MEN

MongoDB

Express

NodeJS

**NodeJS – Runtime Environment / Platform**

**Run java code within the browser – applet (plugin)**

**Run .Net complaint within the browser – Silverlight (plugin)**

**Run javascript within the browser – by default.**

Machine

Java JVM – Java Virtual Machine

C#, VB.Net CLR – Common Language Runtime

C, C++ C Runtime

Javascript **Node**

Conceptually there are 3 things that get downloaded.

**Node.exe – Runtime / platform.**

**NodeJS – SDK, Set of classes etc.**

**NPM – Node Package Manager.**

Managing packages which you need either at client side or at server side.

Client-side packages – Angular

Server-Side packages – Mongoose

ORM – Object Relational Mapping

Sequilize, Hibernate, Entity Framework – ORM

NoSQL – MongoDB - ODM - Mongoose

**Bower** – managing client-side packages – flat dependencies.

**Yarn** – Both side packages same as npm.

Node.js is an open source, cross-platform. Javascript runtime environment that allows you to run Javascript code outside of a web browser.

**Javascript Runtime** – Built on V8 Javascript engine from Google Chrome.

**Asynchronous and Event-Driven** – Node.js uses non-blocking, event driven architecture, making it ideal for heavy I/O operations like reading files, or handling network requests.

**Single Threaded** – Node.js can handle many connections simultaneously thanks to its event loop and asynchronous nature.

**Cross Platform** – Node.js runs on various platforms, including windows, mac, linux, unix etc.

**Use Cases**

* Web Servers
* Real Time Applications – chat app, online game application
* Command Line Tools – create command line tools and scripts.
* Stock Trader’s dashboard
* Data Streaming
* System Monitoring Dashboard

Commands

To get the version

**node -v / node --version**

**npm -v / npm –version**

List all the packages installed locally.

**npm ls**

List all the packages installed globally (wherever node is being installed)

**npm ls -g**

To install any of the package

**npm i <package-name> - this will install that package locally.**

**npm i <package-name> -g – this will install that package globally.**

**npm i** [**express@1.2.3**](mailto:express@1.2.3) **– install specific version locally.**

**npm i express@latest -g – install latest version globally.**

Explicitly creating **package.json**

**npm init**

{

  "name": "demo\_app",

  "version": "1.0.0",

  "main": "index.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1"

  },

  "keywords": [

    "abhijeet",

    "node"

  ],

  "author": "Abhijeet D Gole",

  "license": "ISC",

  "description": ""

}

**npm init -y**

{

  "name": "day-i",

  "version": "1.0.0",

  "main": "index.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1"

  },

  "keywords": [],

  "author": "",

  "license": "ISC",

  "description": ""

}

The **package-lock.json** file is an automatically generated file in Node.js projects that provides a detailed deterministic record of the dependency tree.

**Purpose of this file**

* **Dependency locking** – It locks down the specific versions of every installed package, ensuring that subsequent install generates identical dependency trees, regardless of intermediate updates.
* **Version Consistency** – Ensures that all the developers working on the project as well as CI/CD systems use the exact same version of dependencies, avoiding “it works on my machine” issue.
* **Improved Installation Speed**, - Optimizes dependency installation by storing a flat node\_modules structure.

**npm I / npm install** – will get the information from package.json about runtime dependencies and development dependencies and install them.

**npm un / uninstall <package-name>**

**npm outdated – will check for the outdated versions of packages.**

**npm update – will update the package to the latest version.**

Any .js file is a module from node perspective.

To use one module in another we need to export the things and we need to import the things.

At Node i.e. server side we uses CommonJS Module Format

Way to export is **module.exports**

Way to import is **require()** function

// Anonymous exports - single client interface. - only one, the last one would get executed

// Named exports - one Module, many exported things

**Node Modules**

**3 types of node modules**

**Custom Modules** – You create it and acquire it using require () function to use it.

**Core Modules** – Available along with Node installation, you acquire it using require () function to use it.

**Third-Party Modules** – Needs to explicitly install and then you acquire it using require () function to use it.

Course Outline

**NodeJS (Express + MongoDB)**

* What is Node.js?
* Why do you need Node.js?
* Node.js as a Platform
* Node – Single Threaded Event Driven
* NPM, Bower, Yarn(overview)
* Node.js Module System
  + - Core Module,
    - Third party Module
    - Custom Module
    - Module Patterns
* Publishing and Consuming modules
* Scoped and Un-scoped modules

**Core Modules**

* + - Events, Event Emitter
    - Streams
    - File System
    - HTTP Module

**Express – Web Framework**

* Creating Server and Client using Express
* Create Express app with different view engines.
* Express-Generator
* Creating RESTful Service using Express

**MongoDB Basics**

* + - Data Modeling
    - Create Database
    - Drop Database
    - Create Collection
    - Drop Collection
    - Data Types
    - Insert Document
    - Query Document
    - Update Document
    - Delete Document
* Connectivity and CRUD Operations with NoSQL Database – (MongoDB)

**Node Microservices**

* + What Do You Mean by Microservices?
  + Node.js Platform
  + A Few Major Benefits of Using Node.js
  + Create Microservices with Node.js
  + Consuming Microservices through API Gateway
* Express gateway

**Miscellaneous**

HAPI – Along with Express framework we do cover HAPI framework for RESTful Service

**npm** – Node Package Manager

**npx** – Node Package Executer / Runner

npx is a package executer / runner tool that comes with npm (since version 5.2.0). It allows you to execute binaries from npm packages without globally installing them.

npx <package-name>

npx create-react-app <appName>

**Convenience** – Simplifies running commands from npm packages.

**Clean** **Environment** – Keeps your global installations minimal.

**Flexibility** – Allows you to use different versions of packages easily.

**Custom Modules**

**Priority is given to whom**

**Index.js**

**Or file that is specified as an entry point in the packages.json.**

**Third party modules**

**Publishing Custom Modules on npmjs.com**

**Publishing Un-scoped Packages**

**Prepare your package** – Ensure package.json file is correctly configured with all necessary fields like

Name

Version

Description

Main

Scripts

Keywords

Author and

License.

**npm login**

**npm publish.**

To Verify

**npm info <package-name>**

**Scoped Packages**

Naming – These packages include a scope, which is a namespace prefixed with an @ symbol like

**@my-org/my-package e.g. @angular/core, @angular/common etc.**

Publishing – Scoped packages are published under a specific scope, which can be an organization or a user. This helps in organizing packages, especially for larger projects or teams.

Installation

**npm I @scope/package-name.**

**Publishing Scoped Packages**

**Prepare your package** – Ensure package.json file is correctly configured with all necessary fields like

Name – “@your-scope/your—package-name”

Version

Description

Main

Scripts

Keywords

Author and

License.

**npm login**

**npm publish –access public / --access private.**

To Verify

**npm info @your-scope/your-package-name**

**Node Module Versioning**

**Semantic Versioning**

**MAJOR.MINOR.PATCH**

"@zenaug24/zen\_aug\_calc\_square\_cube": "^1.0.0",

"zen\_aug\_calc\_add\_mult": "^1.0.0"

**MAJOR** – Incremented for incompatible API changes.

**MINOR** – Incremented for adding functionality in a backward-compatible manner.

**PATCH** – Incremented for backward-compatible bug fixes.

**Exact Version** – 1.2.3 – uses the exact version.

**Caret (^)** ^1.2.3 – compatible with version 1.2.3, allowing updates that do not change the leftmost non-zero digit (1.2.4, 1.3.0).

**Tilde (~)** ~1.2.3- Compatible with version 1.2.3, allowing updates to the rightmost non-zero digit (e.g. 1.2.4 but not 1.3.0).

Core Modules – Comes along with Node Installation.

OS – Operating System

Fs – file system

Readline

Streams

Events

**Event Loop** – is a fundamental concept in Node.js that allows it to handle asynchronous operations efficiently, despite javascript being single-threaded.

Non-blocking, I/O – The event loop enables Node.js to perform non-blocking I/O operations by offloading tasks to the system kernel whenever possible.

Single-Threaded – The Event Loop allows Node.js to handle multiple operations concurrently by managing asynchronous tasks.

Phases of Event Loop

1. Timers – Executes callbacks scheduled by setTimeout() and setInterval()
2. Pending callbacks – Executes I/O callbacks deferred to the next loop iteration.
3. Idle, Prepare – Internal use only.
4. Poll – Retrieves new I/O events and executed I/O related callbacks.
5. Check – Executes callbacks scheduled by setImmediate()
6. Close Callbacks – Executes close event callbacks such as socket.on(‘close’);

**process.nextTick()** – is a method that allows you to schedule a callback function to be invoked in the next iteration of the event loop, right after the current operation completes.

**Key Features**

**Immediate Execution** – The function passed to the process.nextTick() is executed immediately after the current operation, before any I/O events or timers.

**Priority** – It gives higher priority to the callback function, ensuring it runs before other asynchronous tasks scheduled with setTimeout or setImmediate().

**Use Cases**

**Deferring Execution** – Useful for deferring the execution of a function until the current stack is cleared, ensuring that the function runs as soon as possible.

**Avoiding Recursion limits** – Helps in breaking up long-running operations into smaller chunks to avoid hitting the maximum call stack size.

**Handling Errors** – Allows you to handle errors asynchronously, ensuring that the error handling code runs after the current operation.

N tier / N layer pattern

3 tier / 3 layer

Tier – physical difference – different devices

Layer – logical difference – same device

**3 Tier pattern**

**PL                                           BL                                                                           DAL**

UI                                           Business Logic                                                    Database

UI Logic                                 Classes, Services                                               Classes – Functions – Testing – Yes                                                                                  ORM, ODM – already tested.

UI and UI logic are tightly coupled; hence we need separation of concern.

Hence, we can’t independently do the unit testing of UI logic.

**MVC – Model View Controller**

**Architectural Pattern**

**Two reasons**

* **Separation of Concern**
* **TDD – Test Driven Development**

**MVC – Model View Controller – Client Server Architecture**

**Model – Data Entity**

**View – Representation of the data.**

**Controller – first class citizen, going to handle the request, Controller is going to push model’s data to the view, controller is responsible for handling events associated with controls which are there on the view.**

**The view is as dumb as possible.**

**MVP – Model View Presenter – Thick Client**

**UI and BL are kind of tightly coupled.**

**MVPM – Model View Presenter Model**

**Works same as MVC**

**MVVM – Model View ViewModel**

**Data binding and Commands – MVVM is used.**

**MVC – architectural pattern**

**It is being implemented,**

**In Java                                  Spring, Spring Boot**

**In .net                                   Asp.net MVC**

**In Node                                Express**

**Express** – Fast, unopinionated, minimalist web framework for Node.js

* Web Applications
* APIs
* Performance
* Middleware

RDMBS

NoSQL

MongoDB is a popular NoSQL database that stores data in flexible, JSON-like format called BSON (Binary JSON). Unlike traditional SQL databases that use tables and rows, MongoDB uses collections and documents, making it highly adaptable to various data types and structures.

Unstructured data.

**Benefits**

Flexibility – schema-less design – allows you to store different types of data.

Scalability – built to scale horizontally by distributing data across multiple servers.

Performance – It offers high performance for read and write operations, - efficient indexing and querying capabilities.

Ease of Use – MongoDB’s document model maps naturally to objects in most programming languages.

Community and Eco System – has a strong community and a rich ecosystem of tools and libraries.

Use Case –

Content Management Systems

Real-Time Analytics

Stores user data, comments, and metadata

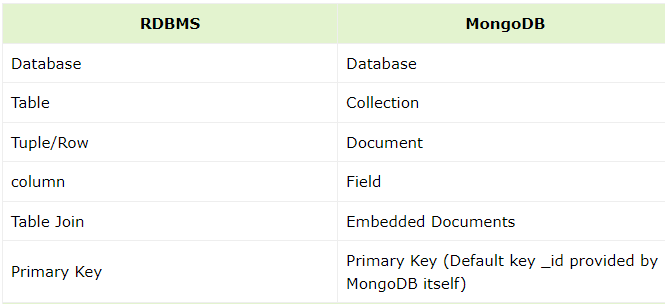
Manage chain data, optimize logistics.

IoT

Database

Collection

Document



Server

RDBMS

Mysql, oracle, sql server

NoSQL

Mongod

Client

RDBMS Mysql, sqlplus / NoSQL – mongosh

To start the mongodb server use the following command.

**mongod --dbpath "D:\MongoDB\_Databases"** (Path of the folder where databases are getting created)

Explicitly download mongo shell

Go to the path of mongosh

And run mongosh

To show all the databases

**show dbs**

To create or use any of the databases.

**use <db-name>**

Drop the database.

**db.dropDatabase();**

To see all the collections from database

**show collections**

Create a collection.

**db.createCollection(‘name’,’option’);**

Drop a collection.

**db.collectionName.drop();**

**Query for document**

find method.

**db.collectionName.find();**

insert a document inside the collection.

**db.collectionName.insert(object of the collection);**

insert multiple records.

**db.collectionName.insert([objects of the collection]);**

**Mongoose – ODM (Object Data Mapping) – NoSQL / RDBMS – ORM**

**Why use mongoose?**

**Mongoose allows you to define schemas for your data models.**

**Mongoose provides built-in validation for your schemas.**

**Mongoose supports middleware.**

**Mongoose makes it easier to manage relationships between different data models.**

**Mongoose provides a powerful query-building API that simplifies the process of querying your mongoDB collections.**

**User – Model**

name – String, required.

email – String, required, unique.

age – number

npx express-generator --view=pug mynewpugapp

for handlebars - npx express-generator --view=hbs mynewhbsapp

db.items.insert([{name:'Item 1',price:123.45},{name:'Item 2',price:456.78}]);

GET ALL

<http://localhost:3000/api/items>

Get By Id

<http://localhost:3000/api/items/66d1909d3c4552f6dab6bee3>

POST – Create New

<http://localhost:3000/api/items>

Body – JSON

{

"name":"Item 3",

"price":"789.22"

}

PUT – Updating the record

<http://localhost:3000/api/items/66d19363c3fe987c079eaec1>

Body – JSON

{

"name":"Modified Item 3",

"price":"789.33"

}

DELETE – Deleting the record

<http://localhost:3000/api/items/66d19363c3fe987c079eaec1>

Assignment – Modularize RESTful Service implementation.

If possible, Create Client app for the same – Javascript, HTML5, CSS3 app / React app / Angular app.

**HAPI**

hapi.js is a rich framework for building applications and services in Node.js. It was originally developed by Walmart Labs to handle their mobile traffic during black Friday, and it has since grown into a powerful and flexible tool for creating scalable and maintainable web applications.

Key Features / Benefits of hapi

* Configuration over Code
* Integrated with Input Validation – using Joi – powerful schema-based validation library.
* Powerful Plugin System
* Advanced Routing – complex validation, authentication, response handling in declarative way.
* Security Features – Validations, CSRF, XSS out of the box taken care.
* Error Handling – robust error handling mechanism.
* Extensive Ecosystem – mature ecosystem of plugins, extensions those integrate well with the framework. Plugins can help with tasks like authentication, logging, caching etc.

When to choose hapi over express

* Large, Complex project – maintainability, consistency, modularity is critical.
* Security-First Application – Security is a top concern.
* Teams and Enterprise – standardization and clear patterns are important.
* API-First Development

Choice between hapi and express ultimately depends on the specific needs of your project and your preference for flexibility versus configuration.

**Microservices**

Microservices are a software architecture where a large application is broken down into smaller, independent services that communicate with each other using well-defined APIs.

These services are developed and deployed independently, allowing for greater flexibility, scalability, and maintainability.

**Key Characteristics of Microservices**

* Decentralized Governance – Each microservice has its own team responsible for development, deployment, and maintenance.
* Bounded Contexts – focuses on specific business domain or functionality.
* Independent Deployments.
* Technology Agnostic
* API-First approach

**Challenges of Microservices**

* Complexity
* Overhead – communication between services
* Testing
* Deployment

**Netflix platform is built on microservices, Amazon, Spotify**

**Communication between Microservices**

1. **RESTful APIs**
   1. HTTP-based – Use HTTP methods (GET, POST, PUT, DELETE) to represent actions.
   2. JSON or XML – data is typically formatted in Json or xml.
2. **Message Queues**
   1. Asynchronous Communication – Messages are sent to a queue and consumed by interested services.
   2. Decoupling – Services don’t need to be directly connected, improving scalability and resilience.
   3. E.g. RabbitMQ, Apache Kafka, Amazon SQS
3. **Event-Driven Architecture**
   1. Events – occurrences that trigger actions in other services.
   2. Event Bus – A centralized mechanism for publishing and subscribing to events.
   3. E.g. Apache Kafka, Amazon Kinesis
4. **RPC (Remote Procedure Calls)**
   1. Synchronous Communication – A service calls a procedure on another service.
   2. Protocol Buffers – Often used for efficient serialization and deserialization of data.
   3. E.g. gRPC, Thrift
5. **Shared Database**

**Choosing the right approach - depending on factors such as**

* Data consistency requirements
* Performance needs
* Scalability requirements
* Complexity

E-commerce application

Microservices for

product catalog,

order processing,

inventory management

* **Product catalog** and **Order processing** might use **REST APIs** to communicate, as they need to exchange product information and order details.
* **Order processing** and **Inventory Management** could use a **message queue** to asynchronously update inventory levels when an order is placed.

**API Gateway** – in microservices architecture acts as a central point of entry for clients to access multiple microservices. It provides a unified interface, simplifying the process of interacting with a complex system of interconnected services.

**Key Functions of an API Gateway**

* Simplified Client Interface – single entry point for clients.
* Routing – directs incoming requests to the appropriate microservice based on the request’s path, query parameters, and headers.
* Aggregation – Combines responses from multiple microservices into a single response for the client.
* Security – Enforces security measures like authentication, authorization and rate limiting.
* Caching – Caches frequently accessed data to improve the performance.
* Protocol Translation – converts different communication protocols (e.g., REST to gRPC)
* Load Balancing – Distributes traffic across multiple instances of a microservice to ensure even load distribution.