



**CHANDIGARH
UNIVERSITY**
Discover. Learn. Empower.

**NAAC
GRADE A+**
Accredited University

UNIVERSITY INSTITUTE OF ENGINEERING

Department of Computer Science & Engineering

(BE-CSE-6th Sem)



Subject Name: Advanced Programming Lab-2

Subject Code: 22CSP-351

Submitted to:

Vishal Kumar

Submitted by:

Name: JEET BHARTI

UID: 22BCS14804

Section: FL_IOT-602

Group: 'A'

INDEX

Name: JEET BHARTI

UID: 22BCS14804

Ex No	Name of Experiments	Date	Conduct (MM: 12)	Viva (MM: 10)	Worksheet (MM: 8)	Total (MM: 30)	Remarks	Signature
1	Full Stack Development (MERN): In this experiment, students will learn to develop a full-stack web application using the MERN Stack							
2	Arrays, Stacks, Queues: To implement the concept of Arrays, Queues and Stack.							
3	Linked Lists: This experiment combines data structures and operating system concepts, focusing on Linked Lists and OS-related problems.							
4	String Matching, Hashing, Heap: In this experiment, students will learn about String Matching, Hashing, and Heap data structures.							
5	Trees: To demonstrate the concept of Trees.							
6	Graph: This experiment focuses on Graph theory and its application in Computer Networks.							
7	Divide and conquer: To demonstrate the concept of Divide and Conquer.							
8	Greedy: To demonstrate the concept of Greedy approach.							
9	Backtracking and DP: To demonstrate the concept of Backtracking and Dynamic Programming							
10	Lab Based Mini Project							



EXPERIMENT – 1

Student Name: JEET BHARTI

Branch: BE-CSE

Semester: 6th

Subject Name: AP LAB-II

UID: 22BCS14804

Section: FL_IOT-602 'A'

Date of Performance: 16/01/25

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. Objective:

- Understand the fundamentals of MongoDB, Node.js, React, and Express
- Create a functional frontend for Login/Signup pages
- Develop a backend using Express and MongoDB
- Test the backend API using Postman

3. 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.

4. Implementation/Code:

➤ BACKEND

cd backend
npm install

(i) db.js

```
const { MongoClient, ServerApiVersion } = require('mongodb');
const uri =
"mongodb+srv://rosh63441:<password>@jeetbharti.yhbuk.mongodb.net/?retryWrites=true&w=m
ajority&appName=JeetBharti";
```

// 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 };
```

(ii) authController.js

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



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
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 } };
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
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');  
}  
};
```

(iii) User.js

```
// filepath: /backend/models/User.js  
const mongoose = require('mongoose');
```

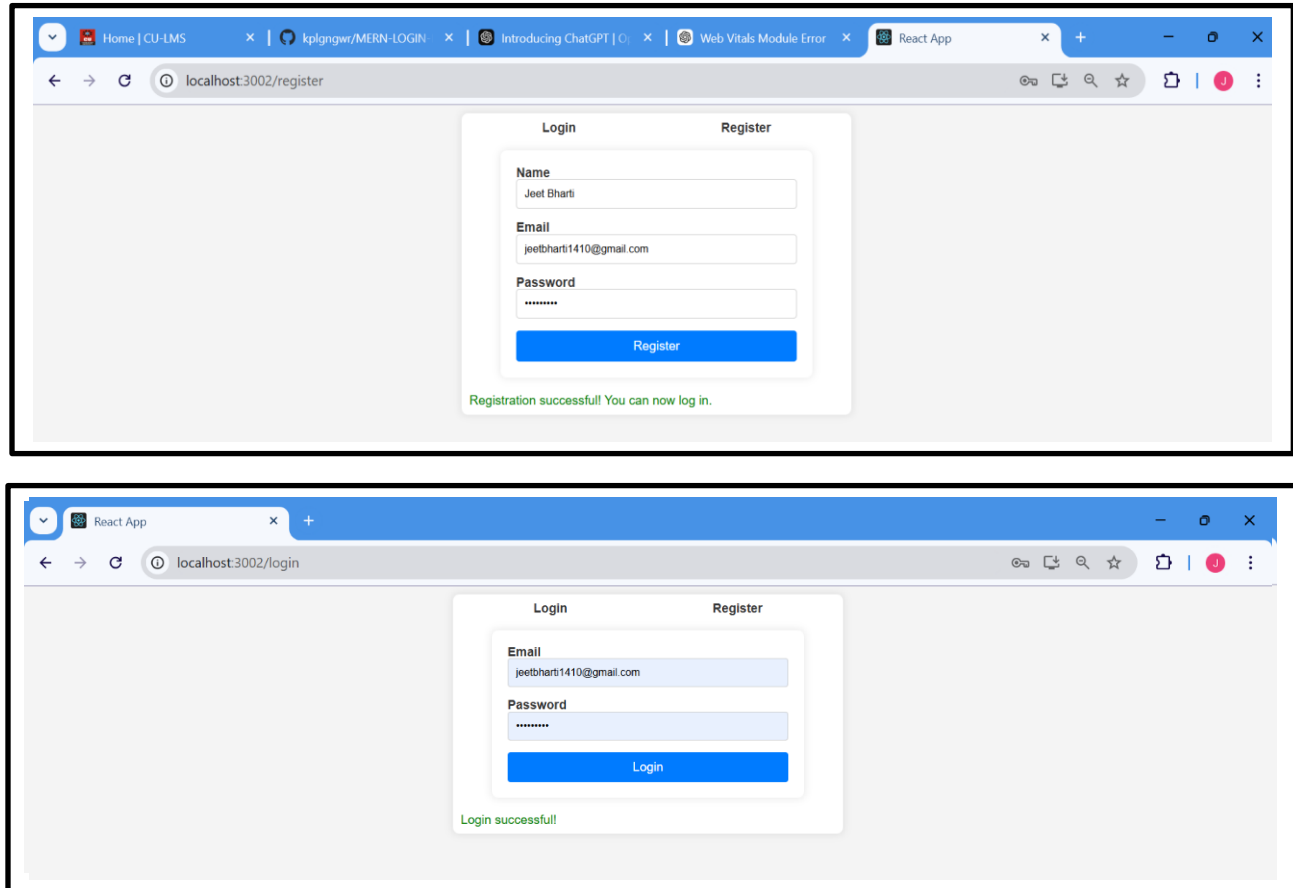
```
const UserSchema = new  
mongoose.Schema({ name: {  
  type: String,  
  required: true,  
},  
email: {  
  type: String,  
  required: true,  
  unique: true,  
},  
password: {  
  type: String,  
  required: true,  
},  
});
```

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

➤ FRONTEND

```
cd fronted  
npm install  
npm start
```

5. Output:



The first screenshot shows the registration page at localhost:3002/register. The form has fields for Name (Jeet Bharti), Email (jeetbharti1410@gmail.com), and Password (masked with asterisks). A blue 'Register' button is at the bottom. A green message at the bottom states: 'Registration successfull You can now log in.'

The second screenshot shows the login page at localhost:3002/login. The form has fields for Email (jeetbharti1410@gmail.com) and Password (masked with asterisks). A blue 'Login' button is at the bottom. A green message at the bottom states: 'Login successfull'.

6. Learning Outcomes:

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