# **Class 0 – NodeJS Basics**

Run app.js file:

Node app.js

File System – fs

Node.js uses ES5 by default

http: hypertext transfer protocol

fs.readdir(directorypath, (error, file) {

if(error) {

console.log(error);

} else {

console.log(file);

}

} // this method reads all the files on this directory

fs.readFile(./abc.txt, utf8, (error, fileread) {

if(error) {

console.log(error);

} else {

console.log(fileread);

}

} // this method reads the given file and returns everything which lies on it

fs.writeFile('./abc.txt', 'Some new text from server.js...', err => {

if(err) {

console.log(err);

}

}) // this method changes text in the given file (it overrides the text)

fs.appendFile('./abc.txt', ' \nSome new text from server.js...', (err) => {

if(err) {

console.log(err);

}

}); // this method appends text in the given file (it doesn’t override the text)

// HTTP Requests

let server = http.createServer((req, res) => { // req is the client side request & res is the response to send

res.write("Server is listening...");

res.end();

});

server.listen(5000); // To listen the server at Port: 5000

# **Class 1 – ExpressJS**

**NodeJS** is a JavaScript runtime environment.

**ExpressJS** is a backEnd web application framework for building RESTful APIs with Node.js

**Framework:**

A framework has a lot of built-in tools which makes development easy but customization is hard.

Create Package.json (for node.js app):

npm init / npm init -y

Install Express:

npm i express / npm install express

Node.JS app uses ES5 (aka. commonJS) instead of ES6 (aka. Module JS) which is not directly supported in Node.js applications. We have to add Module JS to our app by adding this in our package.json

{

"type": "module"

}

**APIs:**

APPLICATION PROGRAMMING INTERFACE - API

Api is an interface used to communicate Client-Side to the server through programming.

How does API work?

API works on http protocol

It takes a request from client and sends it to the server then

it takes the response of the server and sends it to the client

Http: HYPERTEXT TRANSFER PROTOCOL

Https: HYPERTEXT TRANSFER PROTOCOL SECURE (SSL Certified)

These API's are called Requests:

app.get('/user', () => {

});

A request can be of any of the following methods:

1. Get (body cannot be sent through get method on the browser)

2. Post

3. Put (used to update multiple data/items)

4. Patch (used to update one specific data)

4. Delete etc

http://localhost:5000/user is a URL

http://localhost:5000 is the URL and

/user is endPoint here

**Postman:**

Postman is a testing tool. With this tool you can build, test your APIs.

Nodemon Installation in the System:

npm i -g nodemon

Now go to package.json and edit Script object:

"scripts": {

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

"start": "node app.js",

"dev": "nodemon app.js"

},

Now you can use these conmmands:

npm start

npm run dev

You can use the following command if you are using node version 18 or later

npm start --watch

**Note:** Every time you want to send data from Client-Side use post method.

You can never send body(data) using get method from the browser because get method is only used for getting data.

Urdu: Get se kabhi bhi browser per body nahi send hoti q k get ki request se sirf data get hota hai

Go to Body > raw > JSON

A research says that In API request/response JSON format is used 96% worldwide

Body is not directly accessible through request.body. To access the body content we have to use this body-parser (i.e. express.json()) inside app.use Middleware:

app.use(express.json());

**RESTful API's:**

Main Concept of rest Api is that it returns JSON Format

An API created by following all Rules /Standards is called Rest API.

Some of the rules are as follows:

For example we want to do a CRUD operation:

User - Create

User - get

User - Update

User - Delete

Rule No 1: Use the Same method as the purpose of using the API

Rule No 2: Same End Point for one purpose

Rule No 3: To Differentiate End Points use identifier e.g. /api

**Middleware:**

The term Middleware is used to refer to pre-built software components that can be added to the framework's request/response processing pipeline, to handle tasks such as database access.

# **Class 2 – Database Connection**

**Database:**

A database is an organized collection of data stored and accessed electronically. A database can be of two types:

1. **SQL (Structured Query Language):**
   1. entity-based database (table format with rows and columns)
   2. Schema is must
2. **NoSQL:**
   1. BSON format (alike JSON format)
   2. Is schema less
   3. Can be relational or non-relational

**Relational database:**

Uses a key/id to identify the user.

Or

A relational database is a type of database that stores and provides access to data points that are related to one another.

**MongoDB:**

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses BSON (JSON-like) documents with optional schemas.

MongoDB altas is a cloud database.

Now create a project and then create a Cluster.

1. For Security purpose we have to create a database user using username and password so that the user can access database using the same credentials.
2. Any IP Address is acceptable if you give this:

IP Address: 0.0.0.0/0

Finish and Close > Go to database

1. Click on Connect > Connect your application > copy and paste the database connection string in your code then modify your connection string (DBURI) by replacing **test** with your project name and **<password>** with your password.

**'mongodb+ske://name:<password>@cluster0.abcdefg.mongodb.net/projectname'**

Node.js doesn’t connect directly with MongoDB we have to use a Library called mongoose.

**Mongoose:**

Mongoose is a JavaScript object-oriented programming library that creates a connection between MongoDB and the Node.js JavaScript runtime environment. It is used:

* To build connection between node.js app and MongoDb.
* To run Database Queries.

To Build Connection between node.js app and mongoDB use:

mongoose.connect(DBURI)

**Database Query:**

A database query is either an action query or a select query. A select query is one that retrieves data from a database. An action query asks for additional operations on data, such as insertion, updating, deleting or other forms of data manipulation.

**Migration:**

The process of moving data from one database to another is called migration.

After migration you just have to change the connection string.

**Schema:**

Schema/Model is defined to restrict the database to save only the given fields for Security Purpose (i.e. to protect the server from any unwanted script/string by avoiding any extra field). While creating models keep the collection name singular (i.e. user) because MongoDB adds an s/es by default (i.e. users).

**Database Rules:**

* Store data in snake\_case.

**Parameters/ Query Parameters:**

If we want to send any data from front-end to the server using get method we use Params/Query Params.

**Parameters/Params example:**

API Request: app.get(“/api/user/:userId”)

Front-End URL: https://myapp.com/api/user/:userId

console.log(request.params)

**Query Params example:**

API Request: app.get(“/api/user”)

Front-End URL: https://myapp.com/api/user?id = userId

console.log(request.query)

**Mongoose Queries:**

**Create User:**

**userModel.create({objToSend})**

* Takes an object as an argument
* Sends the object in database.

**Get User:**

**.find({})**

* Takes an object as an argument
* Finds & returns the whole collection
* Returns an array of objects

**.find({first\_name: "John"})**

* Finds and returns all the matching results
* Returns an object

**.findOne({\_id: mongoose.Types.ObjectId(userId)})**

* Id must be converted into mongodb id in the Argument.
* Takes an object as an argument.
* Finds & returns only the first matching one.
* Returns an object.

**.findById(userId)**

* Takes id (in string format) as an argument.
* Returns an object.

**Update User:**

**.findByIdAndUpdate( userId, objToSend, {new: true} )**

* Takes three arguments
  1. Id (in string)
  2. Object to update
  3. {new: true} is used to get updated data in response.

**Delete User:**

**.remove({})** // For deleting all users

* Takes an object as an argument
* Empty object will remove the whole collection

**.findByIdAndUpdate(userId)**

* Takes Id (in string)