Experiment-1

Student Name: Ajitesh **UID:** 22BCS12705

Branch: BE-CSE **Section/Group:NTPP_IOT-603**

Semester:6th **Date of Performance:8/11/2025**

Subject Name: AP LAB-II **Subject Code:** 22CSP-351

1. Aim:

Full Stack Development (MERN).

The primary aim of this experiment is to provide students or developers with an understanding of full-stack development involving MongoDB, Node.js, React, and Express.

- Problem 1.1.1: Give understanding of MongoDB, Nodejs, React, Express.
- Problem 1.1.2: Create a Frontend design of Login/Signup pages and create a backend of it.

Problem Breakdown.

- Problem 1.1.3: Test the Backend API Using Postman.

2. Introduction to the MERN Stack:

The MERN stack is a popular set of technologies for building full-stack web applications. It consists of:

- MongoDB: A NoSQL database that stores data in a flexible, JSON-like format, which is great for handling unstructured data.
- Express.js: A minimal and flexible Node.js web application framework that simplifies routing and handling HTTP requests.
- React: A JavaScript library for building user interfaces, particularly for single-page applications. It allows developers to create dynamic web applications with ease.
- Node.js: A JavaScript runtime environment that lets you run JavaScript code on the server-side, enabling full-stack JavaScript development.

The MERN stack is highly favored for its ease of use, scalability, and the fact that it allows developers to work in JavaScript across both the client and server sides.

3. Implementation/Code:

BACKEND

cd backend npm install const { MongoClient, ServerApiVersio n } = require('mongod b'); const uri = "mongodb+srv://rosh63441:<password>@rosh.yhbuk.mongodb.net/?retryWrites=true&w=majority&appName=Rosh";

```
// Create a MongoClient with a MongoClientOptions object to set the Stable API version const client = new MongoClient(uri, {
  serverApi: {  version: ServerApiVersion.v1,
    strict: true,
deprecationErrors: true,
  }
});

const connectDB = async () => {
try {
    // Connect the client to the server (optional starting in v4.7)  await client.connect();
    // Send a ping to confirm a successful connection  await client.db("admin").command({ ping: 1 });
    console.log("Pinged your deployment. You successfully connected to MongoDB!");
  } catch (err) {
console.error(err.message);
    process.exit(1);
  }
};

module.exports = { connectDB, client };

const bcrypt = require('bcryptjs'); const jwt = require('jsonwebtoken'); const {
client } = require('../config/db');
```

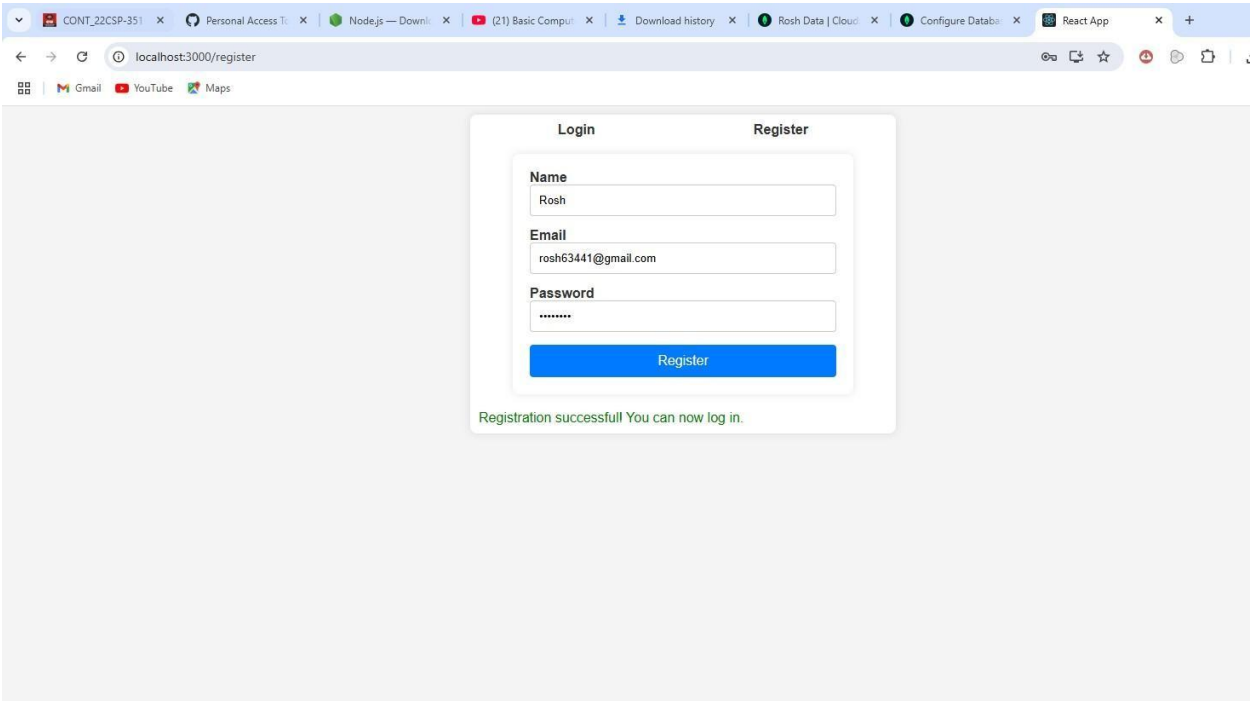
```
exports.register = async (req, res) => {
  const { name, email, password } = req.body;

  try {
    const db = client.db('auth');     const
users = db.collection('users');

    let user = await users.findOne({ email });
if (user) {        return res.status(400).json({ msg: 'User
already exists' });      }

    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);

    user = { name, email, password: hashedPassword };
await users.insertOne(user);

    const payload = { user: { id: user._id } };

    jwt.sign(payload, 'secret', { expiresIn: 360000 }, (err, token) => {
if (err) throw err;       res.json({ token });
    });
  }       catch    (err)     {     console.error(err.message);
res.status(500).send('Server error');
  }
};

exports.login = async (req, res) => {
  const { email, password } = req.body;

  try {     const db = client.db('auth');
    const users = db.collection('users');

    let user = await users.findOne({ email });      if (!user) {
return res.status(400).json({ msg: 'Invalid credentials' });
    }

    const isMatch = await bcrypt.compare(password, user.password);
if (!isMatch) {
```

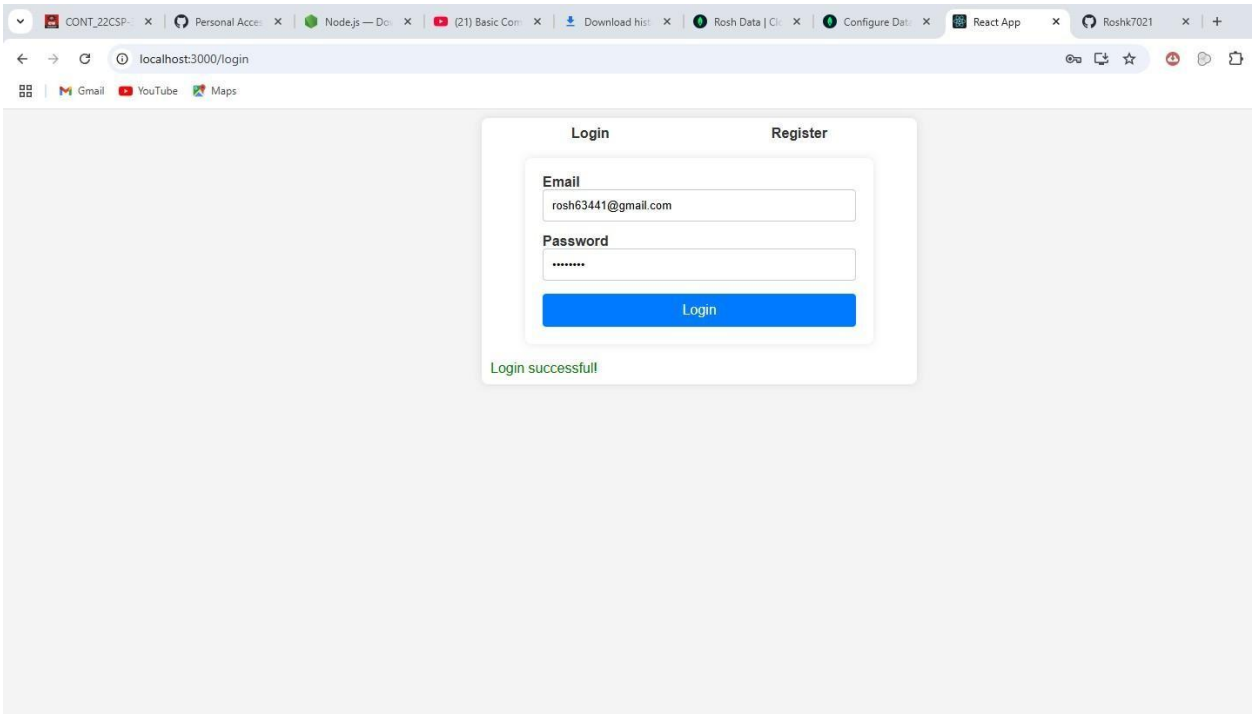
```
      return res.status(400).json({ msg: 'Invalid credentials' });
    }

    const payload = { user: { id: user._id } };

    jwt.sign(payload, 'secret', { expiresIn: 360000 }, (err, token) => {
if (err) throw err;       res.json({ token });
    });
  }       catch    (err)    {     console.error(err.message);
res.status(500).send('Server error');
  }
};

// filepath: /backend/models/User.js const mongoose
= require('mongoose'); const
UserSchema = new mongoose.Schema({   name:
{    type: String,    required: true,
  },  email: {    type:
String,    required:
true,
unique: true,
  },
  password: {    type:
String,
required: true,
  },
});

module.exports = mongoose.model('User', UserSchema);
```

- **FRONTEND**

cd fronted npm install
npm start

4. Output


CONT_22CSP-351
Personal Access T
Node.js — Downl
(21) Basic Comput
Download history
Rosh Data | Cloud
Configure Databa
React App
localhost:3000/register
Gmail
YouTube
Maps
Login
Register
Name
Rosh
Email
rosh63441@gmail.com
Password
Register
Registration successfull You can now log in.


CONT_22CSP-
Personal Acces
Node.js — Do
(21) Basic Com
Download hist
Rosh Data | Cl
Configure Dat
React App
Roshk7021
localhost:3000/login
Gmail
YouTube
Maps
Login
Register
Email
rosh63441@gmail.com
Password
Login
Login successful!

5. Learning Outcome

- Learn about MongoDB: Understand how to use MongoDB as a NoSQL database for storing and retrieving user data.

DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

- Learn about Node.js: Understand how to set up and use Node.js as a backend server and handle API requests.
- Learn about Express.js: Understand how to use Express.js to create routes and handle HTTP requests in the Node.js server.
- Learn about React: Learn how to create a simple frontend interface with React to handle user interactions (login/signup).
- Backend API Testing: Use tools like Postman to test backend APIs and ensure the server is responding correctly.
- Integration: Integrate the frontend (React) with the backend API to create a full-stack authentication system.