Back-End Topics:

- REST API
- MongoDB Connection
- MongoDB CRUD App
- Login SignUp complete flow API
- JWT - JSON Web Token
- bcrypt password hash
- Middleware
- Schema
- JSON Schema Validation
- Query Params
- url-parameter
- Connect server with React App
- Server Deployment

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):**
   a. entity-based database (table format with rows and columns)
   b. Schema is must

2. **NoSQL:**
   a. BSON format (alike JSON format)
   b. Is schema less
   c. 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

3. 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:

.deleteMany({}) // For deleting all users

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

.findByIdAndDelete(userId)

- Takes Id (in string)

Class 3 – Authentication

MongoDb Compass Installation:

To make the development process easy we will use MongoDb Compass (the desktop application) instead of MongoDb web console

Always use snake_case for DB properties.

To perform asynchronous tasks in JavaScript we can use:

1. new Promise
2. Async Await
3. Callback function
4. FUNCTIONS

Authentication API's:

SignUp API:

1. Method Post
2. Create a Schema
3. Add Validation (for required fields)
4. Check Unique Email/Username by using findOne() method.
5. Hash password
6. Create User using userModel.create() query

There are two Validation Standard:

1. Client side validation - Not Secure because client has the access to the Front-end.
2. Server side validation - Secure because nobody has the access to it except the developer/owner.

   If key and value have the same name in an object then only key can be added by skipping the value.

Password Hashing:

Hashing is a form of encryption. Hashing turns your password (or any other piece of data) into a short string of letters and/or numbers using an encryption algorithm.

Install the bcryptjs npm package

```
$ npm i bcryptjs
```

Use:

bcrypt.hash(string, salt)
Don't use bcrypt.hashSync() as it is blocking
bcrypt.compare(password, User's hashedPassword)

string = password to be hashed
salt = No. of rounds

Login API:

1. Method Post (because we cannot send data with a get() request.
2. Add Validation
3. Find email using findOne()
4. Compare password

   Never throw proper errors in Authentication. Simply send "Credential Error".

JSON Web token (JWT):

The JWT is often used to secure RESTful APIs because it can be used to authenticate a client who wants to access the APIs.
This token is generated at the time of login and then sent to Front-End.

Last Class – CRUD Operations

1. Create a Front-End folder and create a react app.
2. Create a Back-End folder.

Back-End Folder:

1. npm init - creates a package.json file
2. Create aap.js
3. Go to package.json and make some changes.
   - main: app.js
   - start: node app.js
   - dev: nodemon app.js
4. npm i express
5. npm i mongoose
6. npm i dotenv
7. In app.js require("dotenv").config()
8. Create .env file
   Note: env file is created for security purpose and all the sensitive data e.g. db strings/keys/port etc. is stored in this file. Never push this file on GitHub.
9. npm i cors - to allow cross origin
10. Connect database using mongoose
11. Create Schema - You can add:
    - Timestamp in Schema, and then sort it when needed.
    - Default value using default: "any value"
    - Validation can also be added with schema but error handling is a bit complex here as the errors are not readable.
12. Add Validation

Create To-do:

1. Method Post
2. Create To-do using **TodoModel.create()**

Read To-do:

1. Method Get
2. Create To-do using **TodoModel.find({})**

Update To-do:

1. Method Put
2. Add Validation
3. Create an object for updated to-do value and existing id.
4. Update To-do using **TodoModel.findByIdAndUpdate({id, obj})**

Delete To-do:

1. Method Delete
2. Just send the to-do's I'd in Params don't have to send the whole body
3. Delete To-do using **TodoModel.findByIdAndDelete({}))**
4. Delete All using **TodoModel.deleteMany({})**

MVC Architecture (Model View Controller):

Model-View-Controller is a software architectural pattern commonly used for developing user interfaces that divide the related logic into three interconnected elements.

Model:	Data related Logics
View:	UI i.e. React App
Controller:	Business logic/Server

Router:

Create a routes folder and then an index.js file
const router = express.Router()
Now use router.get/post/put/delete/…

Route:

URL Endpoint is called route and the callback function is called controller.

Conditional Routing:

app.use("/api", router)
Only routes having API will proceed

Controller:

Create a controller folder and then a todo.js file
Inside todo.js create an object and store all the functions/business logic.

Server Deployment:

cyclic.sh is a platform for deploying full stack apps.

.gitignore:

Create a .gitignore file
node_modules
.env

Link your GitHub repo, set environment variables and deploy.

READme.md:

Create a READme.md file and add your live server URL.
server url : my-server-url.cyclic.app

Front-End Folder:

1. Create a UI for To-do

It is Axios default behavior that response.data is simply the response that was provided by the server.

In axios if the status code is not handled by the server axios will set the status code 200 by default and .catch() is triggered when the server returns any status other than 200. So even if the required field is empty .then() will be triggered.

So for empty fields we have to add status code 400 (i.e. Bad Request) from Back-End or handle the Error from Front-End.

Status Codes:

HTTP response status codes indicate whether a specific HTTP request has been successfully completed.
Status Codes defines the behavior of API.
Responses are grouped in five classes:

- Informational responses (100 – 199)
- Successful responses (200 – 299)
- Redirection messages (300 – 399)
- Client error responses (400 – 499)
- Server error responses (500 – 599)

- Mostly Used Codes: 200, 400, 401, 404, 500

To update the UI in react so that it feels real-time, we add a refresh state in the dependency array. Then using it as a flag in the create, update, delete functions.

Now handle the validation Error in Front-End.

Create To-do:

axios.post(URL, obj)

- Validation Error handling:
- Inside .then()   if(response.data.status) {setRefresh(!refresh)}
  else{alert(response.data.message)}

Read To-do/s:

axios.get(URL)

- Set to-dos in an empty array state using spread operator. example: setTodos([…response.data.data])

Arrays are non-primitive/reference data types i.e. it firstly creates a reference so to update the array so that it feels in real-time we use spread operator.

Update To-do

axios.put(URL, obj)

- Just send the edited value and id in the object
- Add flag to refresh the get API in useEffect.

Delete To-do:

axios.delete(`${BASE_URL}/todo/${id}`)

- Add flag to refresh the get API in useEffect.

Delete All To-do:

axios.delete(`${BASE_URL}/alltodos`)

- Add flag to refresh the get API in useEffect.

config.js:

Create a config.js in src and save:

```
const BASE_URL = http://localhost:8000/api/
// const BASE_URL = LIVE URL

export { BASE_URL }
```

After Deployment add your Live URL in config.js and uncomment it.

Network tab:

Check your API calling in the Network tab.

Back-End Installations:

```
npm init
npm i express
npm i mongoose
npm i dotenv
npm i cors
npm i bcryptjs
npm i jsonwebtoken
```