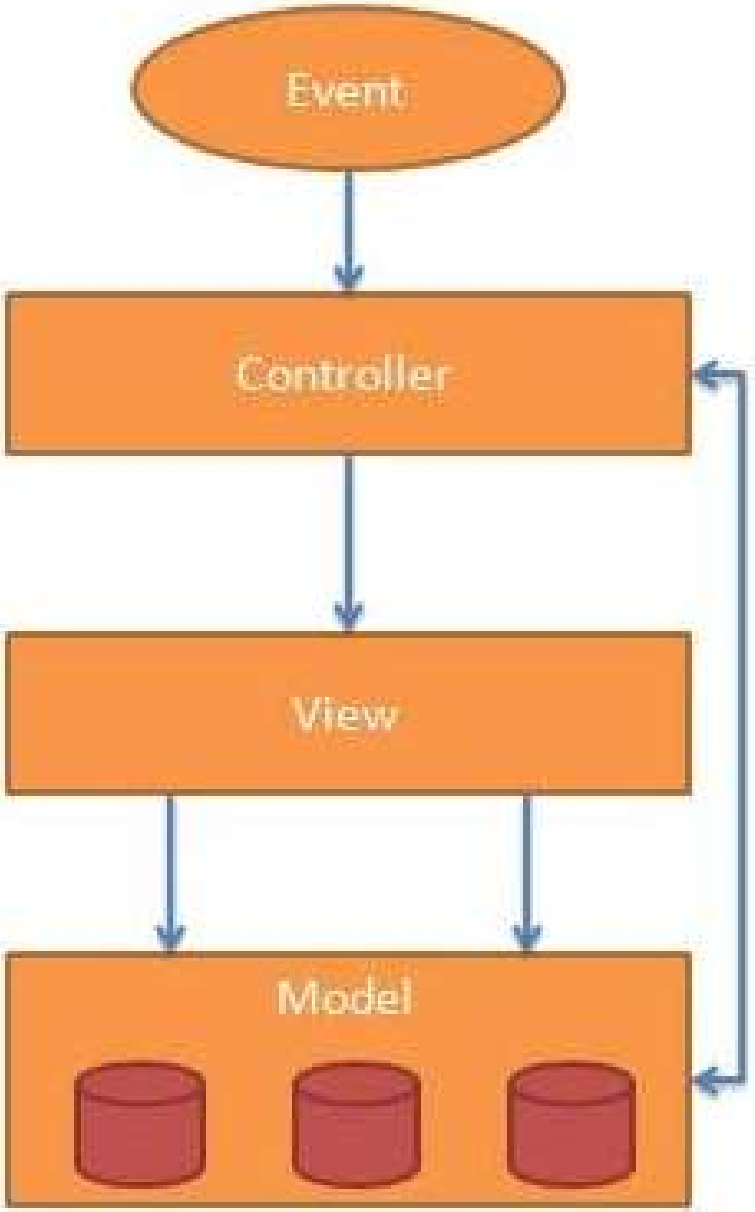
- **Not Secure** − Being JavaScript only framework, application written in AngularJS are not safe. Server side authentication and authorization is must to keep an application secure.
- **Not degradable** − If the user of your application disables JavaScript, then nothing would be visible, except the basic page.

AngularJS - MVC Architecture

- **M**odel **V**iew **C**ontroller or MVC as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts –
- **Model** – It is the lowest level of the pattern responsible for maintaining data.
- **View** – It is responsible for displaying all or a portion of the data to the user.
- **Controller** – It is a software Code that controls the interactions between the Model and View.

AngularJS - MVC Architecture

- MVC is popular because it isolates the application logic from the user interface layer and supports separation of concerns. The controller receives all requests for the application and then works with the model to prepare any data needed by the view. The view then uses the data prepared by the controller to generate a final presentable response. The MVC abstraction can be graphically represented as follows.


Event
Controller
View
Model

AngularJS - MVC Architecture

- The Model-

The model is responsible for managing application data. It responds to the request from view and to the instructions from controller to update itself.

- The View-

A presentation of data in a particular format, triggered by the controller's decision to present the data. They are script-based template systems such as JSP, ASP, PHP and very easy to integrate with AJAX technology.

- The Controller

The controller responds to user input and performs interactions on the data model objects. The controller receives input, validates it, and then performs business operations that modify the state of the data model.

AngularJS Directives

The AngularJS framework can be divided into three major parts –

- **ng-app** – This directive defines and links an AngularJS application to HTML.
- **ng-model** – This directive binds the values of AngularJS application data to HTML input controls.
- **ng-bind** – This directive binds the AngularJS application data to HTML tags.

Creating AngularJS Application

Step 1: Load framework

- Being a pure JavaScript framework, it can be added using <Script> tag.
- <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"> </script>

Creating AngularJS Application

Step 2: Define AngularJS application using ng-app directive

- ...

Step 3: Define a model name using ng-model directive

- Enter your Name: <input type = "text" ng-model = "name">

Creating AngularJS Application

- Step 4: Bind the value of above model defined using ng-bind directive
- Hello !

Code Hello

```
<html>
  <head>
    <title>AngularJS First Application</title>
  </head>

  <body>
    <h1>Sample Application</h1>

    <div ng-app = "">
      <p>Enter your Name: <input type = "text" ng-model = "name"></p>
      <p>Hello <span ng-bind = "name"></span>!</p>
    </div>

    <script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
    </script>

  </body>
</html>
```

AngularJS - Directives

- AngularJS directives are used to extend HTML. They are special attributes starting with **ng**-prefix. Let us discuss the following directives –
- **ng-app** – This directive starts an AngularJS Application.
- **ng-init** – This directive initializes application data.
- **ng-model** – This directive defines the model that is variable to be used in AngularJS.
- **ng-repeat** – This directive repeats HTML elements for each item in a collection.
- **List of other AngularJS directives**

ng-app directive

- The ng-app directive starts an AngularJS Application. It defines the root element. It automatically initializes or bootstraps the application when the web page containing AngularJS Application is loaded.
- In the following example, we define a default AngularJS application using ng-app attribute of a element.
- ...

ng-init directive

- The ng-init directive initializes an AngularJS Application data. It is used to assign values to the variables. In the following example, we initialize an array of countries. We use JSON syntax to define the array of countries.
- ...

ng-model directive

- The ng-model directive defines the model/variable to be used in AngularJS Application. In the following example, we define a model named *name*.
- ... Enter your Name: <input type = "text" ng-model = "name">

ng-repeat directive

- The ng-repeat directive repeats HTML elements for each item in a collection. In the following example, we iterate over the array of countries.

```
<div ng-app = ""> ... <p>List of Countries with locale:</p>
 <ol>
<li ng-repeat = "country in countries"> {{ 'Country: ' +
country.name + ', Locale: ' + country.locale }} </li>
 </ol>
</div>
```

AngularJS Expressions

- AngularJS expressions can be written inside double braces: {{ *expression* }}.
- AngularJS expressions can also be written inside a directive: ng-bind="*expression*".
- AngularJS will resolve the expression, and return the result exactly where the expression is written.
- **AngularJS expressions** are much like **JavaScript expressions:** They can contain literals, operators, and variables.
- Example {{ 5 + 5 }} or {{ firstName + " " + lastName }}

Example

- ```
  <!DOCTYPE html>
  <html>
  <script src="https://ajax.googleapis.com/ajax/libs
  /angularjs/1.6.9/angular.min.js"></script>
  <body>

    My first expression: {{ 5 + 5 }}

  </body>
  </html>
  ```
  ```

AngularJS Controllers

- AngularJS applications are controlled by controllers.
- The **ng-controller** directive defines the application controller.
- A controller is a **JavaScript Object**, created by a standard JavaScript **object constructor**.

Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">

First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{firstName + " " + lastName}}

</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
   $scope.firstName = "John";
   $scope.lastName = "Doe";
});
</script>

</body>
</html>
```

AngularJS Filters

AngularJS provides filters to transform data:

- *currency* - Format a number to a currency format.
- *date* - Format a date to a specified format.
- *filter* - Select a subset of items from an array.
- *json* - Format an object to a JSON string.
- *limitTo* - Limits -an array/string, into a specified number of elements/characters.
- *lowercase* - Format a string to lower case.
- *number* - Format a number to a string.
- *orderBy* - Orders an array by an expression.
- *uppercase* - Format a string to upper case.

Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="myApp" ng-controller="personCtrl">

<p>The name is {{firstName + " " + lastName | uppercase }}</p>

</div>

<script>
angular.module('myApp', []).controller('personCtrl', function($scope) {
    $scope.firstName = "John",
    $scope.lastName = "Doe"
});
</script>

</body>
</html>
```

AngularJS Tables

- The ng-repeat directive is perfect for displaying tables.

Example

```
<div ng-app="myApp" ng-controller="customersCtrl">

<table>
  <tr ng-repeat="x in names">
    <td>{{ x.Name }}</td>
    <td>{{ x.Country }}</td>
  </tr>
</table>

</div>
```

AngularJS HTML DOM

- AngularJS has directives for binding application data to the attributes of HTML DOM elements.

Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="" ng-init="mySwitch=true">
<p>
<button ng-disabled="mySwitch">Click Me!</button>
</p>
<p>
<input type="checkbox" ng-model="mySwitch"/>Button
</p>
<p>
{{ mySwitch }}
</p>
</div>

</body>
</html>
```

Explaination

- The **ng-disabled** directive binds the application data **mySwitch** to the HTML button's **disabled** attribute.
- The **ng-model** directive binds the value of the HTML checkbox element to the value of **mySwitch**.
- If the value of **mySwitch** evaluates to **true**, the button will be disabled:

AngularJS Forms

- Forms in AngularJS provides data-binding and validation of input controls.

Input controls are the HTML input elements:

- input elements
- select elements
- button elements
- textarea elements

Data-Binding

- Input controls provides data-binding by using the ng-model directive.
- <input type="text" ng-model="firstname">
- The application does now have a property named firstname.
- The ng-model directive binds the input controller to the rest of your application.
- The property firstname, can be referred to in a controller:

Checkbox

- A checkbox has the value true or false. Apply the ng-model directive to a checkbox, and use its value in your application.

- <form>
  Check to show a header:
  <input type="checkbox" ng-model="myVar">
  </form>

  <h1 ng-show="myVar">My Header</h1>

Radiobuttons

- Bind radio buttons to your application with the ng-model directive.
- Radio buttons with the same ng-model can have different values, but only the selected one will be used.
- <form>
  Pick a topic:
  <input type="radio" ng-model="myVar" value="dogs">Dogs
  <input type="radio" ng-model="myVar" value="tuts">Tutorials
  <input type="radio" ng-model="myVar" value="cars">Cars
  </form>

Select box

- Bind select boxes to your application with the ng-model directive.
- The property defined in the ng-model attribute will have the value of the selected option in the select box.
- <form>
  Select a topic:
  <select ng-model="myVar">
  <option value=“">
  <option value="dogs">Dogs
  <option value="tuts">Tutorials
  <option value="cars">Cars
  </select>
  </form>

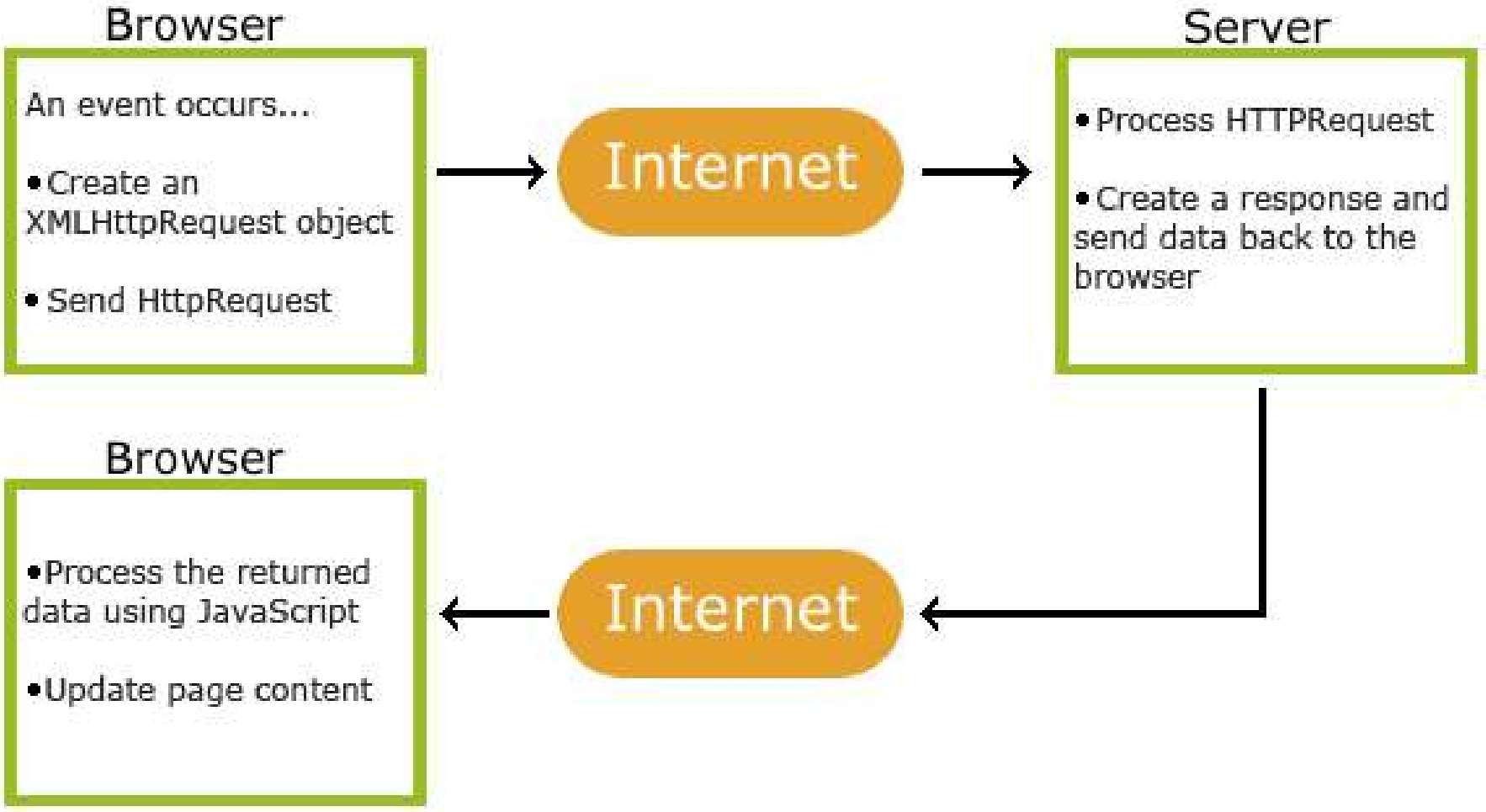
AJAX Introduction

- What is AJAX?
- AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.
- AJAX is not a programming language.
- AJAX just uses a combination of:
- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)

AJAX Introduction

- AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

How AJAX Works



Explaination

1. An event occurs in a web page (the page is loaded, a button is clicked)
2. An XMLHttpRequest object is created by JavaScript
3. The XMLHttpRequest object sends a request to a web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript

NodeJS

What is Node.js?

- Node.js is an open source server environment
- Node.js is free
- Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js uses JavaScript on the server

Why Node.js?

- A common task for a web server can be to open a file on the server and return the content to the client.

Here is how PHP or ASP handles a file request:

- Sends the task to the computer's file system.
- Waits while the file system opens and reads the file.
- Returns the content to the client.
- Ready to handle the next request.

Why Node.js?

Here is how Node.js handles a file request:

- Sends the task to the computer's file system.
- Ready to handle the next request.
- When the file system has opened and read the file, the server returns the content to the client.
- Node.js eliminates the waiting, and simply continues with the next request.
- Node.js runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.

What Can Node.js Do?

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database

What is a Node.js File?

- Node.js files contain tasks that will be executed on certain events
- A typical event is someone trying to access a port on the server
- Node.js files must be initiated on the server before having any effect
- Node.js files have extension ".js"

Node.js Get Started

Download Node.js

- The official Node.js website has installation instructions for Node.js: https://nodejs.org

Node.js Get Started

- Once you have downloaded and installed Node.js on your computer, let's try to display "Hello World" in a web browser.
- Create a Node.js file named "myfirst.js", and add the following code:

Node.js Get Started

- var http = require('http');

```
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-
Type': 'text/html'});
  res.end('Hello World!');
}).listen(8080);
```

Node.js Get Started

- Save the file on your computer: C:\Users*Your Name*\myfirst.js
- Navigate to the folder that contains the file "myfirst.js“
- Start your command line interface, write node myfirst.js and hit enter:
- C:\Users*Your Name*>node myfirst.js
- http://localhost:8080

What is Callback?

- Callback is an asynchronous equivalent for a function. A callback function is called at the completion of a given task. Node makes heavy use of callbacks. All the APIs of Node are written in such a way that they support callbacks.

What is Callback?

- For example, a function to read a file may start reading file and return the control to the execution environment immediately so that the next instruction can be executed.
- Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as a parameter.
- So there is no blocking or wait for File I/O. This makes Node.js highly scalable, as it can process a high number of requests without waiting for any function to return results.

Blocking Code Example

- Create a text file named **input.txt** with the following content –
- Create a js file named **main.js** with the following code –
- To include the File System module, use the require() method-

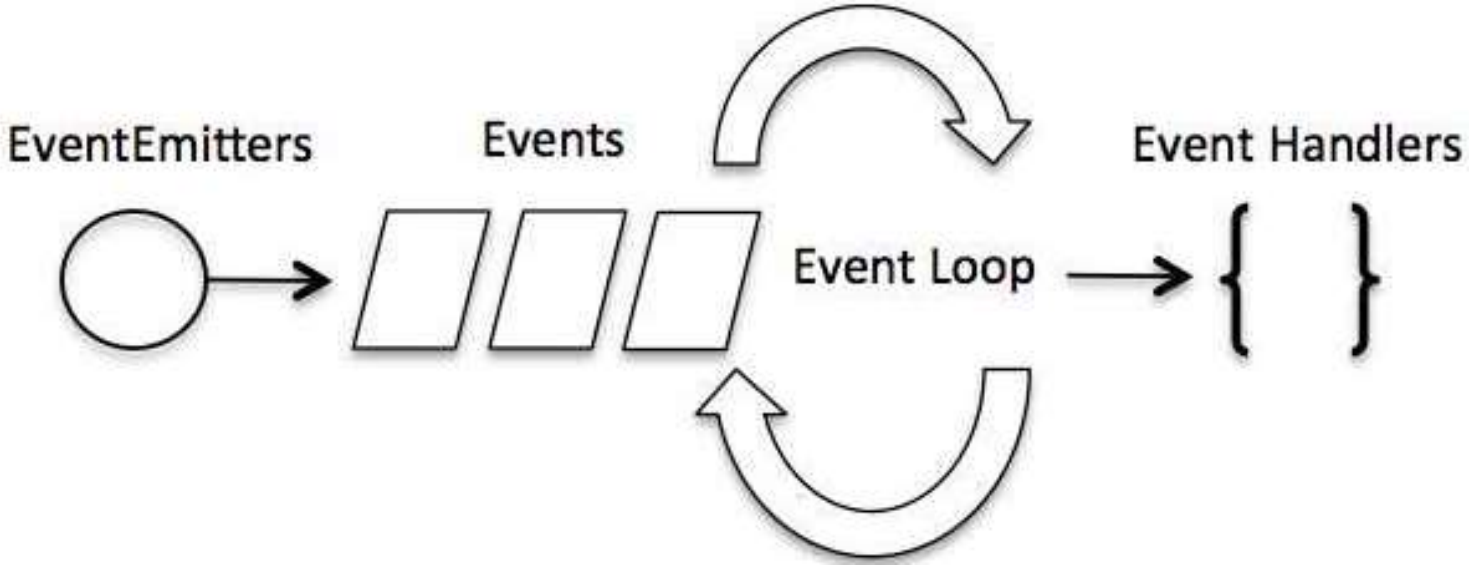
- var fs = require("fs");
- var data = fs.readFileSync('input.txt');
- console.log(data.toString());
- console.log("Program Ended");

Event-Driven Programming

- Node.js uses events heavily and it is also one of the reasons why Node.js is pretty fast compared to other similar technologies.
- As soon as Node starts its server, it simply initiates its variables, declares functions and then simply waits for the event to occur.

Event-Driven Programming

- In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected.



Event-Driven Programming

- Although events look quite similar to callbacks, the difference lies in the fact that callback functions are called when an asynchronous function returns its result, whereas event handling works on the observer pattern.
- The functions that listen to events act as **Observers**. Whenever an event gets fired, its listener function starts executing.
- Node.js has multiple in-built events available through events module and EventEmitter class which are used to bind events and event-listeners.

Creating Buffers

- Node Buffer can be constructed in a variety of ways.

Method 1

- Following is the syntax to create an uninitiated Buffer of **10** octets –

```
var buf = new Buffer(10);
```

Creating Buffers

- **Method 2**
- Following is the syntax to create a Buffer from a given array −

```
var buf = new Buffer([10, 20, 30, 40, 50]);
```

Writing to Buffers

buf.write(string[, offset][, length][, encoding])

- **string** – This is the string data to be written to buffer.
- **offset** – This is the index of the buffer to start writing at. Default value is 0.
- **length** – This is the number of bytes to write. Defaults to buffer.length.
- **encoding** – Encoding to use. 'utf8' is the default encoding.

Reading from Buffers

buf.toString([encoding][, start][, end])

- **encoding** – Encoding to use. 'utf8' is the default encoding.
- **start** – Beginning index to start reading, defaults to 0.
- **end** – End index to end reading, defaults is complete buffer.

What are Streams?

Streams are objects that let you read data from a source or write data to a destination in continuous fashion.

In Node.js, there are four types of streams –

- **Readable** – Stream which is used for read operation.
- **Writable** – Stream which is used for write operation.
- **Duplex** – Stream which can be used for both read and write operation.
- **Transform** – A type of duplex stream where the output is computed based on input.

What are Streams?

Each type of Stream is an **EventEmitter** instance and throws several events at different instance of times.

For example, some of the commonly used events are –

- **data** – This event is fired when there is data is available to read.
- **end** – This event is fired when there is no more data to read.
- **error** – This event is fired when there is any error receiving or writing data.
- **finish** – This event is fired when all the data has been flushed to underlying system.

What are Streams?

- var fs = require("fs");
- // Create a readable stream var readerStream = fs.createReadStream('input.txt');
- // Create a writable stream var writerStream = fs.createWriteStream('output.txt');
- // Pipe the read and write operations
- // read input.txt and write data to output.txt readerStream.pipe(writerStream); console.log("Program Ended");

What are Streams?

- Now run the main.js to see the result –
- $ node main.js Verify the Output.
- Program Ended
- Open output.txt created in your current directory; it should contain the following input file data.

File System

- Node implements File I/O using simple wrappers around standard POSIX functions.
- **POSIX,** the **Portable Operating System Interface**, defines the standard APIs for Unix.
- The Node File System (fs) module can be imported using the following syntax −
- var fs = require("fs")

Synchronous vs Asynchronous

- Every method in the fs module has synchronous as well as asynchronous forms.
- Asynchronous methods take the last parameter as the completion function callback and the first parameter of the callback function as error.
- It is better to use an asynchronous method instead of a synchronous method, as the former never blocks a program during its execution, whereas the second one does.

Example

- Create a text file named **input.txt** with the following content –
- Let us create a js file named **main.js** with the following code –

Example

- var fs = require("fs");
- // Asynchronous read fs.readFile('input.txt', function (err, data) { if (err) { return console.error(err); } console.log("Asynchronous read: " + data.toString()); });
- // Synchronous read var data = fs.readFileSync('input.txt'); console.log("Synchronous read: " + data.toString()); console.log("Program Ended");

- Now run the main.js to see the result –
- $ node main.js
- Verify the Output.

Object

- Node.js global objects are global in nature and they are available in all modules. We do not need to include these objects in our application, rather we can use them directly. These objects are modules, functions, strings and object itself as explained below.

__filename

- The **__filename** represents the filename of the code being executed. This is the resolved absolute path of this code file. For a main program, this is not necessarily the same filename used in the command line. The value inside a module is the path to that module file.
- console.log(__filename);

__dirname

- The **__dirname** represents the name of the directory that the currently executing script resides in.
- console.log(__dirname);

Node.js - Utility Modules

- There are several utility modules available in Node.js module library. These modules are very common and are frequently used while developing any Node based application.

Node.js - Utility Modules

Sr.No.	Module Name & Description
1	OS Module- Provides basic operating-system related utility functions.
2	Path Module- Provides utilities for handling and transforming file paths.
3	Net Module- Provides both servers and clients as streams. Acts as a network wrapper.
4	DNS Module- Provides functions to do actual DNS lookup as well as to use underlying operating system name resolution functionalities.
5	Domain Module- Provides ways to handle multiple different I/O operations as a single group.

What is REST architecture?

- REST stands for REpresentational State Transfer. REST is web standards based architecture and uses HTTP Protocol. It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods. REST was first introduced by Roy Fielding in 2000.

What is REST architecture?

- A REST Server simply provides access to resources and REST client accesses and modifies the resources using HTTP protocol. Here each resource is identified by URIs/ global IDs. REST uses various representation to represent a resource like text, JSON, XML but JSON is the most popular one.

HTTP methods

- Following four HTTP methods are commonly used in REST based architecture.
- **GET** – This is used to provide a read only access to a resource.
- **PUT** – This is used to create a new resource.
- **DELETE** – This is used to remove a resource.
- **POST** – This is used to update a existing resource or create a new resource.

Thank you