



WORKSHEET 1

Student Name: Parth Arora

UID: 22BCS16661

Branch: BE-CSE

Section/Group: 22BCS_NTPP-602-A

Semester: 6th

Date of Performance: 15/01/2025

Subject Name: AP LAB - II

Subject Code: 22CSP-351

1. Aim: 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.

2. Objective:

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

The MERN stack is a widely used combination of technologies for building robust full-stack web applications. It comprises:

- **MongoDB:** A NoSQL database that stores data in a flexible, JSON-like format, ideal for managing unstructured or semi-structured data.
- **Express.js:** A lightweight and flexible web application framework for Node.js, simplifying routing and handling of HTTP requests.
- **React:** A JavaScript library focused on building user interfaces, especially for single-page applications, making it easier to create dynamic and responsive web applications.
- **Node.js:** A JavaScript runtime environment that allows developers to execute JavaScript code on the server side, enabling seamless full-stack development in a single language.

The MERN stack is widely preferred for its simplicity, scalability, and the ability to use JavaScript consistently across the front-end and back-end, streamlining the development process.

3. Source Code:

Login.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <div class="form-box">
      <h2>Login</h2>
      <form id="loginForm" action="#" method="post">
        <div class="input-group">
          <label for="loginEmail">Email:</label>
          <input type="email" id="loginEmail" name="email" required>
        </div>
        <div class="input-group">
          <label for="loginPassword">Password:</label>
          <input type="password" id="loginPassword" name="password" required>
        </div>
        <button type="submit">Login</button>
      </form>
      <button id="toggleSignup" onclick="toggleForms()">Switch to Signup</button>
    </div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

Setup.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Signup</title>
<link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <div class="form-box">
      <h2>Signup</h2>
      <form id="signupForm" action="#" method="post">
        <div class="input-group">
          <label for="firstName">First Name:</label>
          <input type="text" id="firstName" name="firstName" required>
        </div>
        <div class="input-group">
          <label for="lastName">Last Name:</label>
          <input type="text" id="lastName" name="lastName" required>
        </div>
        <div class="input-group">
          <label for="signupEmail">Email:</label>
          <input type="email" id="signupEmail" name="email" required>
        </div>
        <div class="input-group">
          <label for="signupPassword">Password:</label>
          <input type="password" id="signupPassword" name="password" required>
        </div>
        <div class="input-group">
          <label for="confirmPassword">Confirm Password:</label>
          <input type="password" id="confirmPassword" name="confirmPassword"
required>
        </div>
        <div class="input-group">
          <label for="phone">Phone:</label>
          <input type="tel" id="phone" name="phone" pattern="[0-9]{10}" required>
        </div>
        <button type="submit">Signup</button>
      </form>
      <button id="toggleLogin" onclick="toggleForms()">Switch to Login</button>
    </div>
  </div>
```

```
<script src="script.js"></script>
</body>
</html>
```

App.js:

```
const handleSignup = async (event) => {
  event.preventDefault();
  const firstName = document.getElementById('firstName').value;
  const lastName = document.getElementById('lastName').value;
  const email = document.getElementById('signupEmail').value;
  const password = document.getElementById('signupPassword').value;
  const confirmPassword = document.getElementById('confirmPassword').value;
  const phone = document.getElementById('phone').value;

  if (password !== confirmPassword) {
    alert('Passwords do not match. ');
    return;
  }

  if (! /^[0-9]{10}$/.test(phone)) {
    alert('Enter a valid 10-digit phone number. ');
    return;
  }

  try {
    const response = await fetch('http://localhost:3000/signup', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ firstName, lastName, email, password, phone }),
    });

    const result = await response.json();
    alert(result.message);
  } catch (error) {
    console.error('Signup error:', error);
  }
};

const handleLogin = async (event) => {
  event.preventDefault();
  const email = document.getElementById('loginEmail').value;
  const password = document.getElementById('loginPassword').value;
```

```
try {
  const response = await fetch('http://localhost:3000/login', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ email, password }),
  });

  const result = await response.json();
  alert(response.ok ? `Welcome, ${result.username}!` : result.message);
} catch (error) {
  console.error('Login error:', error);
}
};
document.getElementById('signupForm').addEventListener('submit', handleSignup);
document.getElementById('loginForm').addEventListener('submit', handleLogin);
```

Server.js:

```
const express = require('express');
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');
const cors = require('cors');

const app = express();
const PORT = 3000;

app.use(express.json());
app.use(cors());

mongoose.connect('mongodb://your-mongodb-uri', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
.then(() => console.log('Connected to MongoDB'))
.catch((error) => console.error('MongoDB connection error:', error));

const userSchema = new mongoose.Schema({
  firstName: { type: String, required: true },
  lastName: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  phone: { type: String, required: true },
});
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
const User = mongoose.model('User', userSchema);

app.post('/signup', async (req, res) => {
  const { firstName, lastName, email, password, phone } = req.body;

  try {
    const existingUser = await User.findOne({ email });
    if (existingUser) return res.status(400).json({ message: 'Email already exists' });

    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = new User({ firstName, lastName, email, password: hashedPassword,
    phone });

    await newUser.save();
    res.status(201).json({ message: 'Signup successful' });
  } catch (error) {
    console.error('Signup error:', error);
    res.status(500).json({ message: 'Server error' });
  }
});

app.post('/login', async (req, res) => {
  const { email, password } = req.body;

  try {
    const user = await User.findOne({ email });
    if (!user) return res.status(400).json({ message: 'Invalid email or password' });

    const validPassword = await bcrypt.compare(password, user.password);
    if (!validPassword) return res.status(400).json({ message: 'Invalid email or password' });

    res.status(200).json({ username: user.firstName, message: 'Login successful' });
  } catch (error) {
    console.error('Login error:', error);
    res.status(500).json({ message: 'Server error' });
  }
});

app.listen(PORT, () => console.log(`Server running on 
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

4. Screenshots of outputs:

The screenshot shows a login interface centered on a light gray background. The interface is contained within a white rounded rectangle. At the top of this rectangle is the title "Login" in bold black text. Below the title are two input fields: the first is labeled "Username" and the second is labeled "Password". Both fields have a light gray border and a small downward arrow on the right side. Below these fields is a prominent green button with the text "Login" in white. At the bottom of the white rectangle, there is a link that reads "Don't have an account? [Sign Up](#)", where "Sign Up" is in blue text.

5. Learning Outcomes:

- Learners will gain hands-on experience in creating a **full-stack web application** by integrating frontend, backend, and database layers using HTML, CSS, JavaScript, Node.js, Express.js, and MongoDB.
- Learners will develop knowledge of **user authentication** by implementing secure password hashing with bcrypt and handling user data validation in both frontend and backend.
- Learners will understand how to design and interact with **RESTful APIs** for signup and login functionality, and manage data storage and retrieval using MongoDB.