MERN STACK LAB MANUAL – CYBER SECURITY STUDENTS

EXERCISE 3(II): BLOG READER USING REACT

AIM

To create a React-based Blog Reader that lists posts and shows their details when clicked.

ALGORITHM

1. Create a new folder exercise_3ii and open it in VS Code.
2. Inside it, make a single file named index.html.
3. Use CDN links for React 18 and ReactDOM 18.
4. Write the JavaScript inside a <script> tag to display a list of blog posts and show details when a button is clicked.
5. Open index.html in the browser or run with Live Server.
6. Verify that posts appear and can be read individually.

PROGRAM

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Blog Reader</title>
</head>
<body style="font-family:Arial;padding:20px;max-width:700px;margin:auto">
  <h2>Blog Reader</h2>
  <div id="root"></div>

  <script
src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"></script>

  <script>
    const { useState } = React;
    const POSTS = [
      { id: 1, title: 'Welcome', author: 'Tutor', date: '2025-10-13', body:
'Welcome to the Blog Reader application.' },
      { id: 2, title: 'React Basics', author: 'Tutor', date: '2025-10-12',
body: 'React uses components and state for UI development.' }
    ];

    function App() {
      const [open, setOpen] = useState(null);
      return React.createElement('div', null,
```

```
        POSTS.map(p =>
          React.createElement('div', { key: p.id, style: { borderBottom:
'1px solid #ccc', margin: '10px 0' } },
            React.createElement('h3', null, p.title),
            React.createElement('p', null, p.author + ' • ' + p.date),
            React.createElement('button', { onClick: () => setOpen(p) },
'Read')
          )
        ),
        open && React.createElement('div', { style: { padding: '10px',
border: '1px solid #ccc' } },
          React.createElement('h3', null, open.title),
          React.createElement('p', null, open.body),
          React.createElement('button', { onClick: () => setOpen(null) },
'Back')
        )
      );
    }

ReactDOM.createRoot(document.getElementById('root')).render(React.createEle
ment(App));
  </script>
</body>
</html>
```

RESULT

The program displays blog titles and allows the reader to open and read each post.

EXERCISE 3(III): REACT ROUTER

AIM

To demonstrate client-side navigation using React Router.

ALGORITHM

1. Create folder exercise_3iii → open in VS Code.
2. Make index.html.
3. Add React and React Router DOM CDN links.
4. Define Home, About, and Contact components.
5. Configure <Routes> and <Link> for navigation.
6. Open in browser and navigate between pages.

PROGRAM

```
<!doctype html>
<html>
<head><meta charset="utf-8"/><title>React Router Demo</title></head>
<body style="font-family:Arial;padding:20px">
<h2>React Router Example</h2>
<div id="root"></div>

<script src="https://unpkg.com/react@18/umd/react.development.js"></script>
<script src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"></script>
<script src="https://unpkg.com/react-router-dom@6/umd/react-router-
dom.development.js"></script>

<script>
const { HashRouter, Routes, Route, Link } = ReactRouterDOM;
const e = React.createElement;
function Home(){return e('h3',null,'Home Page');}
function About(){return e('h3',null,'About Page');}
function Contact(){return e('h3',null,'Contact Page');}
function App(){
  return e(HashRouter,null,
    e('div',null,
      e(Link,{to:'/'},'Home'),' | ',
      e(Link,{to:'/about'},'About'),' | ',
      e(Link,{to:'/contact'},'Contact')),
    e(Routes,null,
      e(Route,{path:'/',element:e(Home)}),
      e(Route,{path:'/about',element:e(About)}),
      e(Route,{path:'/contact',element:e(Contact)})));
}
ReactDOM.createRoot(document.getElementById('root')).render(e(App));
</script>
</body>
</html>
```

RESULT

Links navigate between Home, About, and Contact without page reload.

EXERCISE 4(I): BOOKSTORE API USING NODE.JS

AIM

To create a RESTful Bookstore API using Express.js for handling GET and POST requests.

ALGORITHM

1. Open PowerShell or VS Code terminal.
2. Create a folder exercise_4i and navigate into it.
3. Initialize a Node.js project using npm init -y.

4. Install Express using npm install express.
5. Create server.js and write code for handling API requests.
6. Run the server using node server.js.
7. Test API using Postman or browser.

PROGRAM

```
const express = require('express');
const app = express();
app.use(express.json());

let books = [{ id: 1, title: 'Hindustan dairies', author: 'cyber students'
}];

app.get('/books', (req, res) => res.json(books));

app.post('/books', (req, res) => {
  const { title, author } = req.body;
  if (!title || !author) return res.status(400).json({ message: 'Title and
author required' });
  const newBook = { id: books.length + 1, title, author };
  books.push(newBook);
  res.status(201).json(newBook);
});

app.listen(3000, () => console.log('Bookstore API running on
http://localhost:3000'));
```

RESULT

The API successfully allows users to view and add books using GET and POST requests.

EXERCISE 4(II): CONNECTING NODE.JS TO MONGODB USING MONGOOSE

AIM

To connect a Node.js application to MongoDB using Mongoose and perform basic CRUD operations.

ALGORITHM

1. Open PowerShell or VS Code terminal.
2. Create a folder exercise_4ii and navigate into it.
3. Initialize a Node.js project using npm init -y.
4. Install dependencies using npm install express mongoose.

5. Create a file server.js.
6. Write code to connect Node.js with MongoDB.
7. Start MongoDB service from Windows or using mongod.
8. Test insert and fetch operations using Postman.

PROGRAM

```
const express = require('express');
const mongoose = require('mongoose');
const app = express();
app.use(express.json());

const MONGO_URI = 'mongodb://127.0.0.1:27017/lab_demo_db';
mongoose.connect(MONGO_URI)
  .then(() => console.log('Connected to MongoDB'))
  .catch(err => console.error('MongoDB connection error:', err.message));

const userSchema = new mongoose.Schema({
  name: String,
  age: Number,
  createdAt: { type: Date, default: Date.now }
});
const User = mongoose.model('User', userSchema);

app.post('/users', async (req, res) => {
  const { name, age } = req.body;
  if (!name) return res.status(400).json({ message: 'Name required' });
  const u = new User({ name, age });
  await u.save();
  res.status(201).json(u);
});

app.get('/users', async (req, res) => {
  const users = await User.find().sort({ createdAt: -1 });
  res.json(users);
});

app.listen(3000, () => console.log('MongoDB App running on
http://localhost:3000'));
```

RESULT

Node.js connects to MongoDB successfully, and users can insert and retrieve data using Postman or MongoDB Compass.

EXERCISE 4(III): JWT AUTHENTICATION USING NODE.JS

AIM

To implement login that returns JWT token and protect routes using middleware in Node.js.

ALGORITHM

1. Create a folder exercise_4iii.
2. Run npm init -y.
3. Install dependencies using npm install express jsonwebtoken.
4. Create a file server.js.
5. Write authentication and middleware logic for JWT token creation and validation.
6. Run the server using node server.js.
7. Test login and protected routes in Postman.

PROGRAM

```
const express = require('express');
const jwt = require('jsonwebtoken');
const app = express();
app.use(express.json());

const SECRET = 'demo_secret';
const USER = { username: 'admin', password: '1234' };

app.post('/login', (req, res) => {
  const { username, password } = req.body;
  if (username === USER.username && password === USER.password) {
    const token = jwt.sign({ username }, SECRET, { expiresIn: '1h' });
    return res.json({ token });
  }
  res.status(401).json({ message: 'Invalid credentials' });
});

function auth(req, res, next) {
  const h = req.headers.authorization;
  if (!h || !h.startsWith('Bearer ')) return res.status(401).json({
message: 'Missing token' });
  const token = h.split(' ')[1];
  try {
    req.user = jwt.verify(token, SECRET);
    next();
  } catch (e) {
    return res.status(401).json({ message: 'Invalid token' });
  }
}

app.get('/protected', auth, (req, res) => {
  res.json({ message: 'Welcome ' + req.user.username });
});

app.listen(3000, () => console.log('JWT Auth running on
http://localhost:3000'));
```

RESULT

JWT authentication implemented successfully, allowing secure access to protected routes.

EXERCISE 5(I): REACT + EXPRESS INTEGRATION

AIM

To demonstrate integration between a React frontend and an Express backend by serving a simple React page from the Express server and fetching data from an API endpoint.

ALGORITHM

1. Create a project folder named exercise_5i and open it in VS Code.
2. Initialize the Node project and install Express using npm init -y and npm install express.
3. Create two files: server.js (Express server) and index.html (frontend using React via CDN).
4. In server.js create an API route /api/message that returns JSON and serve index.html at /.
5. In index.html use React (CDN) to fetch /api/message and display the response.
6. Start the server with node server.js and open http://localhost:3000 in the browser to verify the message appears.

COMMANDS (Windows PowerShell)

```
mkdir exercise_5i
cd exercise_5i
npm init -y
npm install express
code server.js
code index.html
# after pasting code, run:
node server.js
# open http://localhost:3000 in browser
```

PROGRAM — server.js

```
const express = require('express');
const path = require('path');
const app = express();
const PORT = 3000;

app.get('/api/message', (req, res) => {
  res.json({ message: 'Hello from Express' });
});

app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'index.html'));
});
```

```
app.listen(PORT, () => console.log('React + Express running on
http://localhost:' + PORT));
```

PROGRAM — index.html

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>React + Express Integration</title>
</head>
<body style="font-family:Arial;padding:20px">
  <div id="root"></div>

  <script
src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"></script>

  <script>
    const e = React.createElement;
    function App(){
      const [msg, setMsg] = React.useState('Loading...');
      React.useEffect(()=>{
        fetch('/api/message')
          .then(r=>r.json())
          .then(j=>setMsg(j.message))
          .catch(()=>setMsg('Error fetching'));
      },[]);
      return e('div', null, e('h3', null, 'React + Express Integration'),
e('p', null, msg));
    }
    ReactDOM.createRoot(document.getElementById('root')).render(e(App));
  </script>
</body>
</html>
```

RESULT

The React frontend displays the message received from the Express backend.

EXERCISE 5(II): FULL STACK TASK MANAGER

AIM

To create a full-stack Task Manager application where the Express backend provides API endpoints and an HTML+React frontend calls these endpoints. The project is contained in a single folder for simplicity.

ALGORITHM

1. Create folder exercise_5ii and open in VS Code.
2. Initialize Node project and install Express: npm init -y, npm install express.
3. Create server.js (API and static file serving) and index.html (React frontend via CDN).
4. Implement endpoints: GET /api/tasks, POST /api/tasks, POST /api/tasks/:id/toggle, DELETE /api/tasks/:id.
5. Frontend fetches /api/tasks on load and uses the other endpoints to add, toggle, and delete tasks.
6. Start server with node server.js and use the UI to test the features.

COMMANDS (Windows PowerShell)

```
mkdir exercise_5ii
cd exercise_5ii
npm init -y
npm install express
code server.js
code index.html
node server.js
# open http://localhost:3000 in browser
```

PROGRAM — server.js

```
const express = require('express');
const path = require('path');
const app = express();
app.use(express.json());
const PORT = 3000;

let tasks = [{ id: 1, title: 'Example task', done: false }];

app.get('/api/tasks', (req, res) => res.json(tasks));

app.post('/api/tasks', (req, res) => {
  const title = req.body.title || 'Untitled';
  const t = { id: tasks.length + 1, title, done: false };
  tasks.push(t);
  res.status(201).json(t);
});

app.post('/api/tasks/:id/toggle', (req, res) => {
  const id = Number(req.params.id);
  const t = tasks.find(x => x.id === id);
  if (!t) return res.status(404).json({ error: 'Task not found' });
  t.done = !t.done;
  res.json(t);
});

app.delete('/api/tasks/:id', (req, res) => {
  const id = Number(req.params.id);
  tasks = tasks.filter(x => x.id !== id);
  res.json({ ok: true });
});
```

```
app.get('/', (req, res) => res.sendFile(path.join(__dirname,
'index.html')));

app.listen(PORT, () => console.log('Task Manager running on
http://localhost:' + PORT));
```

PROGRAM — index.html

```
<!doctype html>
<html>
<head>
<meta charset="utf-8"/>
<title>Task Manager</title>
<style>
body{font-family:Arial;padding:20px;max-width:700px;margin:auto}
input{padding:8px;width:60%}
button{margin-left:8px;padding:8px}
li{margin:6px 0}
</style>
</head>
<body>
<h1>Task Manager</h1>
<div id="root"></div>
<script src="https://unpkg.com/react@18/umd/react.development.js"></script>
<script src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"></script>
<script>
const e=React.createElement;
function App(){
  const[tasks,setTasks]=React.useState([]);
  const[title,setTitle]=React.useState('');
  const load=()=>fetch('/api/tasks').then(r=>r.json()).then(setTasks);
  React.useEffect(load,[]);
  const add=async()=>{
    if(!title)return alert('Enter title');
    await fetch('/api/tasks',{method:'POST',headers:{'Content-
Type':'application/json'},body:JSON.stringify({title})});
    setTitle('');load();
  };
  const toggle=async(id)=>{await
fetch('/api/tasks/'+id+'/toggle',{method:'POST'});load();};
  const remove=async(id)=>{await
fetch('/api/tasks/'+id,{method:'DELETE'});load();};
  return e('div',null,
    e('div',null,

e('input',{value:title,onChange:ev=>setTitle(ev.target.value),placeholder:'
New task title'}),
      e('button',{onClick:add},'Add Task')
    ),
    e('ul',null,
      tasks.map(t=>e('li',{key:t.id},

e('input',{type:'checkbox',checked:t.done,onChange:()=>toggle(t.id)}),
        ' '+t.title,
        e('button',{onClick:()=>remove(t.id)},'Delete')
      ))
    )
  );
```

```
}
ReactDOM.createRoot(document.getElementById('root')).render(e(App));
</script>
</body>
</html>
```

RESULT

The Task Manager UI loads tasks, allows adding new tasks, toggling completion, and deleting tasks via the backend API.

EXERCISE 5(III): DEPLOYMENT DEMO (GITHUB + RENDER.COM)

AIM

To prepare and deploy a simple Express application to Render.com using a GitHub repository.

ALGORITHM

1. Create a folder exercise_5iii and initialize a Node project with npm init -y.
2. Install Express: npm install express.
3. Create server.js, index.html, and ensure package.json has the start script: "start": "node server.js".
4. Test locally with node server.js.
5. Initialize Git, commit files, and push to a new GitHub repository.
6. Create a Render account and connect GitHub.
7. On Render, create a new Web Service, select the repository, set build command (optional) to npm install and start command to npm start.
8. Deploy and test the live URL.

COMMANDS (Windows PowerShell) — Local and GitHub

```
mkdir exercise_5iii
cd exercise_5iii
npm init -y
npm install express
code server.js
code index.html
# add code files, then:
node server.js   # test locally
# Initialize git and push to GitHub:
git init
git add .
git commit -m "Initial commit - deployment demo"
```

```
# create a repo on GitHub (via website), then:
git remote add origin https://github.com/<your-username>/<repo-name>.git
git branch -M main
git push -u origin main
```

RENDER.COM SETUP (step-by-step)

1. Sign up / log in to https://render.com using your GitHub account.
2. Click New → Web Service.
3. Connect your GitHub repository by authorizing Render.
4. Select the repository you pushed earlier.
5. For Environment, keep Node.
6. Set Build Command to npm install (Render will run this before starting).
7. Set Start Command to npm start.
8. Click Create Web Service and wait for deployment.
9. Once deployed, Render provides a public URL — open it and test the app and /api/message endpoint.

PROGRAM — server.js

```
const express = require('express');
const path = require('path');
const app = express();
const PORT = process.env.PORT || 3000;

app.get('/', (req, res) => res.sendFile(path.join(__dirname,
'index.html')));
app.get('/api/message', (req, res) => res.json({ message: 'Deployed
successfully' }));

app.listen(PORT, () => console.log('App running on port ' + PORT));
```

PROGRAM — index.html

```
<!doctype html>
<html>
<head><meta charset="utf-8"/><title>Deployment Demo</title></head>
<body style="font-family:Arial;text-align:center;margin-top:80px">
<h1>Deployment Demo</h1>
<button id="btn">Check API</button>
<pre id="out"></pre>
<script>
document.getElementById('btn').onclick = async () => {
  const res = await fetch('/api/message');
  const j = await res.json();
  document.getElementById('out').textContent = JSON.stringify(j, null, 2);
};
</script>
</body>
</html>
```

RESULT

After deploying, the Render URL should load the page and the Check API button should fetch JSON from /api/message.

EXERCISE 6: MINI PROJECT — TASK MANAGER DEPLOYMENT

AIM

To deploy the Full Stack Task Manager application (from Exercise 5(ii)) to Render.com via GitHub and verify all endpoints and frontend work on the live site.

ALGORITHM

1. Ensure the Task Manager project (server.js, index.html, package.json) is working locally on port 3000.
2. Make sure package.json includes a start script: "start": "node server.js".
3. Initialize Git, commit, and push the project to a GitHub repository.
4. Sign in to Render.com and create a new Web Service connected to that GitHub repo.
5. Configure Build Command npm install and Start Command npm start (or leave blank if Render auto-detects).
6. Deploy and open the Render URL.
7. Test all API endpoints and UI features on the live deployment.

COMMANDS (Windows PowerShell) — Git & GitHub

```
# from project folder (exercise_5ii or cloned folder for deployment)
git init
git add .
git commit -m "Task Manager ready for deploy"
# create new GitHub repo via website, then:
git remote add origin https://github.com/<your-username>/<task-manager-repo>.git
git branch -M main
git push -u origin main
```

RENDER.COM SETUP (step-by-step)

1. Log in to https://render.com and click New -> Web Service.
2. Authorize and select the GitHub repository containing the Task Manager.
3. Fill in:
    - Name: task-manager (or choose your own)
    - Branch: main
    - Build Command: npm install
    - Start Command: npm start

4. Click Create Web Service. Render will clone the repo, install dependencies and start the service.
5. Monitor the deploy logs — if the build succeeds you will see a public URL.
6. Open the URL and test the UI and API endpoints (GET /api/tasks, POST /api/tasks, etc.).

RESULT

The Task Manager should run on Render with a live URL. All CRUD operations on tasks should work as they did locally.