SE3040 – Application Frameworks
BSc (Hons) in Information Technology
Software Engineering Specialization
3rd Year
Faculty of Computing
SLIIT
2025 - Practical
Lab 04

Lab session 4 – Express.js REST API.

Objective: Teach set of basic concepts in REST API using Express.js.

Getting starting with Express.js

1. Create a new directory for your new Express.js project and run the following command inside of that folder.

```
npm install express body-parser
```

This will install Express.js and Body Parser, which we will use to parse incoming requests.

2. After, all you have to do is create a new file called, index.js and add the following code and save it.

```
const express = require('express');
const bodyParser = require('body-parser');

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

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

app.get('/', (req, res) => {
  res.send('Hello World!');
});

app.get('/api/user', (req, res) => {
  const users = [
    { id: 1, name: 'John Doe' },
    { id: 2, name: 'Jane Doe' },
  ];
  res.send(users);
});
```

Creates an Express app—your web server.

Parses data from forms (e.g., name=Jane in a POST request

Parses JSON data sent in the request body (e.g., {"name": "Jane"}).

A hardcoded array of user objects (in a real app, this might come from a database).res.send(users): Sends the array as a JSON response.

```
app.post('/api/user', (req, res) => {
  const { name } = req.body;
  const user = { id: 3, name };
  res.send(user);
});

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`);
});
```

Gets the data sent by the client (thanks to body-parser parsing it).

Extracts the name field from the request body

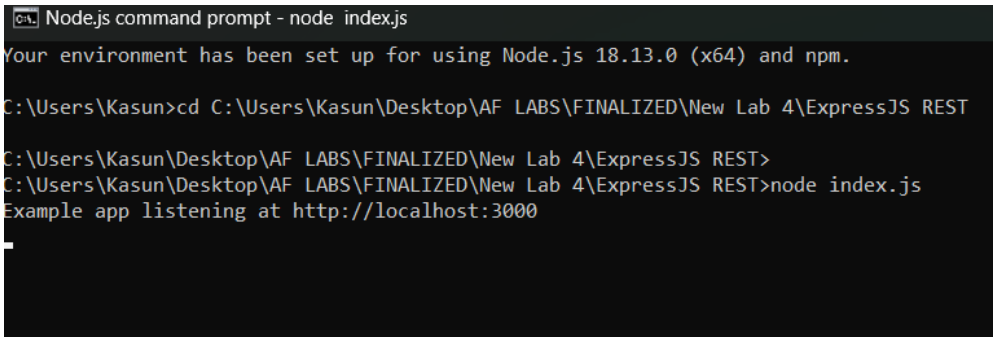
Extracts the name field from the request body

The first route is a simple GET request that returns a "Hello World!" message when you visit the homepage.
The second route is a GET request that returns an array of users when you visit /api/user.
The third route is a POST request that expects a JSON object with a name property in the request body, and returns a new user object with an id of 3 and the provided name.

3. Finally run the file with node command.

```
node index.js
```

4. That will display output like this.



5. Now you have started the Express.js application and if you visit that URL via your browser, that will show a “Hello World” text.

6. We can test our API using Postman.

7. Postman is an API Platform for developers to design, build, test and iterate their APIs and you can download it from following URL and install in your local system.

   https://www.postman.com/downloads/

8. After install Postman, you can make API requests as follows and, also, you can see the responses of particular request in there too.

9. Modify the code as follows,

```
const express = require('express');
const bodyParser = require('body-parser');

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

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

let todos = [  mple array acting as a "database" (in memory)
  { id: 1, text: 'Buy groceries', done: false },
  { id: 2, text: 'Do laundry', done: true },
];

app.get('/api/todos', (req, res) => {
  res.send(todos);      Returns the full list of to-dos as JSON
});

app.get('/api/todos/:id', (req, res) => {
  const id = Number(req.params.id);  Gets the id from the URL. and convert id into number
  const todo = todos.find(todo => todo.id === id);  Searches the array for a to-do with
                                                    that id.
  if (todo) {
    res.send(todo);     If found: Sends the to-do as JSON.
                        If not: Sends a 404 error.
  } else {
    res.status(404).send('Todo not found');
  }
});

app.post('/api/todos', (req, res) => {
  const { text, done } = req.body;   req.body: Gets text and done from the request body
  const id = todos.length + 1;       id: Generates a new ID based on the array length + 1.
  const todo = { id, text, done };   todos.push(): Adds the new to-do to the list.
                                     Response: Sends the new to-do back.
  todos.push(todo);
  res.send(todo);
});

app.put('/api/todos/:id', (req, res) => {
  const id = Number(req.params.id);
  const { text, done } = req.body;
  const todo = todos.find(todo => todo.id === id);
```

req.params.id: Gets the ID to update.
req.body: Gets new text and/or done values.
||: Keeps the old value if the new one isn't provided (e.g., if text is missing, use the current todo.text).
Response: Sends the updated to-do or a 404 if not found.

```
  if (todo) {
    todo.text = text || todo.text;
    todo.done = done || todo.done;
    res.send(todo);
  } else {
    res.status(404).send('Todo not found');
  }
});
```

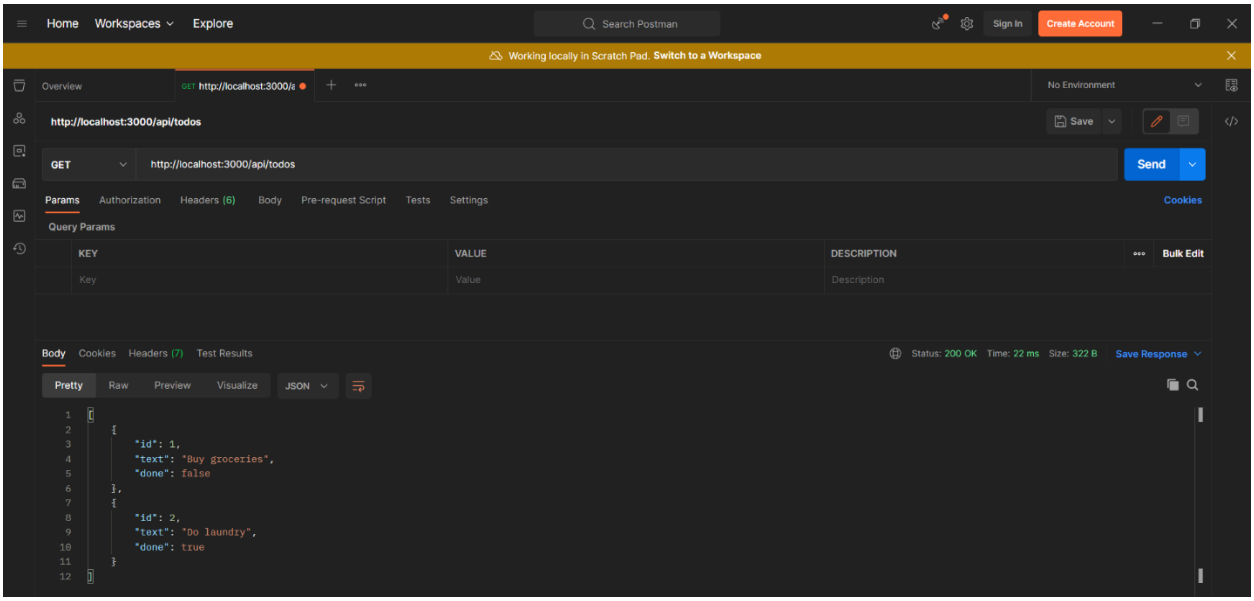
todos.filter(): Removes the to-do with the matching ID.Response: Confirms deletion with a message.

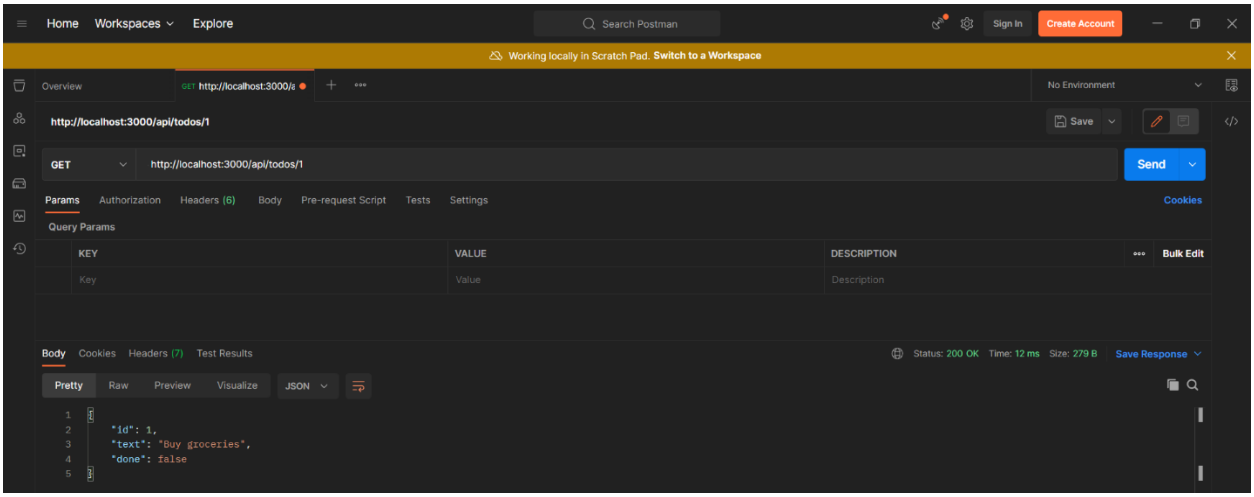
```
app.delete('/api/todos/:id', (req, res) => {
  const id = Number(req.params.id);
  todos = todos.filter(todo => todo.id !== id);
  res.send(`Todo with id ${id} has been deleted`);
});

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`);
});
```

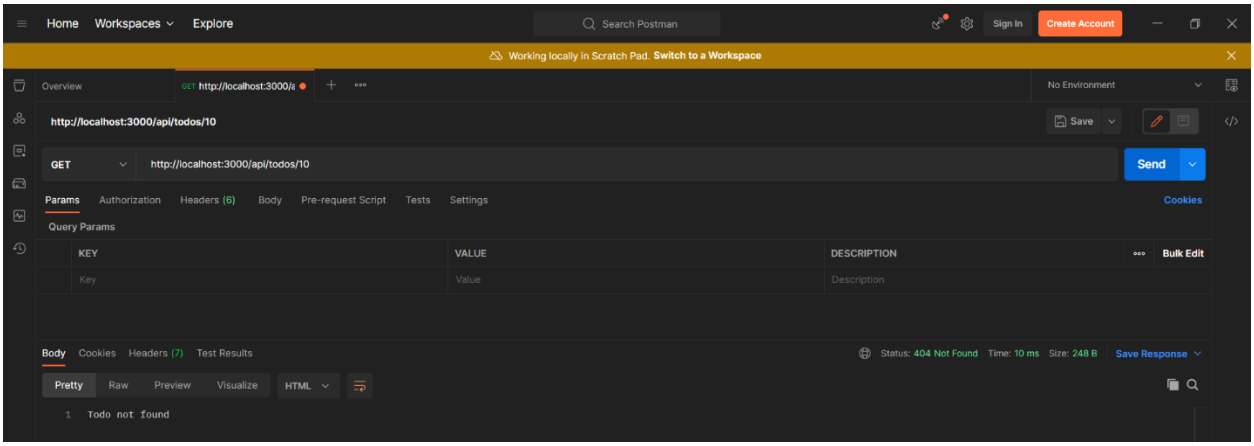
10. Now run the application again and you can see how the API working via Postman.

11. In here you can practicing GET, POST, PUT, DELETE requests.

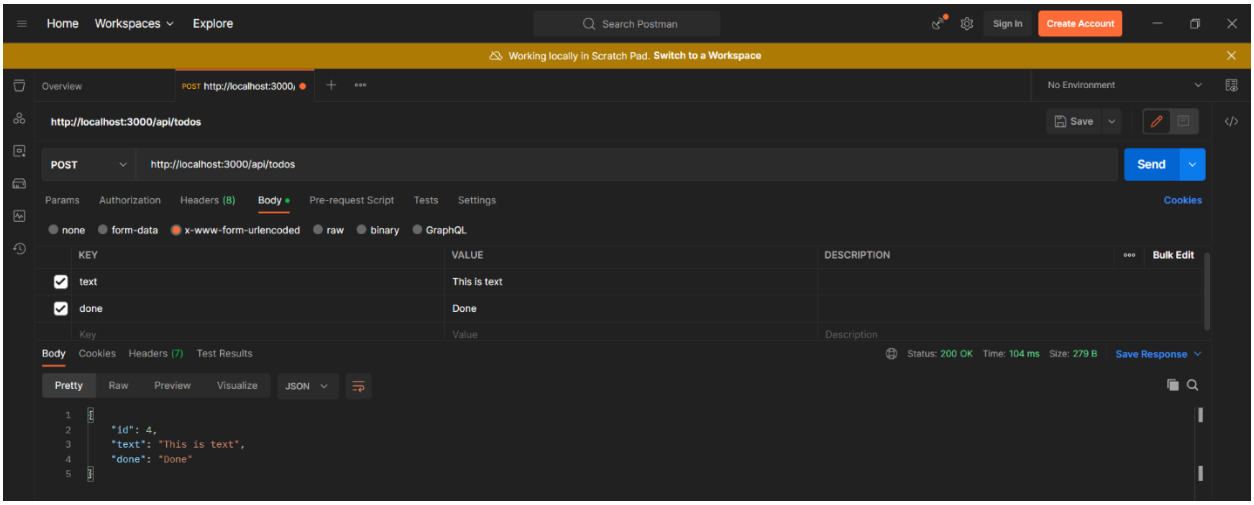
Some screenshots of GET are as follows,

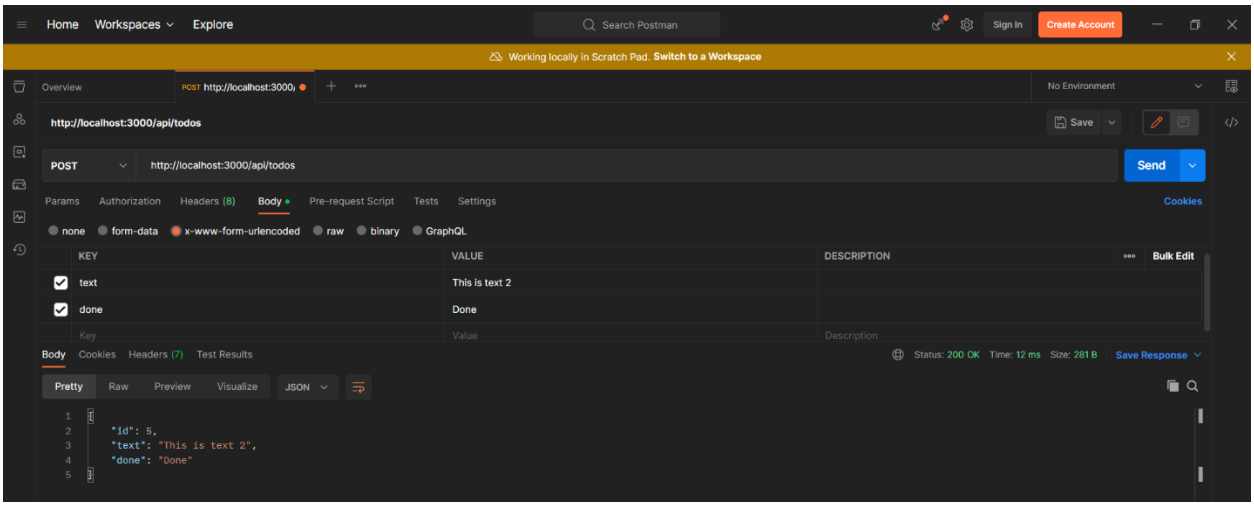




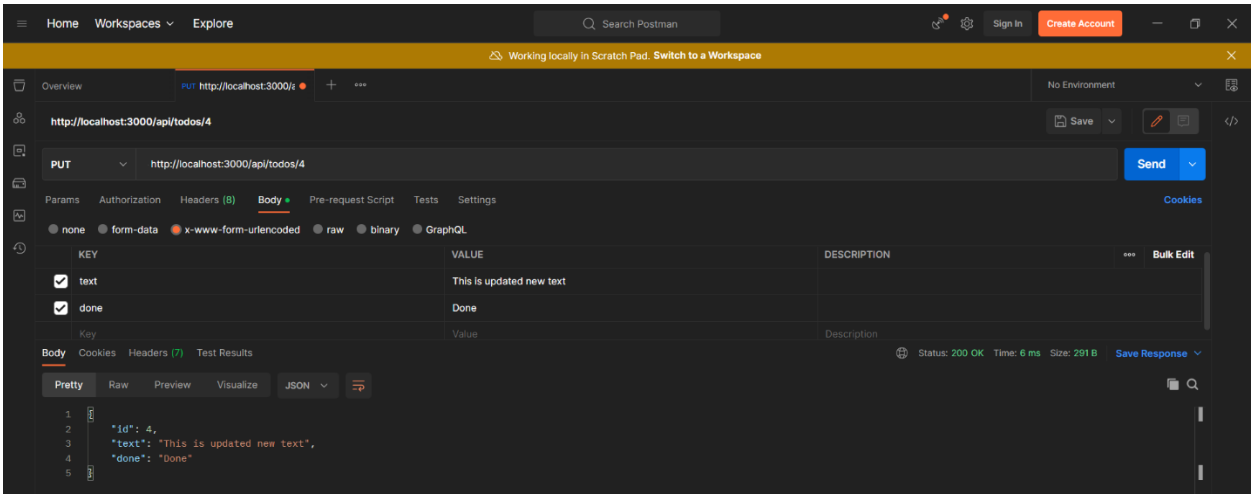


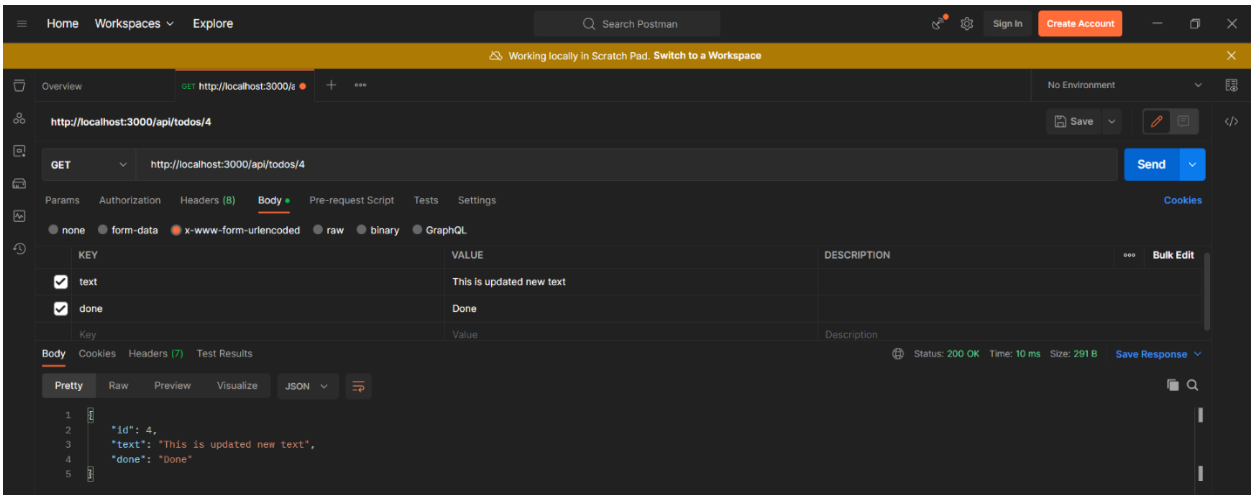
Some Screenshots of POST as follows,





Some screenshots of PUT as follows,





Some screenshots of DELETE as follows,