Building RESTful APIs with Express.js involves defining routes and methods that adhere to the principles of REST. Here's a guide to get you started:

---

1. Setting Up the Project

Follow these initial steps:

Create a project directory:

```
mkdir express-rest-api
cd express-rest-api
npm init -y
```

1.

Install dependencies:

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

2.

Create a file named `app.js`:

```
touch app.js
```

3.

---

2. Basic Structure of a REST API

Example API: Managing a collection of "Users"

Open `app.js` and set up your Express application:

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

// Middleware for parsing JSON
app.use(express.json());

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

1.

Define your endpoints for CRUD operations:

```
 let users = []; // In-memory data store

// Create a user (POST /users)
app.post('/users', (req, res) => {
   const user = req.body;
   users.push(user);
   res.status(201).json({ message: 'User created', user });
});

// Get all users (GET /users)
app.get('/users', (req, res) => {
   res.json(users);
});

// Get a single user by ID (GET /users/:id)
app.get('/users/:id', (req, res) => {
   const user = users.find(u => u.id === parseInt(req.params.id));
   if (!user) {
      return res.status(404).json({ message: 'User not found' });
   }
   res.json(user);
});

// Update a user by ID (PUT /users/:id)
app.put('/users/:id', (req, res) => {
   const user = users.find(u => u.id === parseInt(req.params.id));
   if (!user) {
      return res.status(404).json({ message: 'User not found' });
   }
   Object.assign(user, req.body);
   res.json({ message: 'User updated', user });
});

// Delete a user by ID (DELETE /users/:id)
app.delete('/users/:id', (req, res) => {
   const index = users.findIndex(u => u.id === parseInt(req.params.id));
   if (index === -1) {
      return res.status(404).json({ message: 'User not found' });
   }
   users.splice(index, 1);
   res.json({ message: 'User deleted' });
});
```

2.

---

3. Testing Your API

- Use **Postman** or **cURL** to interact with your API.

Example Requests:

Create a user:
```
curl -X POST http://localhost:3000/users -H "Content-Type: application/json" -d '{"id": 1, "name": "Alice"}'
```

1.

Get all users:
```
curl -X GET http://localhost:3000/users
```

2.

Get a user by ID:
```
curl -X GET http://localhost:3000/users/1
```

3.

Update a user:
```
curl -X PUT http://localhost:3000/users/1 -H "Content-Type: application/json" -d '{"name": "Alice Smith"}'
```

4.

Delete a user:
```
curl -X DELETE http://localhost:3000/users/1
```

5.

---

4. Add Middleware

Example: Error Handling

Add error-handling middleware to catch unexpected issues:

```
// Global error handler
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ message: 'Internal Server Error' });
});
```

---

5. Use a Database

Replace the in-memory data store with a database like MongoDB using Mongoose.

Install Mongoose:

```
npm install mongoose
```

Connect to MongoDB:

```
const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost:27017/rest_api', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

const UserSchema = new mongoose.Schema({
  id: Number,
  name: String,
});

const User = mongoose.model('User', UserSchema);
```

Update CRUD Operations:

Replace in-memory operations with database queries:

```
// Create a user
app.post('/users', async (req, res) => {
  const user = new User(req.body);
  await user.save();
  res.status(201).json({ message: 'User created', user });
});

// Get all users
app.get('/users', async (req, res) => {
  const users = await User.find();
  res.json(users);
});

// Other operations follow a similar pattern
```

---

6. Organize Your Code

For scalability, organize your project:

```
express-rest-api/
├── models/
│   └── user.js      # User schema
├── routes/
│   └── users.js     # User routes
├── app.js           # Main app entry
├── package.json
```

Example: `routes/users.js`

```
const express = require('express');
const router = express.Router();
const User = require('../models/user');

// Define routes here

module.exports = router;
```

This structure ensures your API is maintainable and scalable. Let me know if you need further customization or advanced features like authentication!