Web Frame Work Using JavaScript

Vivek Sharma

The Course Break Down:

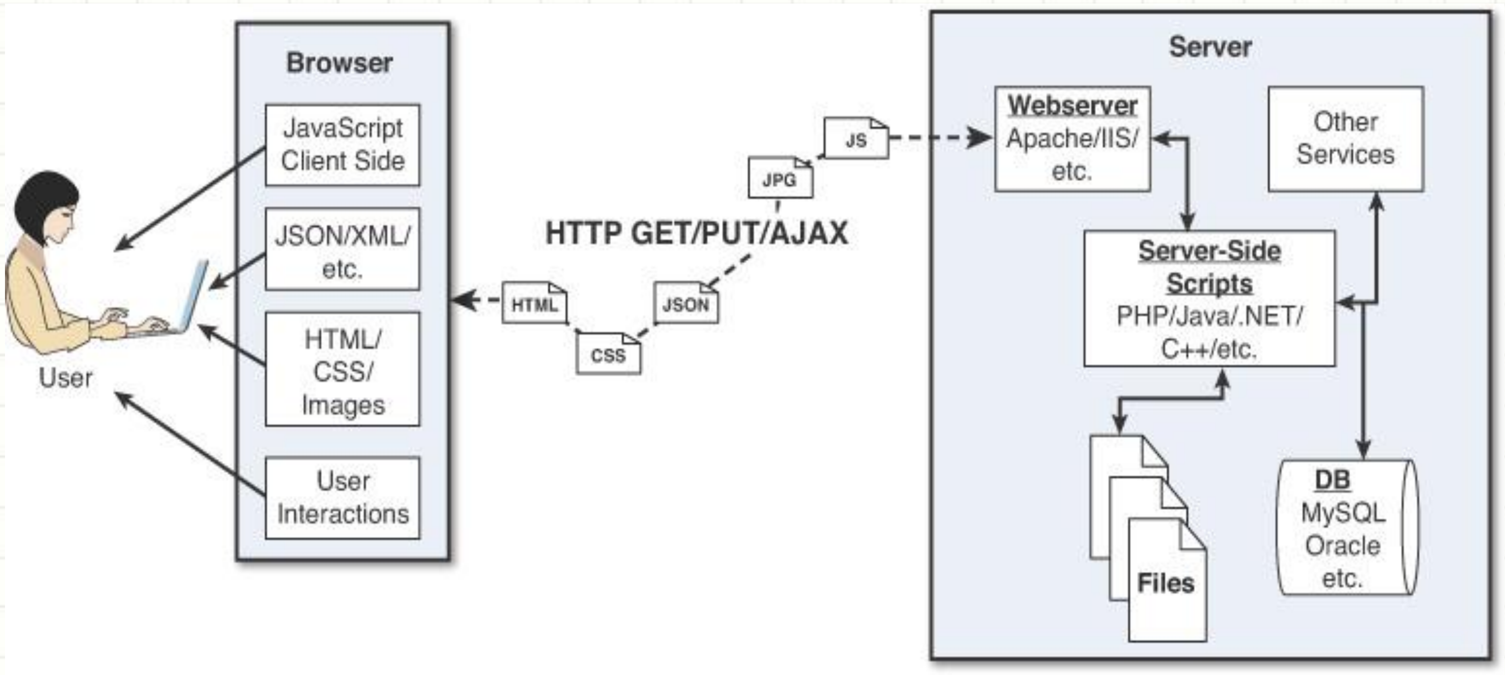
- Class 1: Mean Stack Fundamentals.
- Class 2: Learning Node.js
- Class 3: Implementing HTTP Services in Node.js
- Class 4: Scaling the Applications Using Node.js
- Class 5: Understanding NoSql and MongoDB
- Class 6: Understanding Express and Implementation.
- Class 7: Understanding Angular
- Class 8: Understanding the Angular Directives and Angular Web Application.
- Class 9: Creating a Shopping Cart.
- Class 10: Creating another project.

WELCOME

Some Points before we start off:

- Introduction.
- Grading/ Assignments.
- About the Class. [Basic to Professional]
- Pace.
- Interview Preparation.
- Labs.

The Traditional Stack



The components of the TraditionalStack

- Following are the main components of any traditional stack:
  - The User
  - The Browser
  - The Webserver.
  - The backend services.
- The websites may differ in different dimensions but they all tend to have these basic components in one manifestation or the other.

The Browser

- The browser plays three roles in the web framework:
- Provide communication to and from the webserver
- Interpret the data from the server and render it into the view that the user actually sees
- Handle user interaction through the keyboard, mouse, touchscreen, or other input device and take the appropriate action

Browser-to-Webserver Communication

- Browser-to-webserver communication consists of a series of requests, using the HTTP and HTTPS protocols. Hypertext Transfer Protocol (HTTP) is used to define communication between the browser and the webserver. HTTP defines what types of requests can be made as well as the format of those requests and the HTTP response.
- HTTPS adds an additional security layer, SSL/TLS, to ensure secure connections by requiring the webserver to provide a certificate to the browser. The user can then determine whether to accept the certificate before allowing the connection.
- There are three main types of requests that a browser will make to a webserver:
- GET: The GET request is typically used to retrieve data from the server, such as .html files, images, or JSON data.
- POST: POST requests are used when sending data to the server, such as adding an item to a shopping cart or submitting a web form.
- AJAX: Asynchronous JavaScript and XML (AJAX) is actually just a GET or POST request that is done directly by JavaScript running in the browser. Despite the name, an AJAX request can receive XML, JSON, or raw data in the response.

Rendering the Browser View

- The screen that the user actually views and interacts with is often made up of several different pieces of data retrieved from the webserver. The browser reads data from the initial URL and then renders the HTML document to build a Document Object Model (DOM). The DOM is a tree structure object with the HTML document as the root. The structure of the tree basically matches the structure of the HTML document. For example, document will have html as a child, and html will have head and body as children, and body may have div, p, or other elements as children, like this:
- Document
- + html
-     + head
-     + body
-       + div
-         + p

Rendering the Browser View

- The browser interprets each DOM element and renders it to the user's screen to build the webpage view. The browser often gets various types of data from multiple webserver requests to build a webpage. The following are the most common types of data the browser uses to render the final user view as well as define the webpage behavior:
- HTML files: These provide the fundamental structure of the DOM.
- CSS files: These define how each of the elements on the page is to be styled, in terms of font, color, borders, and spacing.
- Client-side scripts: These are typically JavaScript files. They can provide added functionality to a webpage, manipulate the DOM to change the look of the webpage, and provide any necessary logic required to display the page and provide functionality.
- Media files: Image, video, and sound files are rendered as part of the webpage.
- Data: Data such as XML, JSON, or raw text can be provided by the webserver as a response to an AJAX request. Rather than send a request back to the server to rebuild the webpage, new data can be retrieved via AJAX and inserted into the webpage via JavaScript.
- HTTP headers: HTTP defines a set of headers that the browser can use and client-side scripts to define the behavior of the webpage. For example, cookies are contained in the HTTP headers. The HTTP headers also define the type of data in the request as well as the type of data

Rendering the Browser View

- The browser interprets each DOM element and renders it to the user's screen to build the webpage view. The browser often gets various types of data from multiple webserver requests to build a webpage. The following are the most common types of data the browser uses to render the final user view as well as define the webpage behavior:
- HTML files: These provide the fundamental structure of the DOM.
- CSS files: These define how each of the elements on the page is to be styled, in terms of font, color, borders, and spacing.
- Client-side scripts: These are typically JavaScript files. They can provide added functionality to a webpage, manipulate the DOM to change the look of the webpage, and provide any necessary logic required to display the page and provide functionality.
- Media files: Image, video, and sound files are rendered as part of the webpage.
- Data: Data such as XML, JSON, or raw text can be provided by the webserver as a response to an AJAX request. Rather than send a request back to the server to rebuild the webpage, new data can be retrieved via AJAX and inserted into the webpage via JavaScript.
- HTTP headers: HTTP defines a set of headers that the browser can use and client-side scripts to define the behavior of the webpage. For example, cookies are contained in the HTTP headers. The HTTP headers also define the type of data in the request as well as the type of data

The User Interactions

- The user interacts with the browser via mice, keyboards, and touchscreens.
- A browser has an elaborate event system that captures user input events and then takes the appropriate actions.
- Actions vary from displaying a popup menu to loading a new document from the server to executing client-side JavaScript.

Webservers

- A webserver’s main focus is handling requests from browsers. As described earlier, a browser may request a document, post data, or perform an AJAX request to get data. The webserver uses HTTP headers as well as a URL to determine what action to take. This is where things get very different, depending on the webserver, configuration, and technologies used.
- Most out-of-the-box webservers such as Apache and IIS are made to serve static files such as .html, .css, and media files. To handle POST requests that modify server data and AJAX requests to interact with backend services, webservers need to be extended with server-side scripts.
- A server-side script is really anything that a webserver can execute in order to perform the task the browser is requesting. These scripts can be written in PHP, Python, C, C++, C#, Perl, Java, ... the list goes on and on. Webservers such as Apache and IIS provide mechanisms to include server-side scripts and then wire them up to specific URL locations requested by the browser. This is where having a solid webserver framework can make a big difference. It often takes quite a bit of configuration to enable various scripting languages and wire up the server-side scripts so that the webserver can route the appropriate requests to the appropriate scripts.
- Server-side scripts either generate a response directly by executing their code or connect with other backend servers such as databases to obtain the necessary information and then use that information to build and send the appropriate responses.

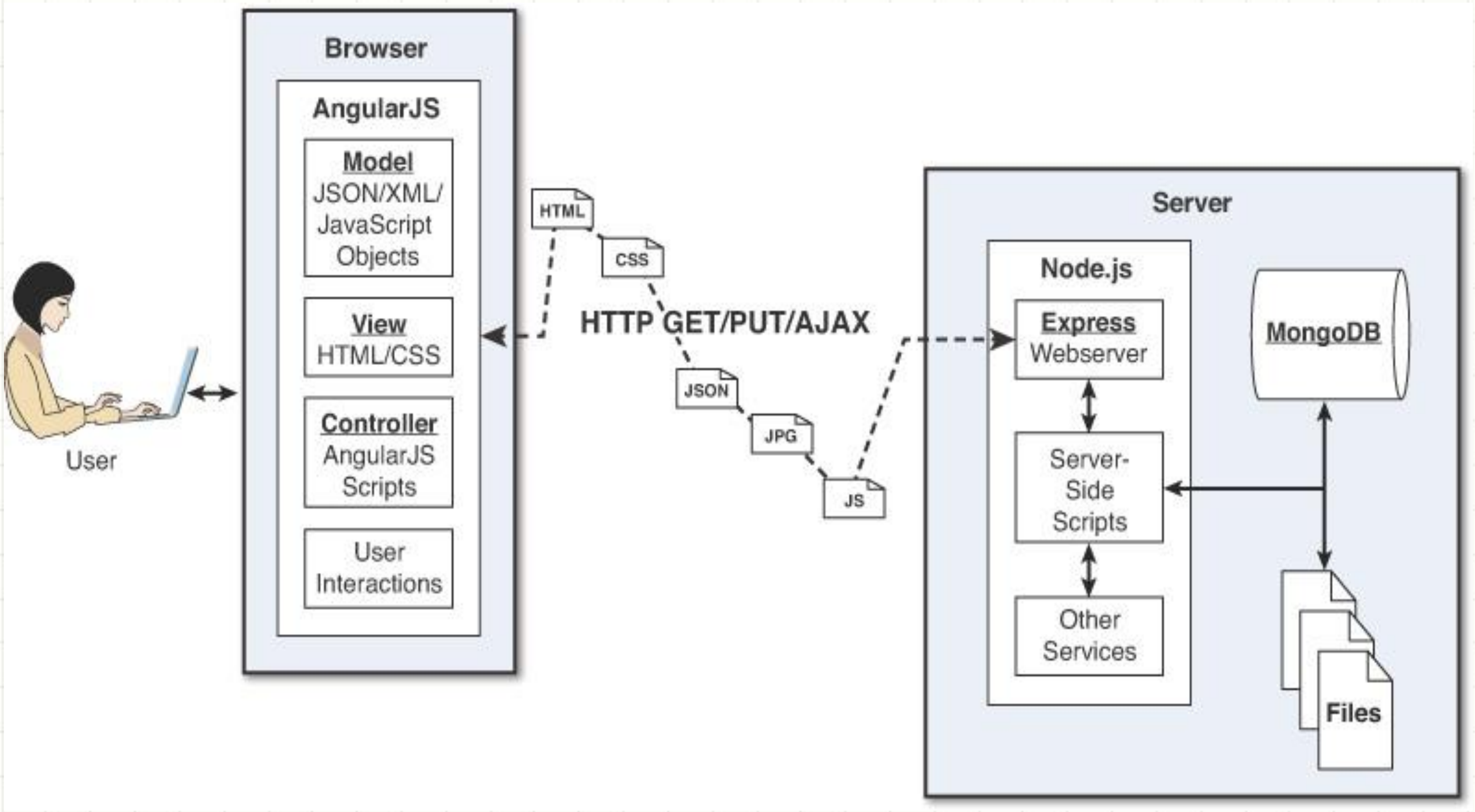
The Backend Services.

- Backend services are services that run behind a webserver and provide data that is used to build responses to the browser.
- The most common type of backend service is a database that stores information.
- When a request comes in from the browser that requires information from the database or other backend service, the server-side script connects to the database, retrieves the information, formats it, and then sends it back to the browser.
- On the other hand, when data comes in from a web request that needs to be stored in the database, the server-side script connects to the database and updates the data.

Moving to the MEAN Stack

- In the Node.js-to-AngularJS stack, Node.js provides the fundamental platform for development. The backend services and server-side scripts are all written in Node.js. MongoDB provides the data store for the website but is accessed via a MongoDB driver Node.js module. The webserver is defined by Express, which is also a Node.js module.
- The view in the browser is defined and controlled using the AngularJS framework. AngularJS is an MVC framework in which the model is made up of JSON or JavaScript objects, the view is HTML/CSS, and the controller is AngularJS JavaScript code.
- The next slide tries to map the traditional components to the mean stack components. Be ware that the following figure is very basic representation and can differ in a lot of cases.

The Mean Stack



Node.js

- Node.js is a development framework that is based on Google's V8 JavaScript engine and executes it.
- You can write most—or maybe even all—of your server-side code in Node.js, including the webserver and the server-side scripts and any supporting web application functionality. The fact that the webserver and the supporting web application scripts are running together in the same server-side application allows for much tighter integration between the webserver and the scripts. Also, the webserver can run directly on the Node.js platform as a Node.js module, which means it's much easier than using, say, Apache for wiring up new services or server-side scripts.

Node.js

- The following are just a few reasons Node.js is a great framework:
- JavaScript end-to-end: One of the biggest advantages of Node.js is that it allows you to write both server- and client-side scripts in JavaScript. There have always been difficulties in deciding whether to put logic in client-side scripts or server-side scripts. With Node.js you can take JavaScript written on the client and easily adapt it for the server and vice versa. An added plus is that client developers and server developers are speaking the same language.
- Event-driven scalability: Node.js applies a unique logic to handling web requests. Rather than having multiple threads waiting to process web requests, with Node.js they are processed on the same thread, using a basic event model. This allows Node.js webservers to scale in ways that traditional webservers can't.
- Extensibility: Node.js has a great following and very active development community. People are providing new modules to extend Node.js functionality all the time. Also, it is very simple to install and include new modules in Node.js; you can extend a Node.js project to include new functionality in minutes.
- Fast implementation: Setting up Node.js and developing in it are super easy. In only a few minutes you can install Node.js and have a working webserver.

Express

The Express module acts as the webserver in the Node.js-to-AngularJS stack. Because it runs in Node.js, it is easy to configure, implement, and control. The Express module extends Node.js to provide several key components for handling web requests. It allows you to implement a running webserver in Node.js with only a few lines of code.

For example, the Express module provides the ability to easily set up destination routes (URLs) for users to connect to. It also provides great functionality in terms of working with HTTP request and response objects, including things like cookies and HTTP headers.

- The following is a partial list of the valuable features of Express:
- Route management: Express makes it easy to define routes (URL endpoints) that tie directly to the Node.js script functionality on the server.
- Error handling: Express provides built-in error handling for “document not found” and other errors.
- Easy integration: An Express server can easily be implemented behind an existing reverse proxy system, such as Nginx or Varnish. This allows you to easily integrate it into your existing secured system.
- Cookies: Express provides easy cookie management.
- Session and cache management: Express also enables session management and cache management.

✓

MongoDB

MongoDB is an agile and very scalable NoSQL database. The name Mongo comes from the word "humongous," emphasizing the scalability and performance MongoDB provides. It is based on the NoSQL document store model, which means data is stored in the database as basically JSON objects rather than as the traditional columns and rows of a relational database.

MongoDB provides great website backend storage for high-traffic websites that need to store data such as user comments, blogs, or other items because it is quickly scalable and easy to implement.

Node.js supports a variety of database access drivers, so the data store can easily be MySQL or some other database. However, the following are some of the reasons that MongoDB really fits in the Node.js stack well:

Document orientation: Because MongoDB is document oriented, data is stored in the database in a format that is very close to what you deal with in both server-side and client-side scripts. This eliminates the need to transfer data from rows to objects and back.

High performance: MongoDB is one of the highest-performing databases available. Especially today, with more and more people interacting with websites, it is important to have a backend that can support heavy traffic.

High availability: MongoDB's replication model makes it very easy to maintain scalability while keeping high performance.

High scalability: MongoDB's structure makes it easy to scale horizontally by sharing the data across multiple servers.

AngularJS

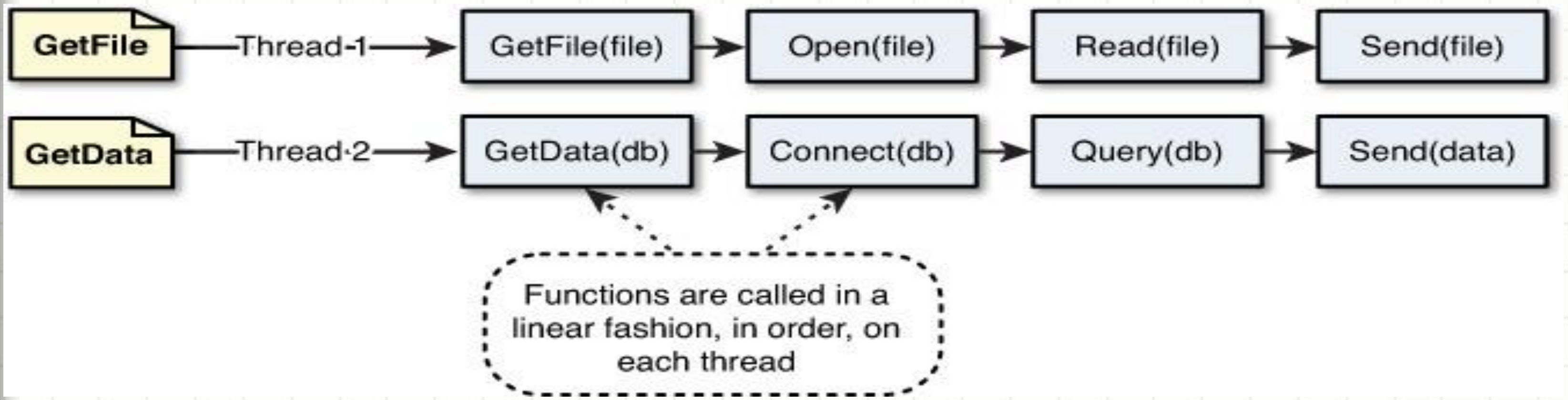
- AngularJS is a client-side framework developed by Google. It provides all the functionality needed to handle user input in the browser, manipulate data on the client side, and control how elements are displayed in the browser view. It is written in JavaScript, with a reduced jQuery library. The theory behind AngularJS is to provide a framework that makes it easy to implement web applications using the MVC framework.
- Other JavaScript frameworks could be used with the Node.js platform, such as Backbone, Ember, and Meteor. However, AngularJS has the best design, feature set, and trajectory at this writing. Here are some of the benefits AngularJS provides:
- Data binding: AngularJS has a very clean method for binding data to HTML elements, using its powerful scope mechanism.

Angular JS continued…

- Image Extensibility: The AngularJS architecture allows you to easily extend almost every aspect of the language to provide your own custom implementations.
- Image Clean: AngularJS forces you to write clean, logical code
- Reusable code: The combination of extensibility and clean code makes it very easy to write reusable code in AngularJS. In fact, the language often forces you to do so when creating custom services.
- Support: Google is investing a lot into this project, which gives it an advantage over similar initiatives that have failed.
- Compatibility: AngularJS is based on JavaScript and has a close relationship with jQuery. This makes it easier to begin integrating AngularJS into your environment and reuse pieces of your existing code within the structure of the AngularJS framework.

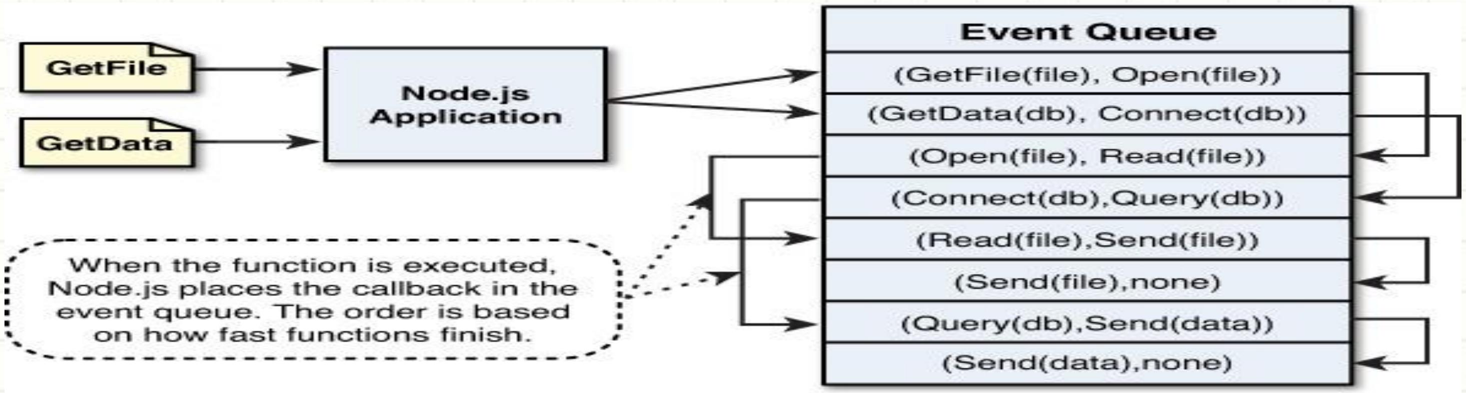
Comparing Callbacks and Threaded Models.

- In the traditional threaded web model, a request comes to a webserver and is assigned to an available thread. The handling of work for that request continues on that thread until the request is complete and a response is sent.
- The following figure illustrates the threaded model processing two requests, GetFile and GetData. The GetFile request opens the file, reads the contents, and then sends the data back in a response. All this occurs in order on the same thread. The GetData request connects to the database, queries the necessary data, and then sends the data in the response.



Comparing Callbacks and Threaded Models.

- Instead of executing all the work for each request on individual threads, Node.js adds work to an event queue and then has a single thread running an event loop pick it up. The event loop grabs the top item in the event queue, executes it, and then grabs the next item. When executing code that is longer lived or has blocking I/O, instead of calling the function directly, it adds the function to the event queue along with a callback that will be executed after the function completes. When all events on the Node.js event queue have been executed, the Node.js application terminates.
- The below fiure illustrates how Node.js handles the GetFile and GetData requests. It adds the GetFile and GetData requests to the event queue. It first picks up the GetFile request, executes it, and completes by adding the Open() callback function to the event queue. Then it picks up the GetData request, executes it, and completes by adding the Connect() callback function to the event queue. This continues until there are no callback functions to be executed.


GetFile
GetData
Node.js Application
Event Queue
(GetFile(file), Open(file))
(GetData(db), Connect(db))
(Open(file), Read(file))
(Connect(db),Query(db))
(Read(file),Send(file))
(Send(file),none)
(Query(db),Send(data))
(Send(data),none)
When the function is executed, Node.js places the callback in the event queue. The order is based on how fast functions finish.

Comparing Callbacks and Threaded Models.

- To help you understand how events work in Node.js compared to traditional threaded webservers, consider an example of having different conversations with a large group of people at a party. You are acting the part of the webserver, and the conversations represent the work necessary to process different types of web requests. Your conversations are broken up into several segments with different individuals. You end up talking to one and then another, then back to the first and then a third, back to the second, and so on.
- The conversation example works well because it has many similarities to webserver processing. Some conversations end quickly (for example, a simple request for a piece of data in memory). Others are broken up in several segments that go back and forth between individuals (much like the more complex server-side conversations). Still others have long breaks while waiting for the other person to respond (like blocking I/O requests to the file system, database, or remote service).
- Using the traditional webserver threading model in the conversation example sounds great at first because each thread acts like a clone of you. The threads/clones can talk back and forth with each person, and it almost seems like you can have multiple conversations simultaneously. There are two problems with this model.

- The first problem is that you are limited by the number of clones. If you have only five clones, then to talk with a sixth person, one clone must completely finish its conversation. The second problem is that there is a limited number of CPUs/brains that the threads/clones must share. This means the clones sharing the same brain have to stop talking/listening while other clones are using the brain. You can see that there really isn't a very big benefit to having clones when they freeze while the other clones are using the brain.
- The Node.js event model acts more similarly to a real-life conversation than does the traditional webserver model. First of all, Node.js applications run on a single thread, meaning there is only one of you—no clones. Each time a person asks you a question, you respond as soon as you can. Your interactions are completely event driven; you move naturally from one person to the next. Therefore, you can have as many conversations going on at the same time as you like and bounce between individuals. Second, your brain is always focused only on the person you are talking to since you aren't sharing it with clones.
- So what happens when someone asks you a question that you have to think about for a while before responding? You can still interact with others at the party while trying to process that question in the back of your mind. That processing may impact how fast you interact with others, but you are still able to communicate with several people while processing the longer lived thought. This is similar to how Node.js handles blocking I/O requests using the background thread pool. Node.js hands blocking requests over to a thread in the thread pool so that they have minimal impact on the application-processing events.

Summary

- Define your challenges
  - Technological as well as personal
- Set realistic expectation
  - Mastery is not achieved overnight
- Keep your eye on the goal
  - Mentorship programs

Resources

- <Intranet site text here>
  <hyperlink here>

- <Additional reading material text here>
  <hyperlink here>

- This slide deck and related resources:
  <hyperlink here>

Questions?

Appendix