Express and Postman Basics

Why Use Express?

- A library that simplifies backend development in Node.js.
- Makes creating and managing routes (URLs) easy.
- Handles tasks like authentication, data validation, and responses efficiently.

Why Use Postman?

- A professional tool for testing APIs (backend routes).
- Allows sending requests like **GET**, **POST**, **PUT**, and **DELETE**.

Steps to Set Up Express

1. **Initialize Project:**
   - Create a folder for your project.
   - Run `npm init` to set up your Node.js project.
2. **Install Express:**
   - Run `npm install express` to add Express to your project.
3. **Set Up a Basic Server:**
   - Create an `index.js` file.
   - Import Express:

```javascript
import express from 'express';
```

   - Create an app:

```javascript
const app = express();
```

   - Define a port and listen:

```javascript
const port = 3000; app.listen(port, () => console.log(`Server running on port ${port}`));
```

4. **Add Routes:**
   - Example of a **GET** route:

```javascript
app.get('/', (req, res) => { res.send("Hello from Express!"); });
```

5. **Run the Server:**
   - Start the server with `node index.js` or use `nodemon` for auto-restart.

CRUD Operations (Create, Read, Update, Delete)

1. Create (POST)

- Route: `/teas`
- Example Code:

```javascript
const teas = []; let nextId = 1; app.post('/teas', (req, res) => { const { name, price } = req.body; const newTea = { id: nextId++, name, price }; teas.push(newTea); res.status(201).send(newTea); });
```

2. Read All (GET)

- Route: `/teas`
- Example Code:

```javascript
app.get('/teas', (req, res) => { res.status(200).send(teas); });
```

3. Read One (GET with ID)

- Route: `/teas/:id`
- Example Code:

```javascript
app.get('/teas/:id', (req, res) => { const tea = teas.find(t => t.id === parseInt(req.params.id)); if (!tea) return res.status(404).send("Tea not found"); res.status(200).send(tea); });
```

4. Update (PUT)

- Route: `/teas/:id`
- Example Code:

```javascript
app.put('/teas/:id', (req, res) => { const tea = teas.find(t => t.id === parseInt(req.params.id)); if (!tea) return res.status(404).send("Tea not found"); const { name, price } = req.body; tea.name = name; tea.price = price; res.status(200).send(tea); });
```

5. Delete (DELETE)

- Route: `/teas/:id`
- Example Code:

```javascript
app.delete('/teas/:id', (req, res) => { const index = teas.findIndex(t => t.id === parseInt(req.params.id)); if (index === -1) return res.status(404).send("Tea not found"); teas.splice(index, 1); res.status(204).send(); });
```

Postman Testing

1. **Set Up Request Types:**
   - **POST**: Add data to the backend.
   - **GET**: Fetch data from the backend.
   - **PUT**: Update existing data.
   - **DELETE**: Remove data.
2. **Testing Example:**
   - Send a **POST** request to `/teas` with:

     ```json
     { "name": "Ginger Tea", "price": 100 }
     ```

   - Verify with a **GET** request to `/teas` to see all teas.
3. **Working with Variables:**
   - Use variables in Postman to make testing reusable and organized.
4. **Handling Errors:**
   - Test edge cases like invalid IDs and missing data.

Tools and Best Practices

- **Use** `nodemon`**:**
  - Automatically restarts the server when files change.
  - Install as a dev dependency:

```bash
npm install nodemon -D
```

  - Add script in `package.json`:

```json
"scripts": { "start": "node index.js", "dev": "nodemon index.js" }
```

- **Structure Code for Scalability:**
  - Separate routes, models, and controllers in different files for better organization.
- **Documentation:**
  - Maintain API documentation using tools like Postman or Swagger.