LIVE TO LEARN & LEARN TO SHARE

MRCET CAMPUS

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS INSTITUTION - UGC, GOVT. OF INDIA)

Affiliated to JNTUH; Approved by AICTE, NBA-Tier 1 & NAAC with A-GRADE | ISO 9001:2015

Maisammaguda, Dhulapally, Komaplly, Secunderabad - 500100, Telangana State, India

LABORATORY MANUAL & RECORD

Name:.....

Roll No:.....Branch:.....

Year:.....Sem:.....

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS INSTITUTION - UGC, GOVT. OF INDIA)

Affiliated to JNTUH; Approved by AICTE, NBA-Tier 1 & NAAC with A-GRADE | ISO 9001:2015

Maisammaguda, Dhulapally, Komaplly, Secunderabad - 500100, Telangana State, India

Certificate

Certified that this is the Bonafide Record of the Work Done by

Mr./Ms.....Roll.No.....of

B.Tech.....year..... Semester for Academic year.....

in.....Laboratory.

Date: Faculty Incharge HOD

Internal Examiner External Examiner

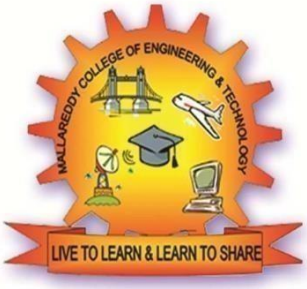
INDEX

S.No	Date	Name of the Activity/Experiment	Grade/ Marks	Faculty Signature

FULL STACK DEVELOPMENT LAB MANUAL

(R22A0513)

B.TECH



(III YEAR – II SEM)
(2024-25)

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE

(CSE-AIML &B.TECH AIML)

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956
(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

Department of Computer Science & Engineering

(Artificial Intelligence & Machine Learning)

Vision

To be a premier centre for academic excellence and research through innovative interdisciplinary collaborations and making significant contributions to the community, organizations, and society as a whole.

Mission

- To impart cutting-edge Artificial Intelligence technology in accordance with industry norms.
- To instill in students a desire to conduct research in order to tackle challenging technical problems for industry.
- To develop effective graduates who are responsible for their professional growth, leadership qualities and are committed to lifelong learning.

Quality Policy

- To provide sophisticated technical infrastructure and to inspire students to reach their full potential.
- To provide students with a solid academic and research environment for a comprehensive learning experience.
- To provide research development, consulting, testing, and customized training to satisfy specific industrial demands, thereby encouraging self-employment and entrepreneurship among students.

Program Specific Outcomes

1. Ability to design, develop, and deploy dynamic web applications using modern web technologies.
2. Ability to create responsive, mobile-friendly websites that adapt to various screen sizes and devices.
3. Knowledge of server-side programming languages and frameworks to develop the backend of a web application
4. Ability to use version control tools for collaboration and code management
5. Understanding and applying web accessibility principles to ensure websites are usable by people with disabilities
6. Proficiency in using client-side programming to enhance user interaction and create dynamic, interactive user experiences

Program Educational Objectives

1. Students will work in roles such as front-end developer, back-end developer, full-stack developer, UI/UX designer, or web application architect
2. Students will engage in continuous learning, adopting the latest web development trends, tools, and best practices such as AI-based web tools, serverless architecture, or Progressive Web Apps (PWAs)
3. Students will be well-versed in writing clean, maintainable code, and use automated testing, version control systems like Git, and debugging tools to produce high-quality solutions.
4. Students will have the skills to launch start-ups, develop personal projects, or contribute to innovation-driven organizations, applying creativity and problem-solving in real-world scenarios.
5. Students will effectively present their ideas, explain complex technical concepts, and document their work in ways that are understandable to clients, colleagues, and stakeholders.
6. Students will participate in projects or initiatives that address societal challenges, such as creating platforms for education, healthcare, social welfare, or sustainable development

PROGRAM OUTCOMES (POs)

Engineering Graduates should possess the following:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design / development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society**: Apply reasoning informed by the contextual knowledge toassess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work**: Function effectively as an individual, and as a member or leaderin diverse teams, and in multidisciplinary settings.

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi-disciplinary environments.

12. **Life- long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

Maisammaguda, Dhulapally Post, Via Hakimpet, Secunderabad – 500100

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE (CSE-AIML)

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student shouldenter into the laboratory with:

a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.

b. Laboratory Record updated up to the last session experiments and other utensils (if any) neededin the lab.

c. Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation notebook, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high-end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is keptproperly.

LAB OBJECTIVES:

This course will enable the students:

1. Usage of various front and back end Tools
2. They can understand and create applications on their own
3. Demonstrate and Designing of Websites can be carried out.
4. Develop web based application using suitable client side and server side code.
5. Implement web based application using effective database access.

LAB OUTCOMES:

Students will be able to understand

1. Usage of various front and back end Tools
2. They can understand and create applications on their own
3. Demonstrate and Designing of Websites can be carried out.
4. Develop web based application using suitable client side and server side code.
5. Implement web based application using effective database access.

Head of the Department **Principal**

Guidelines to students

A. Standard operating procedure

1) Name of the experiment
2) Aim
3) Software/Hardware requirements
4) Writing the python programs by the students
5) Commands for executing programs

Writing of the experiment in the Observation Book

The students will write the today's experiment in the Observation book as per the following format:

a) Name of the experiment
b) Aim
c) Writing the program
d) Viva-Voce Questions and Answers
e) Errors observed (if any)during compilation/execution

Signature of the Faculty

Instructions to maintain the record

- Before start of the first lab they have to buy their record and bring their record to the lab.
- Regularly (Weekly) update the record after completion of the experiment and get it corrected with concerned lab in-charge for continuous evaluation. In case the record is lost inform the same day to the faculty in charge and get the new record within 2 days the record has to be submitted and get it corrected by the faculty.
- If record is not submitted in time or record is not written properly, the evaluation marks (5M) will be deducted.

Awarding the marks for day to day evaluation

Total marks for day to day evaluation is 15 Marks as per Autonomous (JNTUH). These 15 Marks are distributed as:

Regularity	3Marks
Program written	3Marks
Execution & Result	3Marks
Viva-Voce	3Marks
Dress Code	3Marks

Allocation of Marks for Lab Internal

Total marks for lab internal are 40 Marks as per Autonomous (JNTUH.)

These 40 Marks are distributed as:

Average of Continuous Evaluation marks:15

Marks Lab exam:15 Marks

VIVA:10 Marks

Allocation of Marks for Lab External

Total marks for lab Internal and External are 40Marks as per Autonomous/ (JNTUH).

These 60 External Lab Marks are distributed as:

Program Writeup	20 Marks
Program Execution and Result	20 Marks
Viva-Voce	10 Marks
Record	10 Marks

INDEX

Week-1. **Designing following static WebPages required for an Online Book Store website**

CODE:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Online Bookstore</title>
   <style>
      * {
         margin: 0;
         padding: 0;
         box-sizing: border-box;
      }

      body {
         font-family: Arial, sans-serif;
         line-height: 1.6;
      }

      header {
         background-color: #333;
         color: white;
         padding: 10px 0;
         text-align: center;
      }

      header .logo h1 {
         margin: 0;
      }

      nav ul {
         list-style-type: none;
         padding: 0;
      }

      nav ul li {
         display: inline;
         margin: 0 15px;
      }

      nav ul li a {
         color: white;
         text-decoration: none;
         font-size: 1.1em;
      }

      nav ul li a:hover {
```

```
    color: #f0a500;
}

section.hero {
    background-color: #f4f4f4;
    padding: 20px;
    text-align: center;
}

section.catalog {
    display: flex;
    justify-content: space-around;
    flex-wrap: wrap;
    padding: 20px;
}

section.catalog .book {
    width: 30%;
    margin-bottom: 20px;
    text-align: center;
}

section.catalog .book img {
    width: 100%;
    height: auto;
}

section.catalog .book h3 {
    font-size: 1.2em;
    margin: 10px 0;
}

section.catalog .book p {
    font-size: 1em;
    color: #555;
}

section.catalog .book a {
    background-color: #333;
    color: white;
    padding: 10px;
    text-decoration: none;
    display: inline-block;
    margin-top: 10px;
}

section.catalog .book a:hover {
    background-color: #f0a500;
}

section.contact {
    padding: 20px;
```

```
    }

    section.contact form {
      max-width: 600px;
      margin: 0 auto;
    }

    section.contact form input,
    section.contact form textarea {
      width: 100%;
      padding: 10px;
      margin-bottom: 10px;
      border: 1px solid #ccc;
      border-radius: 4px;
    }

    section.contact form button {
      background-color: #333;
      color: white;
      padding: 10px 20px;
      border: none;
      cursor: pointer;
    }

    section.contact form button:hover {
      background-color: #f0a500;
    }

    footer {
      background-color: #333;
      color: white;
      text-align: center;
      padding: 10px;
      margin-top: 20px;
    }
  </style>
</head>
<body>

  <header>
    <div class="logo">
      <h1>Online Bookstore</h1>
    </div>
    <nav>
      <ul>
        <li><a href="#home">Home</a></li>
        <li><a href="#catalog">Catalog</a></li>
        <li><a href="#contact">Contact</a></li>
      </ul>
    </nav>
  </header>
```

```
<!-- Hero Section -->
<section id="home" class="hero">
    <h2>Welcome to the Best Bookstore!</h2>
    <p>Browse and shop from a wide variety of books across all genres.</p>
</section>

<!-- Book Catalog -->
<section id="catalog" class="catalog">
    <h2>Our Book Collection</h2>
    <div class="book">
        <img src="https://via.placeholder.com/150" alt="Book Cover">
        <h3>Book Title 1</h3>
        <p>Description of Book 1.</p>
        <a href="#">Buy Now</a>
    </div>
    <div class="book">
        <img src="https://via.placeholder.com/150" alt="Book Cover">
        <h3>Book Title 2</h3>
        <p>Description of Book 2.</p>
        <a href="#">Buy Now</a>
    </div>
    <div class="book">
        <img src="https://via.placeholder.com/150" alt="Book Cover">
        <h3>Book Title 3</h3>
        <p>Description of Book 3.</p>
        <a href="#">Buy Now</a>
    </div>
</section>

<!-- Contact Section -->
<section id="contact" class="contact">
    <h2>Contact Us</h2>
    <p>If you have any questions, feel free to reach out!</p>
    <form>
        <label for="name">Your Name:</label>
        <input type="text" id="name" name="name" required>

        <label for="email">Your Email:</label>
        <input type="email" id="email" name="email" required>

        <label for="message">Your Message:</label>
        <textarea id="message" name="message" rows="4" required></textarea>

        <button type="submit">Send Message</button>
    </form>
</section>

<footer>
    <p>&copy; 2024 Online Bookstore. All rights reserved.</p>
</footer>

<!-- JavaScript -->
```

```
    <script>
        document.querySelector('form').addEventListener('submit', function(e) {
            e.preventDefault();
            alert('Thank you for contacting us! We will get back to you soon.');
        });
    </script>
</body>
</html>
```

OUTPUT:

EXERCISE PROGRAM:

Week-2. Designing a webpage using CSS Which includes different styles

CODE:

Complete Webpage Code with CSS

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Stylish Webpage</title>
    <style>
        /* Reset some default styles */
        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        body {
            font-family: 'Arial', sans-serif;
            line-height: 1.6;
            background-color: #f4f4f9;
            color: #333;
        }

        /* Header styles */
        header {
            background-color: #333;
            color: #fff;
            padding: 20px;
            text-align: center;
        }

        header h1 {
            font-size: 2.5em;
            margin: 0;
            font-weight: bold;
        }

        header nav {
            margin-top: 10px;
        }

        header nav ul {
            list-style-type: none;
```

```
}

header nav ul li {
    display: inline-block;
    margin: 0 15px;
}

header nav ul li a {
    text-decoration: none;
    color: #fff;
    font-size: 1.2em;
}

header nav ul li a:hover {
    color: #f0a500;
}

/* Main content area */
.main-content {
    display: flex;
    justify-content: space-around;
    padding: 40px 20px;
    margin-top: 30px;
}

.content-box {
    background-color: #fff;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    border-radius: 10px;
    padding: 20px;
    width: 30%;
    text-align: center;
    transition: transform 0.3s ease;
}

.content-box:hover {
    transform: scale(1.05);
}

.content-box h2 {
    color: #333;
    font-size: 1.8em;
    margin-bottom: 20px;
}

.content-box p {
    font-size: 1.1em;
    color: #555;
}

.content-box button {
    background-color: #333;
```

```
    color: white;
    border: none;
    padding: 10px 20px;
    border-radius: 5px;
    cursor: pointer;
    font-size: 1.1em;
    margin-top: 20px;
}

.content-box button:hover {
    background-color: #f0a500;
}

/* Footer styles */
footer {
    background-color: #333;
    color: #fff;
    text-align: center;
    padding: 20px;
    margin-top: 40px;
}

footer p {
    font-size: 1em;
}

/* Responsive Styles */
@media (max-width: 768px) {
    .main-content {
        flex-direction: column;
        align-items: center;
    }

    .content-box {
        width: 80%;
        margin-bottom: 20px;
    }

    header h1 {
        font-size: 2em;
    }

    header nav ul li {
        margin: 0 10px;
    }

    header nav ul li a {
        font-size: 1em;
    }
}

/* Button Styles */
```

```
        .btn-primary {
            background-color: #4CAF50;
            color: white;
            padding: 10px 20px;
            font-size: 1.2em;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            transition: background-color 0.3s ease;
        }

        .btn-primary:hover {
            background-color: #45a049;
        }

        .btn-secondary {
            background-color: #f44336;
            color: white;
            padding: 10px 20px;
            font-size: 1.2em;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            transition: background-color 0.3s ease;
        }

        .btn-secondary:hover {
            background-color: #e53935;
        }

        /* Section title styles */
        .section-title {
            font-size: 2.2em;
            text-align: center;
            color: #333;
            margin-bottom: 30px;
        }

    </style>
</head>
<body>

    <header>
        <h1>My Stylish Webpage</h1>
        <nav>
            <ul>
                <li><a href="#">Home</a></li>
                <li><a href="#">About</a></li>
                <li><a href="#">Services</a></li>
                <li><a href="#">Contact</a></li>
            </ul>
        </nav>
```

```
    </header>

    <!-- Main Content -->
    <section class="main-content">
        <div class="content-box">
            <h2>Welcome to Our Website</h2>
            <p>We offer high-quality services tailored to your needs. Browse through our offerings and feel free to
contact us.</p>
            <button class="btn-primary">Learn More</button>
        </div>
        <div class="content-box">
            <h2>Our Services</h2>
            <p>Explore our wide range of services. We specialize in web design, digital marketing, and more.</p>
            <button class="btn-secondary">View Services</button>
        </div>
        <div class="content-box">
            <h2>Contact Us</h2>
            <p>Have a question or ready to get started? Reach out to us, and we'll be happy to assist you!</p>
            <button class="btn-primary">Get in Touch</button>
        </div>
    </section>

    <!-- Footer -->
    <footer>
        <p>&copy; 2024 My Stylish Webpage | All Rights Reserved</p>
    </footer>

</body>
</html>
```

OUTPUT:

EXERCISE PROGRAM:

Week-3. Write a JavaScript to implement the following various events.

CODE:

Example JavaScript for Handling Various Events:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>JavaScript Events Example</title>
   <style>
      body {
         font-family: Arial, sans-serif;
         line-height: 1.6;
         background-color: #f4f4f9;
         color: #333;
         text-align: center;
         padding: 20px;
      }

      button {
         background-color: #4CAF50;
         color: white;
         padding: 10px 20px;
         font-size: 1.2em;
         border: none;
         border-radius: 5px;
         cursor: pointer;
         transition: background-color 0.3s ease;
      }

      button:hover {
         background-color: #45a049;
      }

      .input-box {
         margin: 10px 0;
         padding: 10px;
         font-size: 1em;
         width: 300px;
         border-radius: 5px;
         border: 1px solid #ccc;
      }

      .message {
```

```
            margin-top: 20px;
            font-size: 1.5em;
            color: #f44336;
        }
    </style>
</head>
<body>

    <h1>JavaScript Event Handling Example</h1>

    <!-- Button that triggers click event -->
    <button id="clickButton">Click Me!</button>

    <!-- Input box that triggers focus and blur events -->
    <input type="text" id="focusInput" class="input-box" placeholder="Focus and blur events" />

    <!-- Input box that triggers keydown event -->
    <input type="text" id="keyInput" class="input-box" placeholder="Type something (keydown event)" />

    <!-- Form with submit event -->
    <form id="submitForm">
        <input type="text" id="formInput" class="input-box" placeholder="Enter text to submit" />
        <button type="submit">Submit Form</button>
    </form>

    <!-- Div to display event messages -->
    <div id="eventMessage" class="message"></div>

    <script>
        // Click event on button
        document.getElementById("clickButton").addEventListener("click", function() {
            document.getElementById("eventMessage").textContent = "Button Clicked!";
        });

        // Focus event on input
        document.getElementById("focusInput").addEventListener("focus", function() {
            document.getElementById("eventMessage").textContent = "Input field focused!";
        });

        // Blur event on input
        document.getElementById("focusInput").addEventListener("blur", function() {
            document.getElementById("eventMessage").textContent = "Input field blurred!";
        });

        // Keydown event on input (display key pressed)
        document.getElementById("keyInput").addEventListener("keydown", function(event) {
            document.getElementById("eventMessage").textContent = "You pressed: " + event.key;
        });

        // Submit event on form
        document.getElementById("submitForm").addEventListener("submit", function(e) {
            e.preventDefault(); // Prevent form from actually submitting
```

```
        document.getElementById("eventMessage").textContent = "Form submitted with: " +
document.getElementById("formInput").value;
        document.getElementById("formInput").value = ''; // Clear input field after submit
      });
  </script>

</body>
</html>
```

OUTPUT:

EXERCISE PROGRAM:

Week-4. Write a program to create and Build a Password Strength Check using JQuery.

CODE:

To create a Password Strength Checker using jQuery, we will build a program that evaluates the strength of a password based on different criteria such as:

1. Length of the password.
2. Presence of uppercase letters, lowercase letters, numbers, and special characters.
3. Display a strength meter that updates as the user types the password.

Here's a step-by-step guide and code implementation for a Password Strength Checker using jQuery:

HTML + CSS + jQuery Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Password Strength Checker</title>
   <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
   <style>
      body {
         font-family: Arial, sans-serif;
         padding: 20px;
         background-color: #f4f4f9;
      }

      h2 {
         text-align: center;
      }

      .password-container {
         width: 300px;
         margin: 0 auto;
         padding: 20px;
         background-color: #fff;
         border-radius: 8px;
         box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
      }

      .password-container input {
         width: 100%;
         padding: 10px;
         margin-bottom: 10px;
         border: 1px solid #ccc;
```

```
            border-radius: 5px;
            font-size: 16px;
        }

        .password-strength {
            height: 10px;
            border-radius: 5px;
            transition: width 0.4s ease;
        }

        .strength-weak {
            background-color: red;
        }

        .strength-medium {
            background-color: orange;
        }

        .strength-strong {
            background-color: green;
        }

        .strength-message {
            font-size: 14px;
            text-align: center;
            margin-top: 5px;
        }

        .strength-message span {
            font-weight: bold;
        }

    </style>
</head>
<body>

    <h2>Password Strength Checker</h2>

    <div class="password-container">
        <input type="password" id="password" placeholder="Enter your password" />
        <div id="passwordStrength" class="password-strength"></div>
        <div id="strengthMessage" class="strength-message"></div>
    </div>

    <script>
        $(document).ready(function() {
            $('#password').on('input', function() {
                var password = $(this).val();
                var strength = 0;
                var strengthMessage = '';

                // Check length (must be at least 8 characters)
```

```
            if (password.length >= 8) {
                strength += 1;
            }

            // Check for uppercase letter
            if (/[A-Z]/.test(password)) {
                strength += 1;
            }

            // Check for lowercase letter
            if (/[a-z]/.test(password)) {
                strength += 1;
            }

            // Check for number
            if (/\d/.test(password)) {
                strength += 1;
            }

            // Check for special character
            if (/[^A-Za-z0-9]/.test(password)) {
                strength += 1;
            }

            // Set strength level and message
            var strengthPercent = (strength / 5) * 100;
            $('#passwordStrength').width(strengthPercent + '%');

            // Determine password strength
            if (strength <= 1) {
                $('#passwordStrength').removeClass().addClass('strength-weak');
                strengthMessage = '<span>Weak</span> password. Try adding uppercase letters, numbers, and special
characters.';
            } else if (strength <= 3) {
                $('#passwordStrength').removeClass().addClass('strength-medium');
                strengthMessage = '<span>Medium</span> password. Add more characters or special characters to
strengthen it.';
            } else {
                $('#passwordStrength').removeClass().addClass('strength-strong');
                strengthMessage = '<span>Strong</span> password. Great job!';
            }

            // Display strength message
            $('#strengthMessage').html(strengthMessage);
        });
    });
  </script>

</body>
</html>
```

OUTPUT:

EXERCISE PROGRAM:

Week-5. Write a program to create and Build a star rating system using JQuery

CODE:

Steps:

1. **HTML:** We'll create a series of clickable star elements.
2. **CSS:** We'll style the stars, showing an empty star (for unselected) and a filled star (for selected).
3. **jQuery:** We'll use jQuery to handle user interactions, like when a user clicks on a star to rate something.

Here's a simple implementation of a star rating system:

HTML + CSS + jQuery Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Star Rating System</title>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 50px;
            text-align: center;
            background-color: #f4f4f9;
        }

        .rating {
            font-size: 40px;
            direction: rtl; /* Makes it easier to handle clicks from right to left */
            display: inline-block;
        }

        .rating span {
            cursor: pointer;
            color: #ccc; /* Empty stars color */
            transition: color 0.3s ease;
        }

        .rating span.filled {
            color: gold; /* Filled stars color */
        }

        .rating span:hover,
        .rating span:hover ~ span {
            color: gold; /* Hover effect */
```

```
        }

        .rating-result {
            font-size: 18px;
            margin-top: 20px;
            font-weight: bold;
            color: #333;
        }
    </style>
</head>
<body>

    <h1>Star Rating System</h1>
    <div class="rating">
        <span data-value="5">★</span>
        <span data-value="4">★</span>
        <span data-value="3">★</span>
        <span data-value="2">★</span>
        <span data-value="1">★</span>
    </div>

    <div class="rating-result">
        Rating: <span id="ratingValue">0</span> / 5
    </div>

    <script>
        $(document).ready(function() {
            let currentRating = 0;

            // Hover effect - when mouse is over a star, highlight it
            $('.rating span').on('mouseenter', function() {
                let value = $(this).data('value');
                // Fill the stars up to the hovered one
                $(this).prevAll().addBack().addClass('filled');
                $(this).nextAll().removeClass('filled');
            });

            // Remove hover effect when mouse leaves
            $('.rating span').on('mouseleave', function() {
                let value = currentRating; // Maintain the current rating
                // Fill the stars based on the current rating
                $('.rating span').each(function() {
                    if ($(this).data('value') <= value) {
                        $(this).addClass('filled');
                    } else {
                        $(this).removeClass('filled');
                    }
                });
            });

            // Handle star click event to set the rating
```

```
        $('.rating span').on('click', function() {
            currentRating = $(this).data('value');
            // Update the rating result
            $('#ratingValue').text(currentRating);

            // Highlight the stars based on the clicked rating
            $('.rating span').each(function() {
                if ($(this).data('value') <= currentRating) {
                    $(this).addClass('filled');
                } else {
                    $(this).removeClass('filled');
                }
            });
        });
    });
  </script>

</body>
</html>
```

OUTPUT:

EXERCISE PROGRAM:

Week-6. Write a program for sending request to a server by using AJAX.

To send a request to a server using AJAX, we can use JavaScript (along with the jQuery library) to send HTTP requests like GET, POST, or other methods. AJAX allows us to send data to a server and receive a response reloading the webpage.

Below is a simple example of sending a request to the server using AJAX with jQuery:

Example Program: Sending a GET Request to a Server Using AJAX

In this example, we'll send a GET request to a server, fetch some data, and display the result on the page.

HTML + jQuery + AJAX Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>AJAX Request Example</title>
   <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
   <style>
      body {
         font-family: Arial, sans-serif;
         padding: 20px;
         background-color: #f4f4f9;
      }

      h2 {
         text-align: center;
      }

      #response {
         font-size: 18px;
         margin-top: 20px;
         padding: 10px;
         border: 1px solid #ddd;
         background-color: #f9f9f9;
      }

      .loading {
         color: #888;
      }
   </style>
</head>
<body>

   <h2>AJAX Request Example</h2>
```

```
    <button id="getRequestButton">Send GET Request</button>

    <div id="response" class="loading">
        Waiting for response...
    </div>

    <script>
        $(document).ready(function() {
            // Handle the button click to send the GET request
            $('#getRequestButton').on('click', function() {
                // Change the response area to indicate loading
                $('#response').text('Loading...');

                // Send an AJAX GET request
                $.ajax({
                    url: 'https://jsonplaceholder.typicode.com/posts/1',  // This is a placeholder API
                    type: 'GET',
                    dataType: 'json',  // Expecting JSON response
                    success: function(data) {
                        // If the request is successful, display the data
                        $('#response').html('<strong>Title:</strong> ' + data.title + '<br><strong>Body:</strong> ' +
data.body);
                    },
                    error: function(xhr, status, error) {
                        // If there is an error, display it
                        $('#response').html('<strong>Error:</strong> Unable to fetch data.');
                    }
                });
            });
        });
    </script>

</body>
</html>
```

OUTPUT:

EXERCISE PROGRAM:

Week-7. Develop an Angular JS application that displays a list of shopping items. Allow users to add and remove items from the list using directives and controllers. Note: The default values of items may be included in the program

To develop an AngularJS application that displays a list of shopping items and allows users to add and remove items, we'll need to create a simple AngularJS app with the following features:

1. **Display a list of items.**
2. **Allow users to add new items to the list.**
3. **Allow users to remove items from the list.**
4. **Use AngularJS directives (ng-repeat, ng-click) and controllers.**

Here's how we can create such an application:

Steps:

1. **Set up an AngularJS application.**
2. **Create an HTML interface to display the shopping list.**
3. **Use AngularJS controllers and directives to handle the list of items.**

HTML + AngularJS Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Shopping List App</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
    <style>
        body {
            font-family: Arial, sans-serif;
            padding: 20px;
            background-color: #f9f9f9;
        }

        h2 {
            text-align: center;
            color: #4CAF50;
        }

        .shopping-list {
            max-width: 500px;
            margin: 0 auto;
            background-color: #fff;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
```

```
        }

        .shopping-list input[type="text"] {
            width: 80%;
            padding: 8px;
            font-size: 16px;
            border-radius: 4px;
            border: 1px solid #ccc;
            margin-right: 10px;
        }

        .shopping-list button {
            padding: 8px 16px;
            font-size: 16px;
            border: none;
            border-radius: 4px;
            background-color: #4CAF50;
            color: white;
            cursor: pointer;
        }

        .shopping-list button:hover {
            background-color: #45a049;
        }

        .item {
            display: flex;
            justify-content: space-between;
            align-items: center;
            margin: 10px 0;
        }

        .item button {
            background-color: #f44336;
            border: none;
            color: white;
            font-size: 16px;
            cursor: pointer;
        }

        .item button:hover {
            background-color: #e53935;
        }
    </style>
</head>
<body ng-app="shoppingApp" ng-controller="shoppingCtrl">

    <h2>Shopping List</h2>

    <div class="shopping-list">
        <input type="text" ng-model="newItem" placeholder="Enter item" />
        <button ng-click="addItem()">Add Item</button>
```

```
    <div ng-if="shoppingList.length > 0">
      <h3>Items:</h3>
      <ul>
        <li ng-repeat="item in shoppingList">
          <div class="item">
            <span>{{ item }}</span>
            <button ng-click="removeItem($index)">Remove</button>
          </div>
        </li>
      </ul>
    </div>

    <div ng-if="shoppingList.length === 0">
      <p>No items in the list.</p>
    </div>
  </div>

  <script>
    // AngularJS Application
    var app = angular.module("shoppingApp", []);

    // Controller for the shopping list
    app.controller("shoppingCtrl", function($scope) {
      // Initialize the shopping list
      $scope.shoppingList = [];

      // Add item to the shopping list
      $scope.addItem = function() {
        if ($scope.newItem && $scope.newItem.trim() !== "") {
          $scope.shoppingList.push($scope.newItem.trim());
          $scope.newItem = ""; // Clear the input field
        }
      };

      // Remove item from the shopping list
      $scope.removeItem = function(index) {
        $scope.shoppingList.splice(index, 1);
      };
    });
  </script>

</body>
</html>
```

OUTPUT:

EXERCISE PROGRAM:

Week-8. Write a program to create a simple calculator Application using React JS

Creating a simple calculator application using React JS involves using React components, managing state, and handling events like button clicks. Below is a step-by-step guide to creating a calculator app in React.

Steps:

1. **Set up React environment** (You can use tools like Create React App to set up a React project).
2. **Create components**: We'll use one component for the entire calculator.
3. **Manage state**: We need to keep track of the current input, the operation to perform, and the result.
4. **Handle events**: Handle button clicks for numbers and operations.

Simple Calculator in React

Step-by-Step Code:

1. Set up the React Application (Using create-react-app):

If you haven't already created a React app, you can do so by running the following command:

```
npx create-react-app react-calculator
cd react-calculator
npm start
```

2. Modify the App.js File:

Here's how to write the code for the calculator:

```
import React, { useState } from 'react';
import './App.css';

function App() {
  // State to store the current value, operation, and previous value
  const [currentInput, setCurrentInput] = useState('');
  const [previousInput, setPreviousInput] = useState('');
  const [operation, setOperation] = useState('');

  // Function to handle number button clicks
  const handleNumberClick = (number) => {
    setCurrentInput(currentInput + number);
  };

  // Function to handle operation button clicks (+, -, *, /)
  const handleOperationClick = (operator) => {
    setOperation(operator);
    setPreviousInput(currentInput);
    setCurrentInput('');
  };

  // Function to clear the current input
```

```
const handleClear = () => {
  setCurrentInput('');
  setPreviousInput('');
  setOperation('');
};

// Function to evaluate the result of the current operation
const handleEvaluate = () => {
  if (previousInput && operation && currentInput) {
    const prev = parseFloat(previousInput);
    const current = parseFloat(currentInput);
    let result;

    // Perform the operation
    switch (operation) {
      case '+':
        result = prev + current;
        break;
      case '-':
        result = prev - current;
        break;
      case '*':
        result = prev * current;
        break;
      case '/':
        result = prev / current;
        break;
      default:
        return;
    }

    // Set the result to the current input and clear operation and previous input
    setCurrentInput(result.toString());
    setPreviousInput('');
    setOperation('');
  }
};

// Rendering the calculator UI
return (
  <div className="calculator">
    <div className="display">
      <div className="previous-input">
        {previousInput} {operation}
      </div>
      <div className="current-input">{currentInput}</div>
    </div>
    <div className="buttons">
      <button onClick={() => handleNumberClick('7')}>7</button>
      <button onClick={() => handleNumberClick('8')}>8</button>
      <button onClick={() => handleNumberClick('9')}>9</button>
      <button onClick={() => handleOperationClick('/')}>/</button>
```

```
        <button onClick={() => handleNumberClick('4')}>4</button>
        <button onClick={() => handleNumberClick('5')}>5</button>
        <button onClick={() => handleNumberClick('6')}>6</button>
        <button onClick={() => handleOperationClick('*')}>*</button>

        <button onClick={() => handleNumberClick('1')}>1</button>
        <button onClick={() => handleNumberClick('2')}>2</button>
        <button onClick={() => handleNumberClick('3')}>3</button>
        <button onClick={() => handleOperationClick('-')}>-</button>

        <button onClick={() => handleNumberClick('0')}>0</button>
        <button onClick={handleClear}>C</button>
        <button onClick={handleEvaluate}>=</button>
        <button onClick={() => handleOperationClick('+')}>+</button>
      </div>
    </div>
  );
}

export default App;
```

3. Add Styles in App.css:

You can add some basic styles for your calculator to make it look neat:

```
.calculator {
  width: 260px;
  margin: 100px auto;
  background-color: #f4f4f4;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
}

.display {
  text-align: right;
  margin-bottom: 20px;
}

.previous-input {
  font-size: 16px;
  color: #888;
}

.current-input {
  font-size: 32px;
  font-weight: bold;
}
```

```
.buttons button {
  width: 50px;
  height: 50px;
  margin: 5px;
  font-size: 18px;
  border: 1px solid #ccc;
  background-color: #fff;
  cursor: pointer;
  border-radius: 5px;
}

.buttons button:hover {
  background-color: #f0f0f0;
}

.buttons button:active {
  background-color: #ddd;
}
```

How to Run the Application:

1. **Set up the React environment** (if not done already).
    - Use npx create-react-app react-calculator to set up a new React project.
2. **Replace App.js and App.css**:
    - Replace the default content in src/App.js with the provided React code.
    - Replace src/App.css with the provided CSS.
3. **Run the app**:
    - Run the app with npm start and open it in your browser.

OUTPUT:

EXERCISE PROGRAM:

Week-9. Write a program to create a voting application using React JS.

Creating a voting application using React JS involves several key steps: setting up the project, creating the components, managing state, and displaying the results. Here's how you can build a simple voting application:

1. Set up your project

To get started, you'll need to create a React app. If you don’t have create-react-app installed, you can install it and set up your project:

```
npx create-react-app voting-app
cd voting-app
npm start
```

2. Create Components

We will create three components for this app:

- **App.js**: The main container component for managing state.
- **VoteOptions.js**: A component to display voting options.
- **Result.js**: A component to display the results of the voting.

3. App.js

Here we will manage the state of the voting options and the number of votes for each option. We will also handle the logic for voting and displaying results.

```
import React, { useState } from 'react';
import './App.css';
import VoteOptions from './VoteOptions';
import Result from './Result';

function App() {
  const [votes, setVotes] = useState({
    option1: 0,
    option2: 0,
    option3: 0,
  });

  const handleVote = (option) => {
    setVotes({
      ...votes,
      [option]: votes[option] + 1,
    });
  };

  const totalVotes = votes.option1 + votes.option2 + votes.option3;
```

```
  return (
    <div className="App">
      <h1>Vote for your favorite option</h1>
      <VoteOptions votes={votes} onVote={handleVote} />
      {totalVotes > 0 && <Result votes={votes} />}
    </div>
  );
}

export default App;
```

4. VoteOptions.js

This component will display the voting options and allow the user to vote.

```
import React from 'react';

function VoteOptions({ votes, onVote }) {
  return (
    <div>
      <h2>Choose an Option</h2>
      <button onClick={() => onVote('option1')}>Option 1</button>
      <span> ({votes.option1} votes)</span>
      <br />
      <button onClick={() => onVote('option2')}>Option 2</button>
      <span> ({votes.option2} votes)</span>
      <br />
      <button onClick={() => onVote('option3')}>Option 3</button>
      <span> ({votes.option3} votes)</span>
      <br />
    </div>
  );
}

export default VoteOptions;
```

5. Result.js

This component will display the results of the voting.

```
import React from 'react';

function Result({ votes }) {
  const totalVotes = votes.option1 + votes.option2 + votes.option3;

  const percentage = (optionVotes) => {
    return totalVotes > 0 ? ((optionVotes / totalVotes) * 100).toFixed(2) : 0;
  };

  return (
    <div>
```

```
      <h2>Results</h2>
      <p>Option 1: {votes.option1} votes ({percentage(votes.option1)}%)</p>
      <p>Option 2: {votes.option2} votes ({percentage(votes.option2)}%)</p>
      <p>Option 3: {votes.option3} votes ({percentage(votes.option3)}%)</p>
      <p>Total Votes: {totalVotes}</p>
    </div>
  );
}

export default Result;
```

6. Styling (optional)

To make the app look better, you can add some simple styles in App.css.

```
.App {
  text-align: center;
  padding: 20px;
}

button {
  margin: 5px;
  padding: 10px;
  font-size: 16px;
}

h1 {
  color: #333;
}

h2 {
  color: #444;
}
```

7. Run the application

Once you have set up the components and styles, you can run the app using:

```
npm start
```

The application will display three voting options and allow users to vote. After a vote is cast, the results will be displayed with the percentage of votes for each option.

OUTPUT:

EXERCISE PROGRAM:

Week-10. Write a server side program for Accessing MongoDB from Node.js.

To access MongoDB from Node.js, you will need to set up a Node.js application that communicates with a MongoDB database using the mongoose package, which provides a higher-level API to interact with MongoDB.

Below are the steps to set up a server-side Node.js program to access MongoDB:

1. Set Up Your Project

Create a new directory and initialize a Node.js project.

```
mkdir node-mongo-app
cd node-mongo-app
npm init -y
```

2. Install Dependencies

You will need to install express (to create the server) and mongoose (to interact with MongoDB).

```
npm install express mongoose
```

3. Create MongoDB Database

If you don't have MongoDB running locally, you can either install MongoDB or use a cloud-based solution like MongoDB Atlas. If using MongoDB Atlas, you'll need a connection string (e.g., mongodb+srv://<username>:<password>@cluster0.mongodb.net/myDatabase).

For local MongoDB, MongoDB runs on mongodb://localhost:27017/myDatabase by default.

4. Create Server-Side Code

Let's create a basic Node.js server that interacts with MongoDB using the Mongoose library.

index.js (Main Entry Point)

```
const express = require('express');
const mongoose = require('mongoose');
const app = express();
const port = 3000;

// Middleware to parse incoming JSON requests
app.use(express.json());

// MongoDB connection string (update with your own connection string if using MongoDB Atlas)
const mongoURI = 'mongodb://localhost:27017/votingApp';

// Connect to MongoDB using Mongoose
mongoose.connect(mongoURI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
```

```
})
  .then(() => console.log('Connected to MongoDB'))
  .catch((err) => console.log('Error connecting to MongoDB:', err));

// Define a Schema for the Vote (Model)
const voteSchema = new mongoose.Schema({
  option: String,
  votes: { type: Number, default: 0 },
});

const Vote = mongoose.model('Vote', voteSchema);

// Route to get all vote options
app.get('/votes', async (req, res) => {
  try {
    const votes = await Vote.find();
    res.status(200).json(votes);
  } catch (err) {
    res.status(500).json({ message: 'Error retrieving votes', error: err });
  }
});

// Route to cast a vote (update the votes for a given option)
app.post('/vote', async (req, res) => {
  const { option } = req.body;

  try {
    let vote = await Vote.findOne({ option });

    if (!vote) {
      // If the option doesn't exist, create a new vote option
      vote = new Vote({ option, votes: 1 });
      await vote.save();
    } else {
      // Increment the votes for the existing option
      vote.votes += 1;
      await vote.save();
    }

    res.status(200).json(vote);
  } catch (err) {
    res.status(500).json({ message: 'Error casting vote', error: err });
  }
});

// Route to get the results of the voting
app.get('/results', async (req, res) => {
  try {
    const results = await Vote.find().sort({ votes: -1 }); // Sort by votes in descending order
    res.status(200).json(results);
  } catch (err) {
    res.status(500).json({ message: 'Error retrieving results', error: err });
```

```
  }
});

// Start the server
app.listen(port, () => {
  console.log(`Server running on http://localhost:${port}`);
});
```

5. Create MongoDB Schema (Optional)

In the code above, we define a Mongoose schema to structure the vote options. Each vote option will have:

- option: The name of the voting option (e.g., "Option 1").
- votes: The number of votes for that option (default is 0).

This schema is used to create the Vote model that interacts with the MongoDB database.

6. Run MongoDB (for Local MongoDB)

Make sure you have MongoDB running locally:

```
mongod
```

7. Test the API

Now you can test your API using Postman or CURL:

- **Get all vote options**: GET http://localhost:3000/votes
- **Cast a vote**: POST http://localhost:3000/vote with a JSON body like:

```
{
  "option": "Option 1"
}
```

- **Get voting results**: GET http://localhost:3000/results

8. Optional Improvements

You can further improve this application with features such as:

- Validation (using Joi or express-validator).
- Authentication (e.g., with JWT or sessions).
- Rate-limiting to avoid abuse.
- Storing user votes to ensure each user can vote only once.

9. Final Folder Structure

```
node-mongo-app/
│
├── node_modules/        # Dependencies
├── package.json         # Project metadata and dependencies
├── index.js             # Main server file
└── package-lock.json    # Locked dependencies version
```

10. Running the Application

To run the server:

node index.js

The server will now be running on http://localhost:3000 and you can interact with it using Postman or any HTTP client.

OUTPUT:

EXERCISE PROGRAM:

Week-11. Write a server side program for Manipulating MongoDB from Node.js

To manipulate MongoDB from Node.js, we can use the mongoose library, which is an Object Data Modeling (ODM) library that provides a higher-level API for interacting with MongoDB.

Here's a complete guide and example server-side program that demonstrates basic MongoDB manipulations, such as **creating**, **reading**, **updating**, and **deleting** data (CRUD operations).

Steps:

1. **Set up Node.js and MongoDB dependencies.**
2. **Connect to MongoDB using Mongoose.**
3. **Create a Mongoose schema and model.**
4. **Implement CRUD operations.**
5. **Set up the server using Express.**

1. Set Up Project

To begin, initialize your Node.js project and install necessary dependencies.

```
mkdir node-mongo-crud
cd node-mongo-crud
npm init -y
npm install express mongoose
```

2. Server and MongoDB Connection

Create a file called index.js for the server-side logic. This will handle the database connection and the routes for CRUD operations.

```
const express = require('express');
const mongoose = require('mongoose');
const app = express();
const port = 3000;

// Middleware to parse incoming JSON requests
app.use(express.json());

// MongoDB connection URI (replace with your connection string if using MongoDB Atlas)
const mongoURI = 'mongodb://localhost:27017/myDatabase';

// Connect to MongoDB
mongoose.connect(mongoURI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
  .then(() => console.log('Connected to MongoDB'))
  .catch((err) => console.error('Error connecting to MongoDB:', err));
```

```
// Define a Schema (Model) for MongoDB
const itemSchema = new mongoose.Schema({
  name: String,
  description: String,
  price: Number,
});

const Item = mongoose.model('Item', itemSchema);

// --- CRUD Operations ---

// Create (POST) - Create a new item
app.post('/items', async (req, res) => {
  try {
    const { name, description, price } = req.body;
    const newItem = new Item({ name, description, price });
    await newItem.save();
    res.status(201).json(newItem);
  } catch (err) {
    res.status(500).json({ message: 'Error creating item', error: err });
  }
});

// Read (GET) - Get all items
app.get('/items', async (req, res) => {
  try {
    const items = await Item.find();
    res.status(200).json(items);
  } catch (err) {
    res.status(500).json({ message: 'Error fetching items', error: err });
  }
});

// Read (GET) - Get a single item by ID
app.get('/items/:id', async (req, res) => {
  try {
    const item = await Item.findById(req.params.id);
    if (!item) {
      return res.status(404).json({ message: 'Item not found' });
    }
    res.status(200).json(item);
  } catch (err) {
    res.status(500).json({ message: 'Error fetching item', error: err });
  }
});

// Update (PUT) - Update an existing item by ID
app.put('/items/:id', async (req, res) => {
  try {
    const { name, description, price } = req.body;
    const updatedItem = await Item.findByIdAndUpdate(
```

```
      req.params.id,
      { name, description, price },
      { new: true }
    );
    if (!updatedItem) {
      return res.status(404).json({ message: 'Item not found' });
    }
    res.status(200).json(updatedItem);
  } catch (err) {
    res.status(500).json({ message: 'Error updating item', error: err });
  }
});

// Delete (DELETE) - Delete an item by ID
app.delete('/items/:id', async (req, res) => {
  try {
    const deletedItem = await Item.findByIdAndDelete(req.params.id);
    if (!deletedItem) {
      return res.status(404).json({ message: 'Item not found' });
    }
    res.status(200).json({ message: 'Item deleted successfully' });
  } catch (err) {
    res.status(500).json({ message: 'Error deleting item', error: err });
  }
});

// Start the server
app.listen(port, () => {
  console.log(`Server running on http://localhost:${port}`);
});
```

3. Explanation of Operations

Let's break down the routes used for CRUD operations.

- **Create (POST) /items**:
  - Accepts a request body containing name, description, and price.
  - A new item is created and saved to MongoDB.

```
{
  "name": "Item 1",
  "description": "A sample item",
  "price": 100
}
```

- **Read (GET) /items**:
  - Returns all items in the Item collection.
- **Read (GET) /items/:id**:
  - Fetches a single item by its ID.
- **Update (PUT) /items/:id**:
  - Updates an existing item by its ID with new values for name, description, and price.

```
{
  "name": "Updated Item",
  "description": "Updated description",
  "price": 150
}
```

- **Delete (DELETE) /items/:id**:
  - Deletes an item by its ID.

4. Running the Application

1. Make sure you have MongoDB running locally (mongod) or connect to a cloud database like MongoDB Atlas.
2. To run the Node.js server, use the following command in the terminal:

```
node index.js
```

3. The server should now be running at http://localhost:3000. You can use tools like **Postman** or **cURL** to test the API routes.

5. Testing with Postman or cURL

POST (Create a new item)

```
POST http://localhost:3000/items
{
  "name": "Laptop",
  "description": "A high-end laptop",
  "price": 1200
}
```

GET (Get all items)

```
GET http://localhost:3000/items
```

GET (Get a single item by ID)

```
GET http://localhost:3000/items/5f79a8b5bfb8f774ea517d47
```

PUT (Update an item)

```
PUT http://localhost:3000/items/5f79a8b5bfb8f774ea517d47
{
  "name": "Gaming Laptop",
  "description": "A laptop with high-end specs",
  "price": 1500
}
```

DELETE (Delete an item)

```
DELETE http://localhost:3000/items/5f79a8b5bfb8f774ea517d47
```

6. Final Folder Structure

```
node-mongo-crud/
│
├── node_modules/        # Dependencies
├── package.json         # Project metadata and dependencies
├── index.js             # Server code handling CRUD operations
└── package-lock.json    # Locked dependencies version
```

OUTPUT:

EXERCISE PROGRAM: