NodeJS

Contents

- Introduction, Foundation and Node Project
- Introduction to NodeJS, NPM and In-Built Packages
- ECMAScript 6 (ES6) Fundamentals
- Callback Functions, Events and Errors
- Working with asynchronous programming
- Building a HTTP Server with Node.JS using HTTP APIs
- Socket IO
- Using Express JS
- NodeJS with Database integration

Introduction, Foundation and Node Project

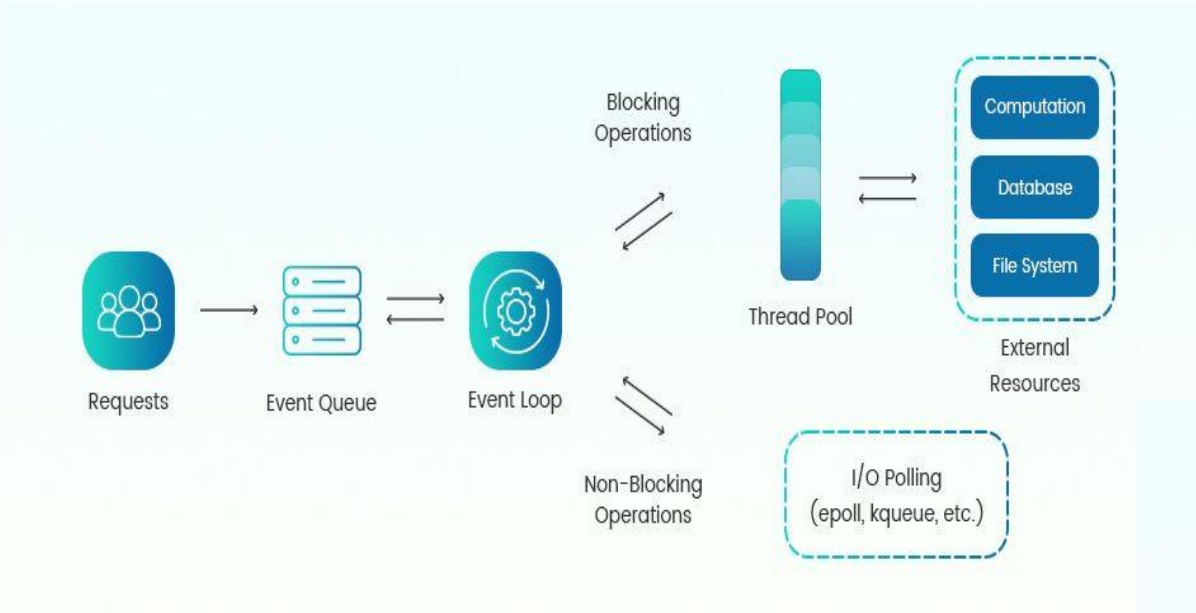
- Node.js is an open-source and cross-platform **JavaScript runtime environment**.
- Node.js runs the **V8 JavaScript engine**, the core of Google Chrome, outside of the browser.
- **Cross-platform** (works on Windows, Mac, Linux, etc.)
- **Event-driven**: Handles tasks efficiently by responding to events as they happen.
- **Asynchronous:** Doesn't block on I/O operations, allowing it to handle concurrent connections at once.
- **Uses JavaScript:** Enables developers who already know JavaScript to write server-side code.
- **Single language:** Developers can use JavaScript for both front-end and back-end.
- **Fast and scalable:** Well-suited for building data-intensive applications. Handles high volumes of concurrent connections
- **Large community:** Extensive libraries and frameworks available.
- **Real-time applications:** Well-suited for building applications with constant updates (chat, streaming)

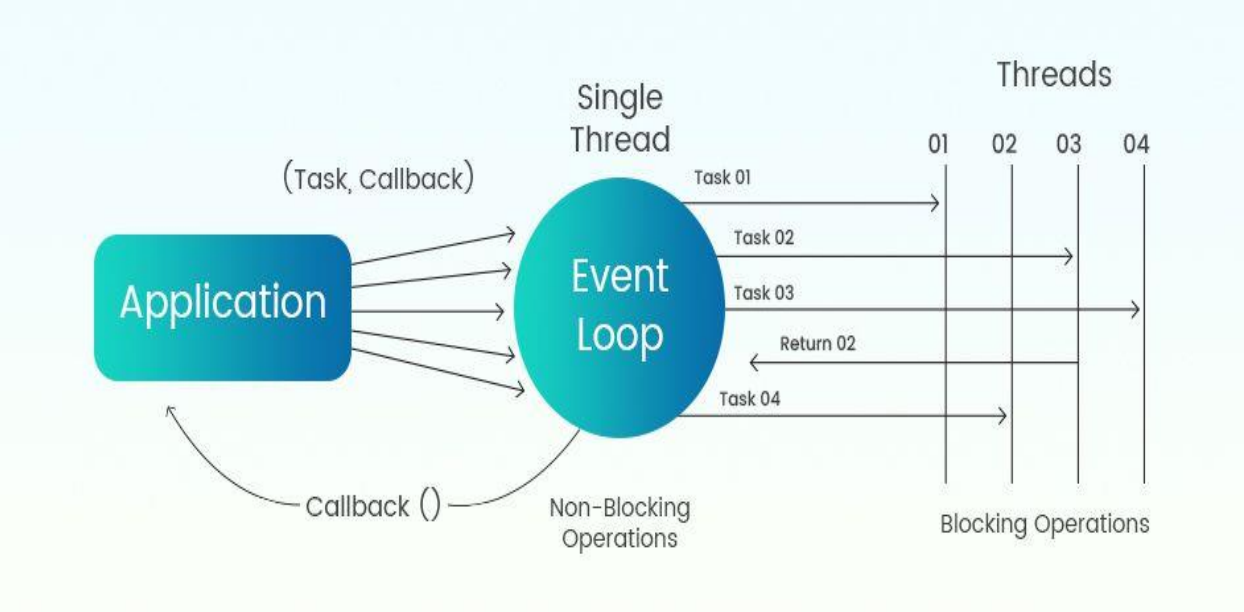
Use cases:

- **Web Applications:** Many popular web applications are built with Node.js, including single-page applications (SPAs) known for their responsiveness and real-time features.
- **Real Time Applications:** Node.js excels at handling real-time communication due to its event-driven architecture. This makes it a popular choice for chat applications, collaboration tools, and other applications requiring constant back-and-forth communication.
- **Microservices (Small, independent services):** Node.js is well-suited for building microservices, which are small, independent services that work together to create a larger application. This approach allows for faster development and easier maintenance.
- **Developer Tools:** Node.js is used to create various developer tools, including build tools, testing frameworks, and code linters. Popular front-end development tools like create-react-app leverage Node.js for functionalities like hot module reloading during development.
- **Data Intensive Applications:** Node.js can be used to build - Real time analytics dashboards, data streaming applications and web scraping applications.

Introduction, Foundation and Node Project

Node.js Architecture:





Introduction, Foundation and Node Project

Node Project

- A Node.js project refers to a collection of files and folders that together create a functioning Node.js application. It encompasses everything from core application logic written in JavaScript to configuration files and potentially static assets.
- Components:
  - **Code:** The heart of project lies in the JavaScript files containing application logic. These files might be structured into folders like models, controllers, and services depending on your chosen architecture.
  - **Dependencies:** Most Node.js projects rely on additional functionalities provided by external libraries and modules. These are typically managed through npm (Node Package Manager) and stored in the node_modules folder.
  - **Configuration:** The package.json file is a crucial configuration file that specifies project dependencies, scripts, and other settings. It acts as a blueprint for the project.
- Optional Elements: Depending on project's needs, there might be additional elements like
  - **Static Assets:** Folders containing static files like images, CSS stylesheets, or HTML files.
  - **Tests:** A dedicated folder holding unit and integration tests written with frameworks like Jest.

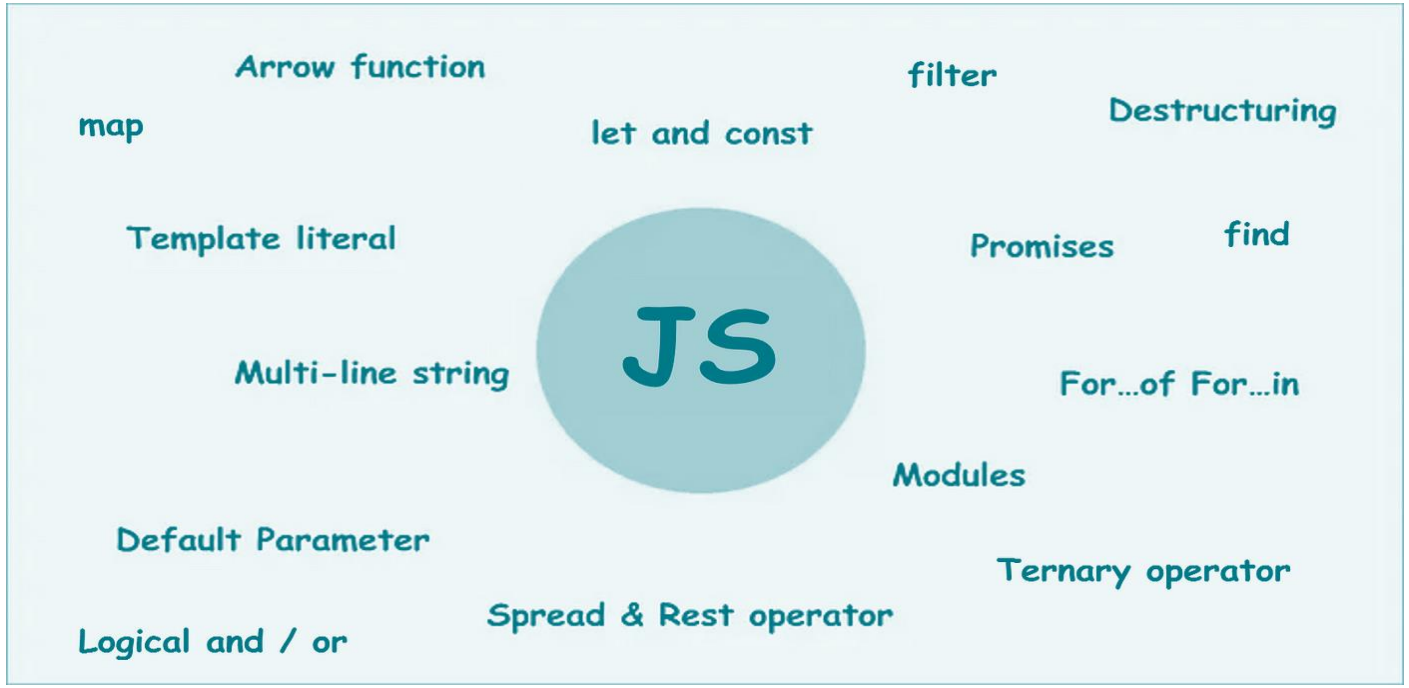
Introduction, Foundation and Node Project

Node Project

- **Structure:** While there's no single enforced structure, common conventions include:
  - **node_modules:** This folder, managed by npm, stores all the external dependencies.
  - **package.json:** The central configuration file for the project.
  - **src or app:** The core application code resides here, often further organized into subfolders based on functionality (models, controllers, etc.).
  - **public (optional):** Stores static assets served directly by the server.
  - **tests (optional):** Holds unit and integration tests.
- **Creating a Node.js Project:**
  - **Creating a Node.js project involves:**
    1. Setting up a project directory.
    2. Initializing a project with ***npm init***
    3. Installing dependencies (if needed) using ***npm install***
    4. Writing your application logic in JavaScript files.
    5. Running the application with ***node index.js***

ECMAScript 6 (ES6) Fundamentals

- ES6 or ECMAScript 2015 is the 6th version of the ECMAScript programming language. ECMAScript is the standardization of JavaScript which was released in 2015 and subsequently renamed as ECMAScript 2015.
- ECMAScript is the specification defined in ECMA-262 for creating a general-purpose scripting language. It is a standardization for creating a scripting language.
- JavaScript is a general-purpose scripting language that confirms the ECMAScript specification. It is an implementation that tells how to use a scripting language.



ECMAScript 6 (ES6) Fundamentals

- ES6 Modules
  - ES6 modules are a way to organize code into reusable, self contained units. They promote code modularity, making applications more maintainable, scalable and easier to collaborate.

  - Key Concepts in ES6 Modules:
    - **Module File:** Typically a .js file containing code to organize or reuse.
    - **export** statement: Use the export statement to make specific variables, functions or classes from the module accessible to other modules.
    - **import** statement: Use the import statement to import (use) the exported elements from another module in the current module.

  - Benefits of ES6 Modules
    - Improved Code Organization , Reduced Global Namespace Pollution, Easier Testing and Collaboration

Import Operation	Syntax
Import entire module	import * as name from ‘module-name’
Import default export from a module	import name from ‘module-name’
I**mport single export from a module**	import { name } from ‘module-name’
Import multiple exports from a module	import { name1 , name2 } from ‘module-name’
Import a module for side effects only	import ‘./module-name’

- Classes
  - ES6 introduced classes in JavaScript. Classes in JavaScript can be used to create new Objects with the help of a constructor, each class can only have one constructor inside it.

ECMAScript 6 (ES6) Fundamentals

- Arrow Functions
  - Arrow functions, introduced in ECMAScript 2015 (ES6), are a concise way to write JavaScript functions. They provide a shorter syntax compared to traditional function expressions and function declarations. They offer several advantages over traditional function declarations:
    - **Simpler Syntax:** Arrow function use **a => (fat arrow)** instead of the function keyword, making them shorter to write.
    - **Lexical this binding:** Unlike regular functions, arrow functions inherit the **this** value from their surrounding scope which helps in avoiding common pitfalls with **this** binding in callbacks. This can be beneficial when dealing with event listeners or callback functions where you want to maintain the context of the enclosing function. ***(For complex functions or those requiring their own this binding, traditional functions might be more suitable.)***
    - **Implicit return for single-line expressions:** For functions with a single expression in their body, the return statement is implicit, further reducing verbosity.
  - Key points to remember about the arrow functions:
    - They cannot be used as constructor with **new**. They are typically anonymous (without a name), but can be assigned to variables. They don't have their own bindings to **arguments** or **super**.

ECMAScript 6 (ES6) Fundamentals

- Template Literals
  - Template literals (also known as template strings) are a powerful way to create strings.
  - Introduced in ES6 (ECMAScript 2015), they offer several advantages over traditional strings enclosed in quotes (single or double).
    - **Backticks for definition:** Template literals are defined using backtick characters (`) instead of quotes. This allows writing multi-line strings without the need for concatenation**.**
    - **Multiline Strings:** template literals can span multiple lines naturally. This makes the code cleaner and easier to read, especially for longer strings.
    - **String interpolation:** This is the most significant feature of template literals. Expressions can be directly embedded using **${expression}** syntax.
- Scoping using let and const keywords:
  - Unlike var which is a function scoped, let and const declarations are block-scoped. This means they are only accessible within the block of code where they are declared.
  - The value of a variable declared with const cannot be reassigned after its initial assignment. It acts as a constant.

ECMAScript 6 (ES6) Fundamentals

- Spread Syntax and Rest Parameters
  - Spread syntax (...) and rest parameters (...) in ES6 (ECMAScript 2015) are two powerful features that deal with iterables (like arrays and strings) but serve different purposes:
- **Spread Syntax – Spreading out the elements**
  - **Expands iterables:** Used to unpack the contents of an iterable (array, string, object) into individual elements within array literals, function calls, or object initializers.
- **Rest Parameters – Gathering the rest into an array**
  - **Collects elements: Used within function parameters to capture an indefinite number of arguments as an array.**

	Location	**Purpose**
Spread Syntax	Array literals, function calls, object literals	Expands iterables into individual elements
Rest Parameters	Function parameter list	Captures remaining arguments as an array

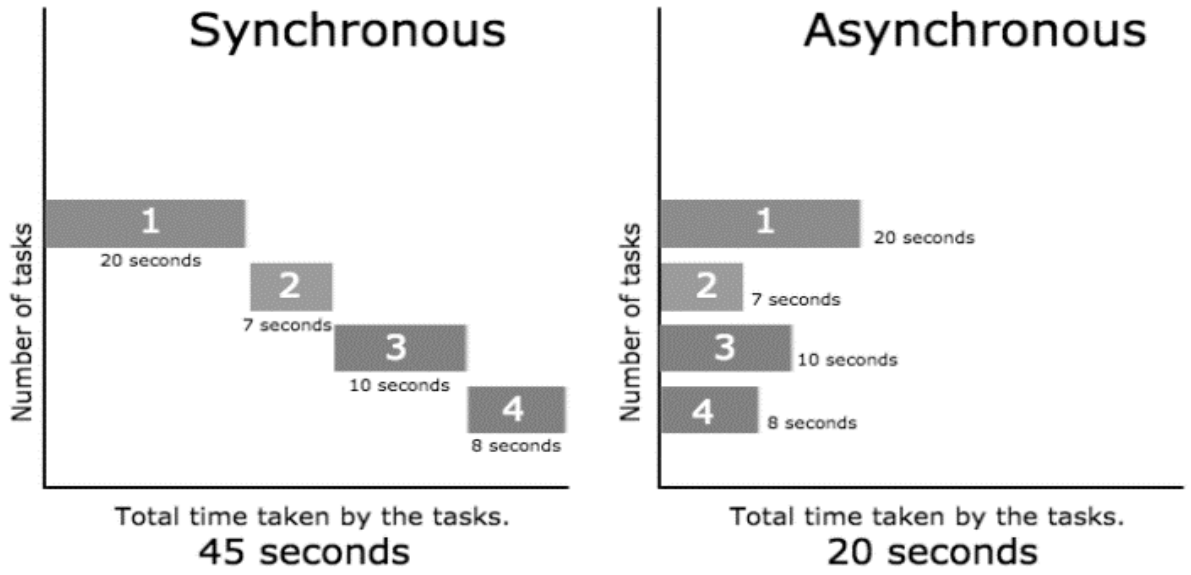
ECMAScript 6 (ES6) Fundamentals

- **Destructuring**
  - Destructuring assignment is a feature introduced in ES6 (ECMAScript 2015) that allows to unpack values from arrays or properties from objects into distinct variables in a more concise and readable way.
  - It's a shorthand way to extract specific data from these data structures in a cleaner and more readable manner. Destructuring provides a concise and elegant way to extract and manage data from arrays and objects in JavaScript.
  - **Breakdown of destructuring**
    - Array Destructuring: Extracts elements from an array and assigns them to variables.
    - Object Destructuring: Extracts properties from an object and assigns them to variables.

  - **Benefits of Destructuring:**
    - Improves code readability by making variable assignments more explicit.
    - Simplifies working with complex data structures like nested objects.
    - Can be used with default values to handle missing properties.

Working with Asynchronous Programming

- **Asynchronous Programming**
  - Node.js is single-threaded, but it excels at handling concurrent requests through asynchronous programming. This means tasks can run independently without blocking the main thread. It's particularly useful for I/O bound operations like network requests, file access, or database interactions.
  - Unlike synchronous programming where a task must finish before the next one starts, asynchronous tasks can be initiated and then continue executing other code while waiting for the asynchronous operation to complete.



Working with Asynchronous Programming

- **Benefits of Asynchronous Programming**
  - Improved Performance: Because of non-blocking of main thread, Node.js can handle multiple requests concurrently, leading to faster response times and better resource utilization.
  - Scalability: Asynchronous applications can efficiently manage a high volume of concurrent requests, making them suitable for building high-performance web servers and real-time applications.

- **Approaches for asynchronous programming in Node.js:**
  - **Callbacks:** A function passed to an asynchronous function to handle the result when the operation completes. Promises: An object representing the eventual completion (or failure) of an asynchronous operation. Promises provide a cleaner way to handle the flow of asynchronous code.
  - **Async/Await:** Built on top of Promises that makes asynchronous code look more synchronous.

Working with Asynchronous Programming

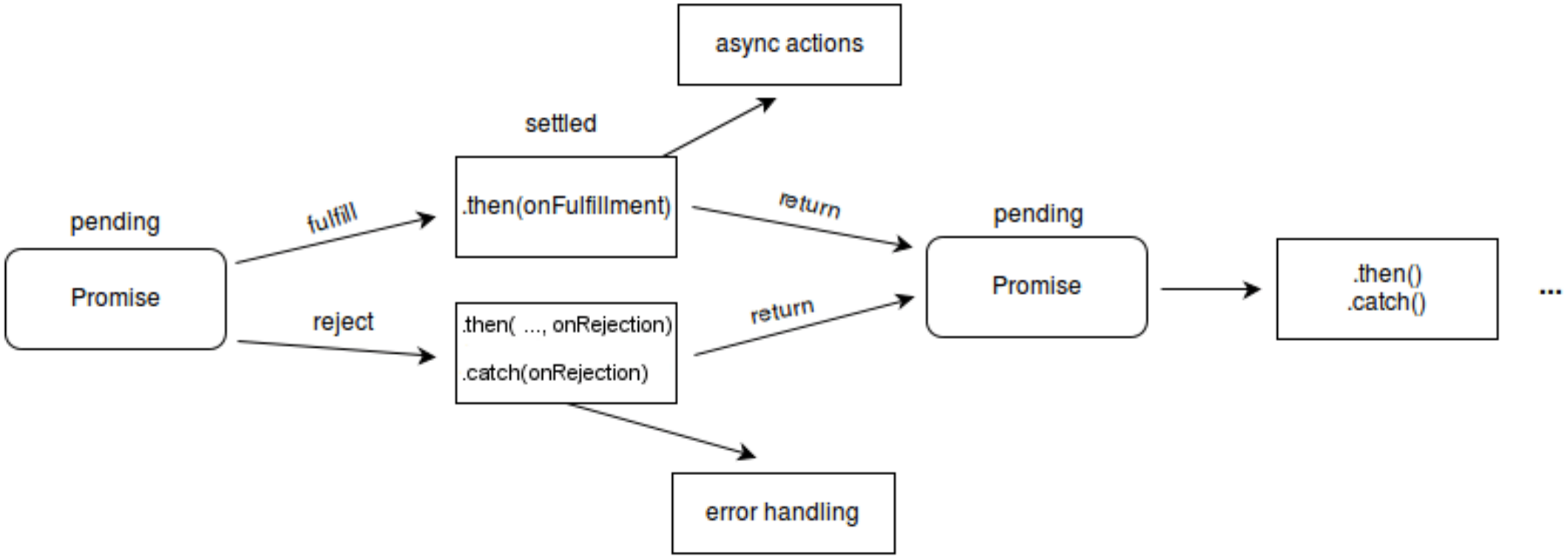
- **Callback Function**
  - The function that is passed as an argument inside of another function is called a callback function.
  - The benefit of using a callback function is that you can wait for the result of a previous function call and then execute another function call. The callback function is helpful when you have to wait for a result that takes time. For example, the data coming from a server because it takes time for data to arrive.
  - **Synchronous callbacks**
    - A synchronous callback is executed during the execution of the high-order function that uses the callback.
  - **Asynchronous callbacks**
    - An asynchronous callback is executed after the execution of the high-order function that uses the callback.
    - Asynchronicity means that if JavaScript has to wait for an operation to complete, it will execute the rest of the code while waiting.

Working with Asynchronous Programming

- **Promise**
  - The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.
  - A Promise is a proxy for a value not necessarily known when the promise is created. It allows you to associate handlers with an asynchronous action's eventual success value or failure reason.
  - This lets asynchronous methods return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a promise to supply the value at some point in the future.
  - A Promise is in one of these states:
    - pending: initial state, neither fulfilled nor rejected.
    - fulfilled: meaning that the operation was completed successfully.
    - rejected: meaning that the operation failed.

Working with Asynchronous Programming

- **Promise**



Working with Asynchronous Programming

- **Async-Await**
  - The async and await keywords enable asynchronous, promise-based behavior to be written in a cleaner style, avoiding the need to explicitly configure promise chains.
  - async and await make promises easier to write.
  - async makes a function return a Promise and await makes a function wait for a Promise

Callback Hell

- Callback hell refers to the situation where multiple nested callbacks make the code hard to read and maintain.
- This often occurs in asynchronous programming, particularly in JavaScript, when several operations that depend on the results of previous operations.
- To avoid callback use one of the following
  - Promises
  - Async-Await

Node.js Events

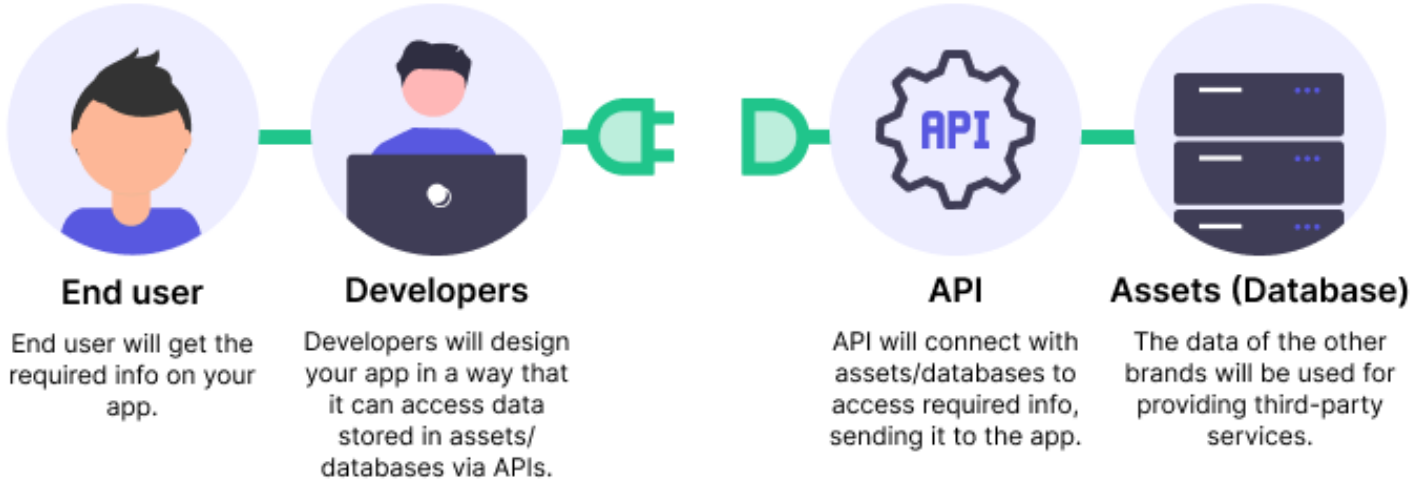
- Node.js events are an essential part of the Node.js platform,
- It provides a way to handle asynchronous operations. The events module in Node.js, which is a core module, is used to create, fire, and listen for your own events.
- **Key Concepts:**
  - **Event Emitter:** This is the core of Node.js event system. It is a class that is used to bind events and listener functions. All objects that emit events are instances of the EventEmitter class.
  - **Event Listeners:** Functions that are called when a particular event is fired.
  - **Binding and Emitting Events:** Events can be bound to a specific function using the .on() method and can be emitted using the .emit() method.
- Events are used extensively in Node.js for many internal modules like HTTP, FS (File System), and Streams.
- While reading a file using the FS module, the event-driven nature of Node.js is used with events like data, end, and error.
- The events module and the event-driven architecture of Node.js make it highly scalable and suitable for real-time applications such as chat applications, live updates, and handling multiple simultaneous connections effectively.

Node.js Errors

- Error in node.js can stem from various sources like syntax errors, runtime errors, or issues with asynchronous operations.
- **Common Types of Node.js Errors**
  - **Syntax Errors:** Occur when there are mistakes in the code syntax.
  - **Runtime Errors:** Happen during the execution of the code, like referencing undefined variables.
  - **Operational Errors:** Result from failed operations, such as I/O failures or network issues.
  - **Logical Errors:** Stem from incorrect logic implemented in the code.
- **Handling Errors in Nodejs:**
  - Try-Catch for synchronous code
  - For asynchronous operations using callbacks, handle errors by passing them to the callback function
  - For promises, use ‘.catch’ to handle errors
  - For async/await, use try-catch within an async function
  - Custom error class can be created to handle specific type of errors.

API

- API stands for application programming interface, which is a set of definitions and protocols for building and integrating application software.
- In the context of APIs, the word Application refers to any software with a distinct function. Interface can be thought of as a contract of service between two applications. This contract defines how the two communicate with each other using requests and responses. API documentation contains information on how developers are to structure those requests and responses.



Different Types of APIs

APIs are classified both according to architecture and scope of use.

- Private APIs: These are internal to an enterprise and only used for connecting systems and data within the business.

- Public APIs: These are open to the public and may be used by anyone. There may or not be some authorization and cost associated with these types of APIs.

- Partner APIs: These are only accessible by authorized external developers to aid business-to-business partnerships.

- Composite APIs: These combine two or more different APIs to address complex system requirements or behaviors.

Securing API

All APIs must be secured through proper authentication and monitoring. The two main ways to secure REST APIs include:

- **Authentication tokens:** These are used to authorize users to make the API call. Authentication tokens check that the users are who they claim to be and that they have access rights for that particular API call. For example, when logging to the email server, the email client uses authentication tokens for secure access.

- **API keys:** API keys verify the program or application making the API call. They identify the application and ensure it has the access rights required to make the particular API call. API keys are not as secure as tokens but they allow API monitoring in order to gather data on usage.

Using an API

The steps to implement a new API include:

- **Obtaining an API key**. This is done by creating a verified account with the API provider.
- **Set up an HTTP API client:** This tool allows you to structure API requests easily using the API keys received.
- If not having an API client, one can try to structure the request in the browser by referring to the API documentation.
- Start writing the code with the API Syntax

NodeJS Request Module

- The request module is used to make HTTP calls. It is the simplest way of making HTTP calls in node.js using this request module.
- installation:
  - npm install request
- The first argument to request can either be a URL string, or an object of options. Common option available are:
  - url: The destination URL of the HTTP request
  - method: The HTTP method to be used (GET, POST, DELETE, etc)
  - headers: An object of HTTP headers (key-value) to be set in the request
  - form: An object containing key-value form data

Fetch API

- The Fetch API provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses. It also provides a global fetch() method that provides an easy, logical way to fetch resources asynchronously across the network.
- The fetch() method starts the process of fetching a resource from a server.
- The fetch() method returns a Promise that resolves to a Response object.
- The fetch() method in JavaScript is used to request to the server and load the information on the webpages. The request can be of any APIs that return the data of the format JSON or XML. This method returns a promise.
- Parameters: This method accepts two parameters as mentioned above and described below:
    - URL: It is the URL to which the request is to be made.
    - Options: It is an array of properties. It is an optional parameter.
- Return Value: It returns a promise whether it is resolved or not. The return data can be of the format JSON or XML. It can be an array of objects or simply a single object.

Building a HTTP Server with Node.JS and HTTP APIs

- HTTP is a protocol, which is used for fetching resources like HTML documents and more.
- It is the most common foundation for any data exchange on the Web and is basically a client-server protocol.
- Node.js supports creating web servers out of the box, which means a Node js web server can be directly created from a basic command set.
- A web server receives HTTP requests from a client, like browser, and provides an HTTP response, like an HTML page or JSON from an API.
- Steps:
  - Creating a Basic HTTP Server
  - Managing API routes using an HTTP Request Object

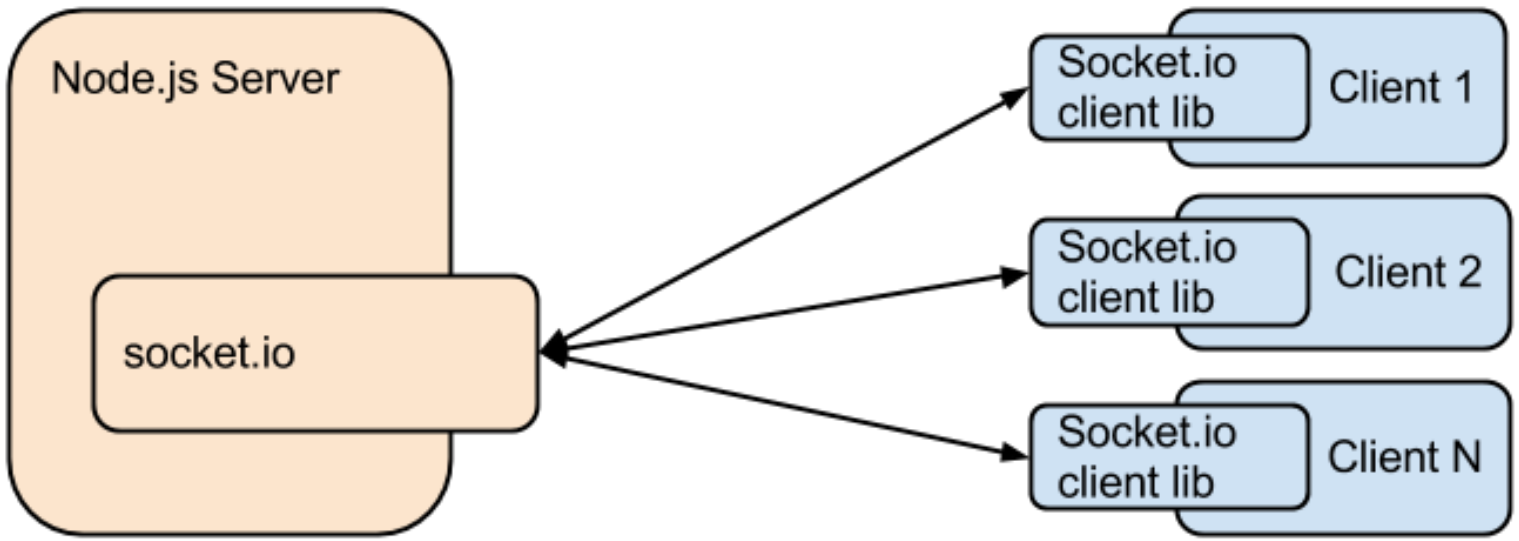
Express

- Express is a node js web application framework that provides features for building web and mobile applications.
- It is a layer built on the top of the Node js that helps manage servers and routes.
  - npm install express
- **Functionalities:**
  - Routing: Express allows you to define routes that map URLs to specific functions that handle requests.
  - Middleware: Middleware are functions that is injected into application's request-response cycle. They can perform various tasks like logging requests, parsing data, or enabling authentication.
  - Templating: Express integrates with various templating engines like Pug or EJS. These engines help you generate HTML dynamically based on data.
- **Use Cases:**
  - Building RESTful APIs: A common use case for Express is creating APIs.
  - Web applications: Express can be used to build full-fledged web applications.
  - Hybrid applications: Express can also be used as a backend for single-page applications (SPAs) that use JavaScript frameworks like React or Angular on the frontend.

Socket.IO

- Socket.IO is a library in Node.js that enables real-time, bidirectional communication between clients (such as web browsers) and servers.
- It is particularly useful for applications that require instant updates or live data streaming, like chat applications, live notifications, real-time analytics, and collaborative tools.
    - npm install socket.io
- Key Features:
    - **Real-time Communication:** It allows for real-time data exchange between client and server with minimal latency.
    - **Cross-platform:** Works across different platforms, including web browsers and mobile apps.
    - **Automatic Reconnection:** If the connection drops, Socket.IO automatically tries to reconnect.
    - **Binary Support:** Allows transmission of binary data (like images and files) efficiently.
    - **Namespaces:** Helps in segmenting the connection into different channels for better organization.
    - **Rooms:** Allows grouping of sockets in rooms, making it easier to broadcast messages to multiple clients.
    - **Fallback Mechanism:** If WebSockets are not supported, Socket.IO can fall back to other communication methods like long polling.

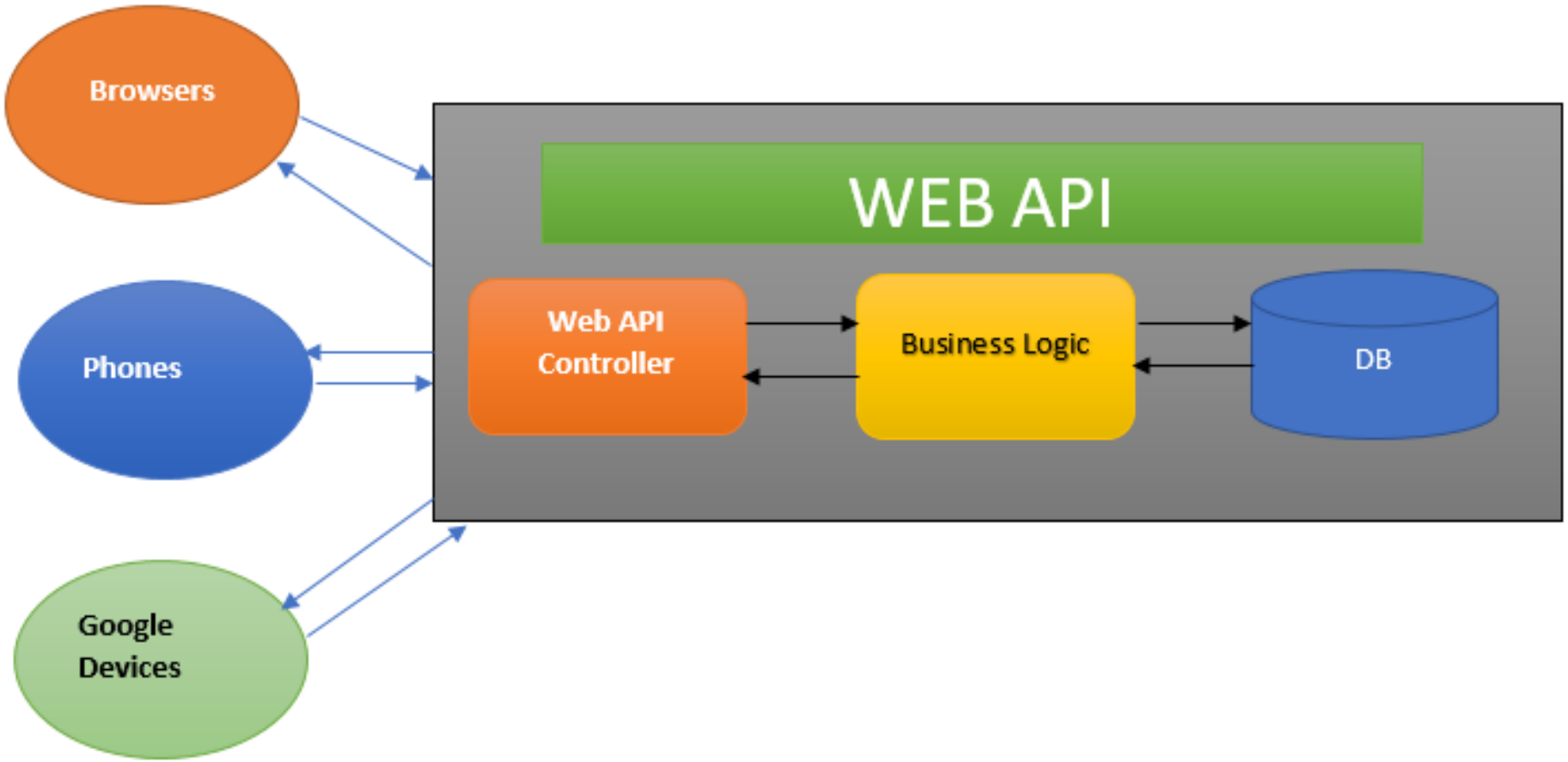
Socket.IO


Node.js Server
socket.io
Socket.io client lib
Client 1
Socket.io client lib
Client 2
Socket.io client lib
Client N

REST API

- REST stands for Representational State Transfer. It is an architectural style that defines a set of rules to create Web Services. REST defines a set of functions like GET, PUT, DELETE, etc. that clients can use to access server data. Clients and servers exchange data using HTTP.
- In a client-server communication, REST suggests creating an object of the data requested by the client and send the values of the object in response to the user.
- The main feature of REST API is statelessness. Statelessness means that servers do not save client data between requests. Client requests to the server are similar to URLs you type in your browser to visit a website. The response from the server is plain data, without the typical graphical rendering of a web page.
- For example, if the user is requesting for cab booking in certain place and time, then an object of that service is created on the server-side. So, over here, the state of the object is send. Therefore, REST is known as Representational State Transfer.
- The architectural style of REST helps in leveraging the lesser use of bandwidth to make an application more suitable for the internet. It is often regarded as the “language of the internet” and is completely based on the resources.

REST API


Browsers
Phones
Google
Devices
WEB API
Web API
Controller
Business Logic
DB

NodeJS Database Integration

- To integrate a MySQL database with your Node.js and Express application, use a MySQL client for Node.js. A popular choice is mysql2, which supports both callbacks and promises.
  - **npm install mysql2**
- To use MongoDB with Node.js and Express, use mongodb or mongoose library. mongoose is an Object Data Modeling (ODM) library that provides a straightforward API for interacting with MongoDB.
  - **npm install express mongoose body-parser**
  - body-parser is middleware used in Express applications to parse incoming request bodies in a middleware before your handlers, making it easier to access and work with data in req.body. It is used for:
    - **Parsing JSON:** By default, Express does not parse incoming JSON request bodies. To handle JSON data sent in a POST or PUT request, middleware is required to parse that data. body-parser provides this functionality.
    - **Parsing URL-encoded Data:** Similarly, if client sends form data (application/x-www-form-urlencoded), body-parser can parse that into a JavaScript object, making it accessible in req.body.
    - **Simplifies Request Handling:** Without body-parser (or built-in body parsing middleware in newer versions of Express), manually handling and parsing of the data needs to be done.

Thank You