What is Back-end Development ?

- Back-end development means working on server- side software , which focuses on everything you can’t see on a website.

- Focusing on databases , back-end logic , APIs (Application Programming interface) and Servers.

- Programming Languages: JavaScript (Node JS) , Python (Django/Flask) , Java (Spring Boot) , PHP (Laravel) etc

Client

Database

Request

Response

Server

VIRTUAL CODE

ULTIMATE BACKEND COURSE

Level-1

Introduction to Node JS

Level-1

Introduction to Node JS

- Node JS is a JavaScript runtime Environment that allows developers to run JavaScript code outside of a web Browser.

- Node JS runs on the V8 JavaScript Engine .

- Node JS is Used to create web Server etc.

Level-1

Introduction to Node JS

VIRTUAL CODE

- Set-UP Node JS
- Install Node JS
- Install npm
- Create folder
- run npm init

VIRTUAL CODE

ULTIMATE BACKEND COURSE

Level-2

Our First Server in Node

Level-2

Our First Server in Node

- A Server is a computer or system that provides services , resources or data to other computers, called Clients , over a network.

- Example-

- When you open a website, your browser (the Client) sends a request to a server , and the server sends back the website data so you can see it.

Level-2

Our First Server in Node

VIRTUAL CODE

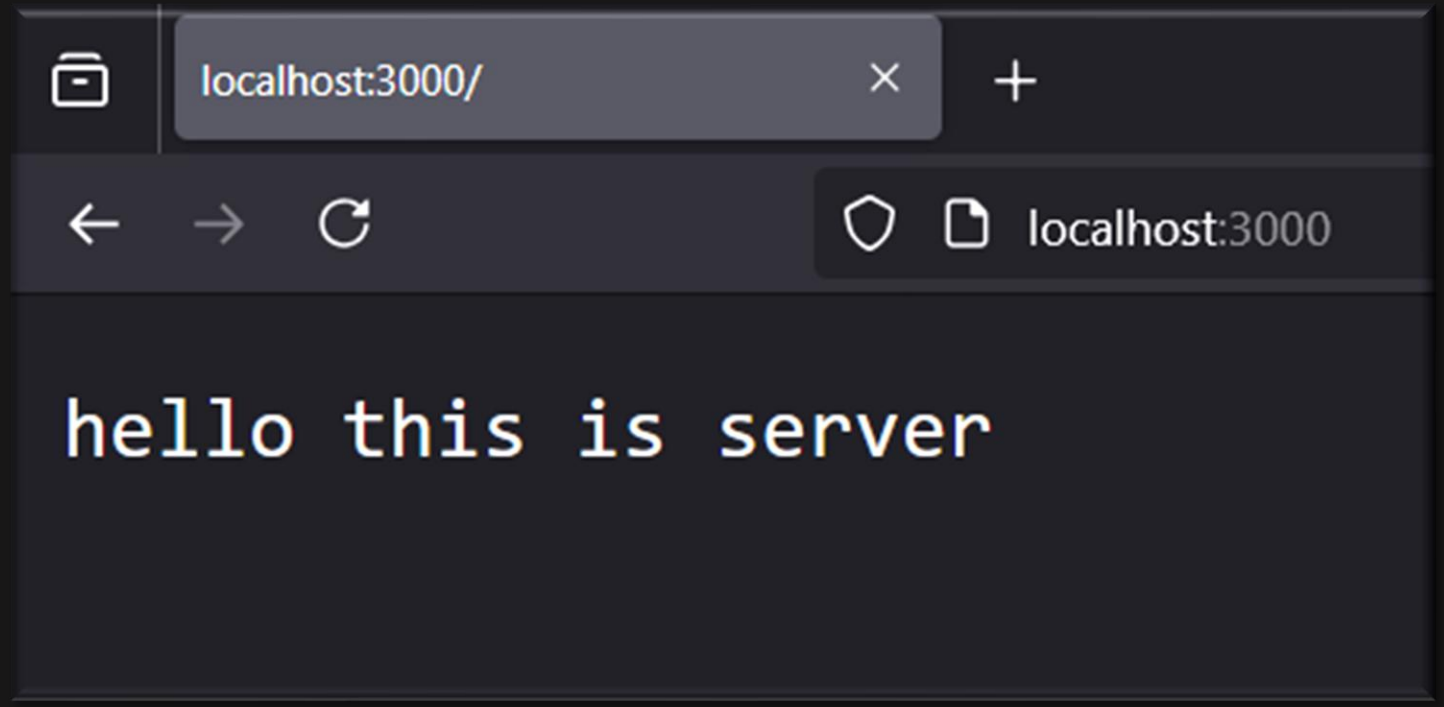
JS index.js > ...

```
1   import http from "http" // Hyper Text Transfer Protocol
2
3   const Port=3000;
4
5   const server=http.createServer((req,res)=>{
6       res.end("hello this is server")
7   })
8
9   server.listen(Port)
10
```

Level-2

Our First Server in Node

VIRTUAL CODE



Level-2

Our First Server in Node

Routing In Node JS -

- Routing is the process of defining how an application responds to different client requests based on the URL (or Route)

- Express JS simplifies route Creation

Level-2

Our First Server in Node

VIRTUAL CODE

JS index.js > ...

```
1   import http from "http" // Hyper Text Transfer Protocol
2
3   const Port=5000;
4
5   const server=http.createServer((req,res)=>{
6       if(req.url=="/"){
7           res.end("Welcome to Home Page")
8       }
9       else if(req.url=="/about"){
10          res.end("Welcome to About Page")
11      }
12      else if(req.url=="/contact"){
13          res.end("Welcome to Contact Page")
14      }
15      else{
16          res.end("404 Not Found")
17      }
18
19  })
20
21  server.listen(Port)
22
```

VIRTUAL CODE

ULTIMATE BACKEND COURSE

Level-3 Introduction to Express JS

Level-3 Introduction to Express JS

ex

- Express JS is the most Popular Framework of Node JS

- Instead of writing Everything manually with the node.JS http module , Express JS gives you shortcuts and a cleaner way to organize your code.

- Install Express JS package by Running this command-

```
npm install express
```

Level-3

Introduction to Express JS

Creating Server using Express JS-

backend > JS index.js > ...

```
1  import express from "express"
2  let app=express()
3
4  app.get("/",(req,res)=>{
5      res.send(" hello, I'm Home Route.")
6  })
7
8  app.listen(4000,()=>{
9      console.log("server is Created !");
10 })
11
```

Importing express package

HTTP Method

Level-3 Introduction to Express JS ex

HTTP methods-

- HTTP methods are used to handle various types of requests made to a server.

- The most commonly used HTTP methods include-

1. GET Method - Used to retrieve data from the server
2. POST Method - Used to send data to the server(create new resource)
3. PUT Method - Used to update an existing resource.
4. PATCH Method - Used to partially update a resource.
5. DELETE Method - Used to delete a resource

Level-3

Introduction to Express JS

ex

req.params -

```
// test URL :  http://localhost:4000/user/27
```

- req.params is an object that stores route Parameters in express JS.
- It is used to capture dynamic values from the URL.

backend > JS index.js > ...

```
1  import express from "express"
2  let app=express()
3
4  app.get("/user/:id",(req,res)=>{
5      const id= req.params.id
6      res.send(`fetching details of user with ID: ${id}`)
7  })
8
9  app.listen(4000,()=>{
10     console.log("server is Created !");
11 })
12
```

Level-3

Introduction to Express JS

ex

req.query -

- req.query is an object that stores query parameters from the URL.

- Query Parameter are sent as key-value pairs in the URL after the ? Symbol and are typically used for filtering , searching etc.

Level-3

Introduction to Express JS

ex

req.query -

```
// test URL :  http://localhost:4000/user?name=ayush&age=21
```

```
1  import express from "express"
2  let app=express()
3
4  app.get("/user",(req,res)=>{
5      const {name,age}= req.query
6      res.send(`fetching user with name: ${name} and age:$
       {age}`)
7  })
8
9  app.listen(4000,()=>{
10     console.log("server is Created !");
11 })
```

VIRTUAL CODE

ULTIMATE BACKEND COURSE

Level-4

Connect Backend with Frontend

Level-4

Connect Backend with Frontend

RESTful API (Representational State Transfer) –

- A RESTful API is a way for applications to communicate with each other over the internet using standard HTTP requests like GET, POST , PUT and DELETE.

- Client (ex. A mobile app or Website) sends a request.

- Server receives the request and processes it.

- Server sends back a response (usually in JSON format)

Level-4

Connect Backend with Frontend

CORS (Cross Origin Resource Sharing) in express JS –

- CORS is a security feature in web browsers that prevents requests from different origins unless explicitly allowed by the server.

- Same-origin Policy restricts requests from different origins (protocol, domain, or port)

- CORS allows servers to specify who can access their resources.

```
npm install cors
```

VIRTUAL CODE

ULTIMATE BACKEND COURSE

Level-5

Middlewares, Status Code and HTTP Headers

Level-5

Middleware, Status Code and HTTP Headers

Middlewares–

- Middleware runs before the route handler.
- Middleware must call next() to continue to the next function .
- If Middleware does not call next() , the request will hang.
- There are some built-in , custom , thirdparty middleware etc.

```
app.use(cors())
```

→ Third Party Middleware

```
app.use(express.json())
```

→ Built-in Middleware

```
app.use((req,res,next)=>{
  console.log("custom middleware");
  next()
})
```

→ custom Middleware

Level-5

Middleware, Status Code and HTTP Headers

Status Code –

1. Informational responses (`100` – `199`)
2. Successful responses (`200` – `299`)
3. Redirection messages (`300` – `399`)
4. Client error responses (`400` – `499`)
5. Server error responses (`500` – `599`)

Level-5

Middleware, Status Code and HTTP Headers

HTTP Headers–

- HTTP Headers are key-value Pairs used in HTTP requests and responses to pass additional information between the client and the server.

- They help in defining metadata , specifying content type , setting authentication tokens etc.

- Types of HTTP Headers :

1. Request Headers : Sent by the client to the server
   (User-Agent)
2. Response Headers : Sent by the server to the client
   (Content-Type)

Level-5

Middleware, Status Code and HTTP Headers

HTTP Headers–

- Get request Headers:

1. req.get - for getting Specific Headers.
2. req.headers - for getting all headers.

- Set Response Headers: use res.set( ) or res.header( )

- Remove Headers : use res.removeHeader(headerName)

VIRTUAL CODE

ULTIMATE BACKEND COURSE

Level-6

Introduction to MongoDB

Level-6

Introduction to MongoDB

What is Database ? –

A database is a collection of data that allows storing ,managing and retrieving information efficiently.

- Common used databases-

1. SQL database – MySQL, PostgreSQL (stores data in tables)
2. NoSQL database – MongoDB (Document Based)

Level-6

Introduction to MongoDB

MongoDB–

- MongoDB is a NoSQL database that stores data in flexible , JSON – like format instead of tables.

ID	Name	Age
1	Ayush	21
2	Dev	22

SQL

NoSQL

```
[
    {
        "ID":1,
        "Name":"Ayush",
        "Age":21
    },
    {
        "ID":2,
        "Name":"Dev",
        "Age":22
    }
]
```

Level-6

Introduction to MongoDB

MongoDB–

Collection

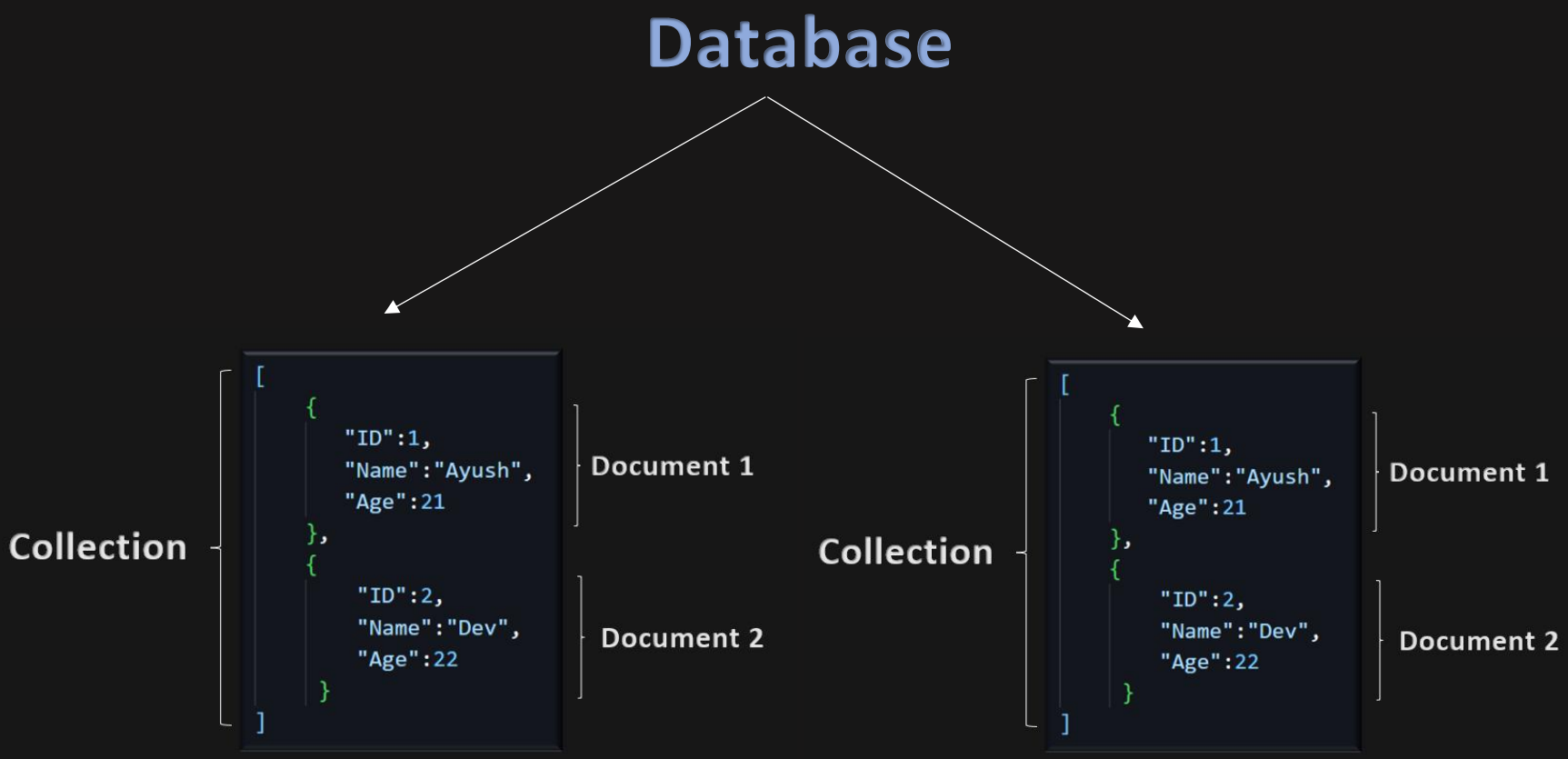
```
[
    {
        "ID":1,
        "Name":"Ayush",
        "Age":21
    },
    {
        "ID":2,
        "Name":"Dev",
        "Age":22
    }
]
```

Document 1

Document 2

Level-6

Introduction to MongoDB



Level-6

Introduction to MongoDB

Mongoose –

- Mongoose is an ODM (Object Data Modeling) library for MongoDB and Node JS.

- It helps developers interact with MongoDB using an easy an Structured approach by defining schemas and models.

- Install Mongoose using -

```
npm install mongoose
```

VIRTUAL CODE

Level-6

Introduction to MongoDB

Set-UP MongoDB

↓

Install mongoose

↓

Connect DB

↓

Design Schema

↓

Create model based on Schema for performing CRUD Operations

Level-6

Introduction to MongoDB

Schema –

- A Schema in Mongoose defines the structure of documents within a MongoDB collection .

- It specifies the fields , their types, validation rules , and default values.

Model –

- A model is wrapper for schema and provides an interface to interact with MongoDB Collection.

```
export let User = mongoose.model("User", userschema)
```

Level-6

Introduction to MongoDB

Schema –

```
import mongoose from "mongoose";

const userschema =new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true,
        unique: true
    },
    password: {
        type: String,
        required: true,
    }

}, {
    timeStamps: true
})
```

Level-6

Introduction to MongoDB

Operation	syntax
Create (Insert one)	User.create({ })
Create (Insert many)	User.insertMany([{ } , { }])
Read (Find All)	User.find( )
Read (Find One)	User.findOne({ })
Read (Find by ID)	User.findById(“id”)
Update (one document)	User.updateOne({ } , { })
Update (Find and Update)	User.findOneAndUpdate({ },{ },{ })
Delete (One Document)	User.deleteOne({ })
Delete (Many Documents)	User.deleteMany({age : {$lt : 18} })

Level-6

Introduction to MongoDB

1. Query Operators

Used in `find()`, `findOne()`, and `aggregate()` to filter documents.

Comparison Operators

Operator	Description	Example
`$eq`	Matches values equal to a specified value	`{ age: { $eq: 25 } }`
`$ne`	Matches values **not** equal to a specified value	`{ status: { $ne: "active" } }`
`$gt`	Matches values **greater than** a specified value	`{ price: { $gt: 100 } }`
`$gte`	Matches values **greater than or equal to** a value	`{ age: { $gte: 18 } }`
`$lt`	Matches values **less than** a specified value	`{ rating: { $lt: 4.5 } }`
`$lte`	Matches values **less than or equal to** a value	`{ age: { $lte: 30 } }`
`$in`	Matches values present in an array	`{ status: { $in: ["active", "pending"] } }`
`$nin`	Matches values **not** in an array	`{ category: { $nin: ["electronics", "furniture"]`

Level-6

Introduction to MongoDB

Logical Operators

Operator	Description	Example
`$and`	Matches documents that satisfy **all** conditions	`{ $and: [{ age: { $gte: 18 } }, { age: { $lte: 30 } }] }`
`$or`	Matches documents that satisfy **at least one** condition	`{ $or: [{ status: "active" }, { status: "pending" }] }`
`$nor`	Matches documents that **do not** satisfy any conditions	`{ $nor: [{ status: "active" }, { status: "pending" }] }`
`$not`	Inverts a query expression	

Evaluation Operators

Operator	Description	Example
`$regex`	Matches a string using regex	`{ name: { $regex: /^J/ } }`
`$expr`	Allows use of aggregation expressions in queries	`{ $expr: { $gt: ["$price", "$discounted } }`
`$mod`	Matches numbers divisible by a value	`{ age: { $mod: [2, 0] } }`

VIRTUAL CODE

ULTIMATE BACKEND COURSE

Level-7

Authentication and Image Upload

Level-7

Authentication & Authorization

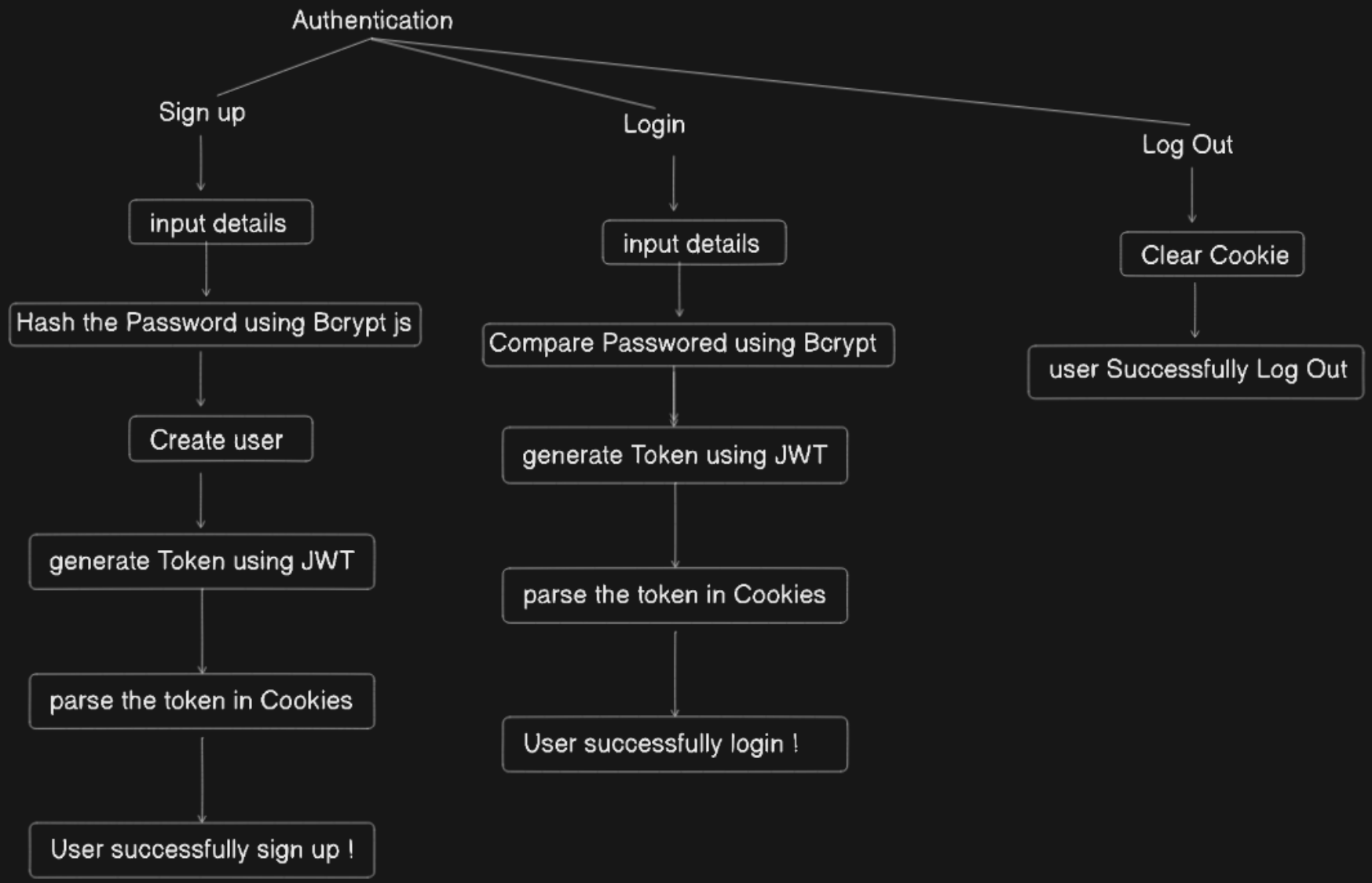
Authentication –

- Verifies who a user is. (Login & SignUp using email/password)

Authorization –

- Determines what a user can access .

Level-7 Authentication & Authorization



Level-7 Authentication & image Upload

Steps for Uploading an Image Using Multer and Cloudinary

Step	Description	Code/Command
1	Install required packages	`npm install multer cloudinary dotenv`
2	Configure Cloudinary account	Sign up at Cloudinary and get API credentials

3	Set up environment variables	Create a `.env` file and add:
		`CLOUDINARY_CLOUD_NAME=your_cloud_name`
		`CLOUDINARY_API_KEY=your_api_key`
		`CLOUDINARY_API_SECRET=your_api_secret`

4	Configure Cloudinary in Node.js	Use Cloudinary SDK for uploading images
5	Set up Multer for handling file uploads	Use `multer` to process file data
6	Create an Express route for file upload	Define API endpoint and upload image to Cloudinary
7	Test image upload API	Use Postman or frontend form to upload images
8	Store the image URL in MongoDB (optional)	Save `result.secure_url` in the database for future access

This method first stores the image locally using `multer`, then uploads it to Cloudinary, and finally deletes the local file to save space. 🚀 Let me know if you need any modifications!