Don Bosco Institute of Technology

Kurla (W), Mumbai 400 070

Internet Programming

Name: Parth Pravin Shikhare

Class: TE-IT

Roll No.: 53

Date: 03/10/25

Experiment No.: 10

Title:

Case Study - Express JS

Problem Definition:

To study Express JS and understand its core concepts including Express Router, REST API, Generator, Authentication, Session, and Integration.

Pre-requisites:

1. Basic understanding of JavaScript and Node.js
2. Knowledge of web servers and HTTP protocols
3. Familiarity with JSON and REST architecture
4. Node.js and npm installed on the system

Theory:

Express JS is a fast, unopinionated, and minimalist web framework for Node.js that simplifies building web and API servers. It provides a robust set of features for handling routes, middleware, sessions, and more.

Key Concepts:
- Express JS: Framework for building web applications and APIs in Node.js using simple, flexible routing.
- Express Router: A modular way to handle multiple routes within an application, promoting cleaner code organization.
- REST API: A standardized method of designing web services using HTTP methods like GET, POST, PUT, and DELETE.
- Express Generator: A command-line tool used to quickly scaffold an Express project structure.
- Authentication: A security process for verifying user credentials using libraries like Passport.js or JWT.
- Session: Used to maintain user data across different requests using packages like express-

session.
- Integration: Express easily integrates with databases (MongoDB, MySQL) and frontend frameworks like React, Vue, or Angular.

Procedure:

1. Install Node.js and npm.
2. Install Express globally using `npm install -g express-generator`.
3. Create a new project using `npx express-generator` and install dependencies.
4. Set up Express Router to define routes for different endpoints.
5. Develop RESTful APIs to handle CRUD operations.
6. Implement authentication using JWT or Passport.js.
7. Configure session management using express-session.
8. Integrate Express with a database such as MongoDB or MySQL.
9. Test routes and APIs using Postman or a similar tool.
10. Deploy the final project on a cloud platform like Render, Vercel, or Heroku.

Program:

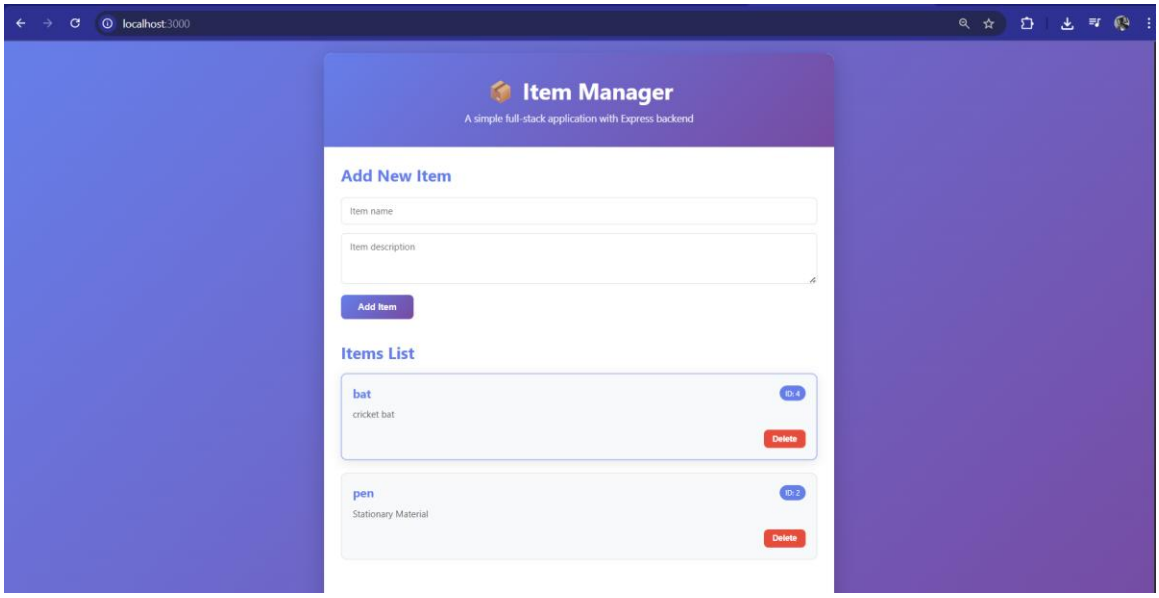
```
const API_URL = 'http://localhost:3000/api';

// Load items when page loads
document.addEventListener('DOMContentLoaded', () => {
  loadItems();
  setupFormHandler();
});

// Load all items from API
async function loadItems() {
  const itemsList = document.getElementById('itemsList');

  try {
    const response = await fetch(`${API_URL}/items`);
    const result = await response.json();

    if (result.success && result.data.length > 0) {
      displayItems(result.data);
    } else {
      itemsList.innerHTML = '<p class="loading">No items found. Add one above!</p>';
    }
  } catch (error) {
    console.error('Error loading items:', error);
    itemsList.innerHTML = '<p class="error">Failed to load items. Make sure the server is
running.</p>';
  }
}

// Display items in the UI
function displayItems(items) {
  const itemsList = document.getElementById('itemsList');
```

```
    itemsList.innerHTML = items.map(item => `
        <div class="item-card" data-id="${item.id}">
            <div class="item-header">
                <span class="item-name">${escapeHtml(item.name)}</span>
                <span class="item-id">ID: ${item.id}</span>
            </div>
            <p class="item-description">${escapeHtml(item.description)}</p>
            <div class="item-actions">
                <button class="btn btn-danger" onclick="deleteItem(${item.id})">Delete</button>
            </div>
        </div>
    `).join('');
}

// Setup form submission handler
function setupFormHandler() {
    const form = document.getElementById('addItemForm');

    form.addEventListener('submit', async (e) => {
        e.preventDefault();

        const name = document.getElementById('itemName').value;
        const description = document.getElementById('itemDescription').value;

        await addItem(name, description);

        // Clear form
        form.reset();
    });
}

// Add new item via API
async function addItem(name, description) {
    try {
        const response = await fetch(`${API_URL}/items`, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ name, description })
        });

        const result = await response.json();

        if (result.success) {
            showMessage('Item added successfully!', 'success');
            loadItems(); // Reload items list
        } else {
            showMessage('Failed to add item', 'error');
```

```
        }
    } catch (error) {
        console.error('Error adding item:', error);
        showMessage('Error adding item. Check console for details.', 'error');
    }
}

// Delete item via API
async function deleteItem(id) {
    if (!confirm('Are you sure you want to delete this item?')) {
        return;
    }

    try {
        const response = await fetch(`${API_URL}/items/${id}`, {
            method: 'DELETE'
        });

        const result = await response.json();

        if (result.success) {
            showMessage('Item deleted successfully!', 'success');
            loadItems(); // Reload items list
        } else {
            showMessage('Failed to delete item', 'error');
        }
    } catch (error) {
        console.error('Error deleting item:', error);
        showMessage('Error deleting item. Check console for details.', 'error');
    }
}

// Show temporary message
function showMessage(message, type) {
    const itemsList = document.getElementById('itemsList');
    const messageDiv = document.createElement('div');
    messageDiv.className = type;
    messageDiv.textContent = message;

    itemsList.insertBefore(messageDiv, itemsList.firstChild);

    setTimeout(() => {
        messageDiv.remove();
    }, 3000);
}

// Escape HTML to prevent XSS
function escapeHtml(text) {
    const div = document.createElement('div');
    div.textContent = text;
```

```
    return div.innerHTML;
}
```

Output:



Results:

Successfully completed a case study on Express JS covering concepts such as Express Router, REST API, Generator, Authentication, Session, and Integration.

References:

1. https://expressjs.com/
2. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs
3. https://www.w3schools.com/nodejs/nodejs_express.asp
4. https://www.npmjs.com/package/express