NodeJS vs AngularJS: Differences You Should Know

Before comparing NodeJS and AngularJS, let’s understand the MEAN stack architecture. MEAN Stack is a collection of Javascript-based technologies used to build dynamic websites and web applications. MEAN stands for MongoDB, ExpressJS, AngularJS, and NodeJS.

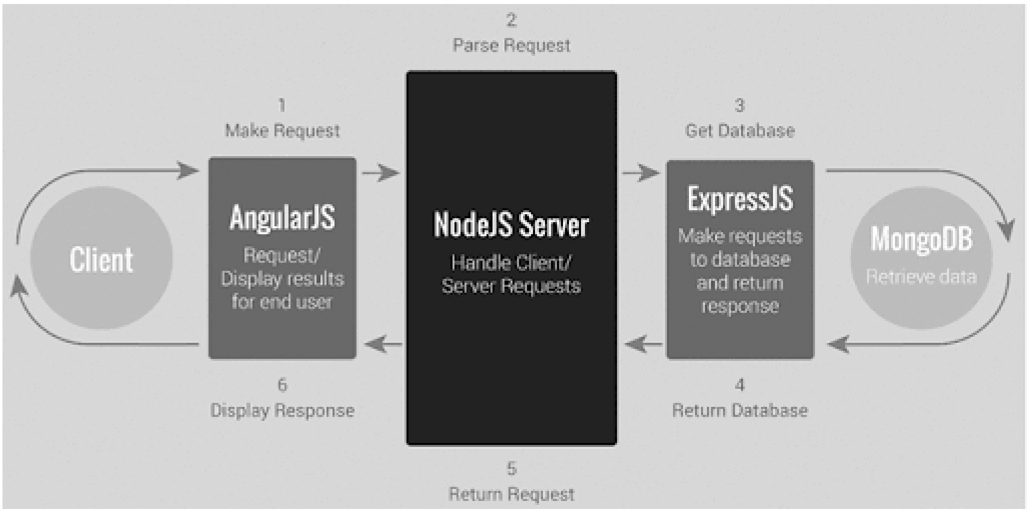
M: MongoDB is a popular database management system that implements a NoSQL data structure.

E: ExpressJS is a framework that makes development in NodeJS painless.

A: AngularJS is built on top of the HTML to expand its horizons to web apps.

N: NodeJS is a run-time environment that can be used as the application backend.

NodeJS and AngularJS were developed to build web apps and websites using JavaScript, but they are quite different in their architecture and working. Let’s have a look at the architecture of a web application so that you can understand how you are going to use both of them.



NodeJS vs AngularJS: Head to Head Comparison

So as you can see AngularJS is used for front-end development which includes interacting with the client and NodeJS is used for backend development which includes interaction with the database server. Both can be combined to create isomorphic web applications.

NodeJS	**AngularJS**
NodeJS is the cross-platform and a run-time environment for Javascript applications.	AngularJS is an open-source platform for web application development which is maintained by Google.

In order to use NodeJS, you need to install it into your system.	In order to use AngularJS, you need to add the javascript file just like other javascript files for using it in applications.
Node.js is a platform built on top of Google's V8 JavaScript engine.	Angular follows the syntax of javascript that's it.
NodeJS is written in C, C++ and Javascript languages.	AngularJS is written in Javascript but it is different from frameworks like JQuery.
Node.Js has many other different frameworks such as Express.js, partial.js, and Sail.js.	It is a framework itself.
It supports nonblocking I/O API and event-driven architecture.	It enables the extension of HTML syntax for describing the components and elements of the application.

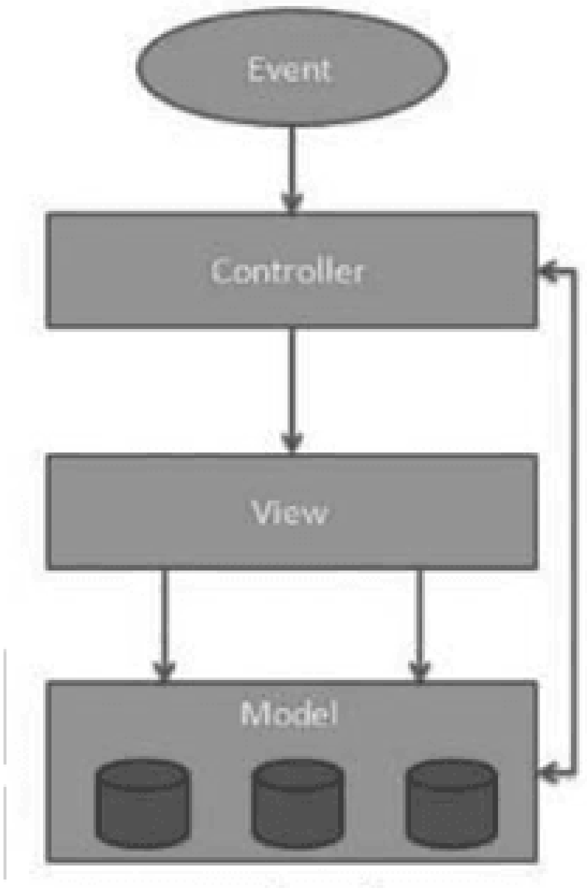
MVC Framework

AngularJS provides a smooth Model View Control Architecture which is also very dynamic in nature. As we know, any application is built up from combining different modules, which are initialized differently from each other. But still, these modules are connected with each other by some logic. The developers have to build all the components separately and then have to combine them with some code and applied logic to convert them into a single application. This, of course, is an overhead for the developers while using an MVC Framework.

MVC makes it easier for developers to build the client-side web application. All the MVC elements which are developed separately are combined automatically using AngularJS Framework. There is no need for developers to write extra code to fit all the elements together. It allows you to put the MVC element separately and automatically sets them together accordingly.

AngularJS Architecture

It is blessed with an MVW (Model-View-Whatever) architecture and is capable of supporting other patterns too, like Model-View-Controller or Model-View-View Model. The view modifies and manipulates the DOM to update the data and the behaviour. But with the use of AngularJS development, the DOM manipulation is the task of the directives and not the view.



1. Model

It is the lowest level responsible for maintaining data and managing application data. It responds to the request from view and to the instructions from the controller to update itself.

2. View

It is responsible for showing all kinds of data to the user. They are script-based template systems such as JSP, ASP, PHP and very easy to integrate with AJAX technology.

3. Controller

It controls the interaction between the model and the view. The controller responds to user input and performs interactions on the data model objects.

Uses of AngularJS

- Manages the state of models
- Integrates with other UI tools
- Manipulates DOM
- Allows writing custom HTML codes
- It is meant for javascript developers to create dynamic web pages in a quick time

NodeJS Architecture

Node.js is a server-side platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

1. Asynchronous and Event-Driven

All APIs of Node.js library are asynchronous. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.

2. Single-Threaded but Highly Scalable

Node.js uses a single-threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single-threaded program and the same program can provide service to a much larger number of requests than traditional servers.

3. Very Fast

Built on Google Chrome's V8 JavaScript Engine, NodeJS is very fast in code execution.

4. No Buffering

Node.js operates on a single thread, using non-blocking I/O calls, allowing it to support tens of thousands of concurrent connections without incurring the cost of thread context switching. The design of sharing a single thread between all the requests that use the observer pattern is intended for building highly concurrent applications, where any function performing I/O must use a callback. In order to accommodate the single-threaded event loop, Node.js utilizes the libuv library that in turn uses a fixed-sized thread pool that is responsible for some of the non-blocking asynchronous I/O operations.

Architecture Description:

- Here "n" number of Clients Send a request to Web Server. Let us assume they are accessing our Web Application concurrently.
- Let us assume, our Clients Are Client-1, Client-2… and Client-n.
- Web Server internally maintains a Limited Thread pool. Let us assume "m" number of Threads in the Thread pool.
- Web Server receives those requests one by one.
  Web Server pickup Client-1 Request-1, Pickup one Thread T-1 from Thread pool and assign this request to Thread T-1

1.
    1. Thread T-1 reads Client-1 Request-1 and process it

2. Client-1 Request-1 does not require any Blocking IO Operations
3. Thread T-1 does necessary steps and prepares Response-1 and send it back to the Server
4. Web Server in-turn send this Response-1 to the Client-1

- Web Server pickup another Client-2 Request-2, Pickup one Thread T-2 from Thread pool and assign this request to Thread T-2

1.
    1. Thread T-2 reads Client-1 Request-2 and process it
    2. Client-1 Request-2 does not require any Blocking IO Operations
    3. Thread T-2 does necessary steps and prepares Response-2 and send it back to the Server
    4. Web Server in-turn send this Response-2 to the Client-2

- Web Server pickups another Client-n Request-n, Pickup one Thread T-n from Thread pool and assign this request to Thread T-n

1.
    1. Thread T-n reads Client-n Request-n and processes it
    2. Client-n Request-n require heavy Blocking IO and computation Operations
    3. Thread T-n takes more time to interact with external systems, does necessary steps and prepares Response-n and send it back to the Server
    4. Web Server in-turn send this Response-n to the Client-n

If “n” is greater than “m” (Most of the time, it's true), then the server assigns Threads to Client Requests up to available Threads. After all m Threads are utilized, then remaining Client’s Request should wait in the Queue until some of the busy Threads finish their Request-Processing Job and free to pick up next Request. If those threads are busy with Blocking IO Tasks (For example, interacting with Database, file system, JMS Queue, external services, etc.) for a longer time, then remaining clients should wait for more.

- Once Threads are free in Thread Pool and available for the next tasks, the Server picks up those threads and assigns them to remaining Client Requests.
- Each Thread utilizes many resources like memory etc. So before going from ‘busy’ to ‘waiting’ state, they should release all acquired resources.