**1. Express.js**

Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for building web and mobile applications.

**Example: Basic Express.js App**

const express = require('express');

const app = express();

const port = 3000;

// Basic route

app.get('/', (req, res) => {

res.send('Hello, World!');

});

app.listen(port, () => {

console.log(`Server is running at http://localhost:${port}`);

});

In this example, we create an Express application that responds with "Hello, World!" when accessed at the root URL.

**2. Middleware**

Middleware functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application’s request-response cycle. They can perform tasks such as modifying the request or response objects, ending the request-response cycle, or calling the next middleware function.

**Example: Logging Middleware**

const express = require('express');

const app = express();

const port = 3000;

// Logging middleware

app.use((req, res, next) => {

console.log(`${req.method} ${req.url}`);

next(); // Pass control to the next middleware function

});

app.get('/', (req, res) => {

res.send('Hello, World!');

});

app.listen(port, () => {

console.log(`Server is running at http://localhost:${port}`);

});

In this example, we have a logging middleware that logs the HTTP method and URL of each request. The next() function passes control to the next middleware.

**3. Modules**

In Express.js, you can organize your application into modules to keep your code clean and maintainable. Modules can be used to group related routes and functionality.

**Example: Modular Routing**

Create a file named routes/users.js:

const express = require('express');

const router = express.Router();

// Route for getting all users

router.get('/', (req, res) => {

res.send('List of users');

});

// Route for creating a user

router.post('/', (req, res) => {

res.send('User created');

});

module.exports = router;

In your main application file (app.js):

const express = require('express');

const app = express();

const port = 3000;

const userRoutes = require('./routes/users');

app.use('/users', userRoutes);

app.listen(port, () => {

console.log(`Server is running at http://localhost:${port}`);

});

In this example, routes related to users are separated into a module (routes/users.js). The main application file (app.js) imports and uses these routes.

**4. async.each**

async.each is part of the async library, which provides utility functions for asynchronous programming. async.each is used to iterate over a list of items and execute an asynchronous function on each item.

**Example: Using async.each**

First, install the async library:

npm install async

Then, use it in your code:

const async = require('async');

const items = [1, 2, 3, 4, 5];

async.each(items, (item, callback) => {

// Simulate an asynchronous operation

setTimeout(() => {

console.log(`Processing item ${item}`);

callback(); // Call callback to indicate completion

}, 1000);

}, (err) => {

if (err) {

console.error('An error occurred:', err);

} else {

console.log('All items have been processed');

}

});

In this example, async.each processes each item in the items array with a simulated asynchronous operation. The final callback is called when all items are processed or if an error occurs.

**5. Express Routing**

Routing in Express.js is used to define the endpoints of your application and the corresponding request handlers.

**Example: Express Routing**

const express = require('express');

const app = express();

const port = 3000;

// Route for the root URL

app.get('/', (req, res) => {

res.send('Welcome to the root page!');

});

// Route for the /about URL

app.get('/about', (req, res) => {

res.send('This is the about page.');

});

// Route with a parameter

app.get('/user/:id', (req, res) => {

const userId = req.params.id;

res.send(`User ID is ${userId}`);

});

app.listen(port, () => {

console.log(`Server is running at http://localhost:${port}`);

});

In this example:

* / responds with a welcome message.
* /about responds with an about page message.
* /user/:id uses a route parameter (:id) to capture and display a dynamic value.

These examples should help you get started with Express.js, middleware, modules, async.each, and routing!

**1. Express.js**

Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for building web and mobile applications.

**Example: Basic Express.js App**

const express = require('express');

const app = express();

const port = 3000;

// Basic route

app.get('/', (req, res) => {

res.send('Hello, World!');

});

app.listen(port, () => {

console.log(`Server is running at http://localhost:${port}`);

});

In this example, we create an Express application that responds with "Hello, World!" when accessed at the root URL.

**2. Middleware**

Middleware functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application’s request-response cycle. They can perform tasks such as modifying the request or response objects, ending the request-response cycle, or calling the next middleware function.

**Example: Logging Middleware**

javascript

Copy code

const express = require('express');

const app = express();

const port = 3000;

// Logging middleware

app.use((req, res, next) => {

console.log(`${req.method} ${req.url}`);

next(); // Pass control to the next middleware function

});

app.get('/', (req, res) => {

res.send('Hello, World!');

});

app.listen(port, () => {

console.log(`Server is running at http://localhost:${port}`);

});

In this example, we have a logging middleware that logs the HTTP method and URL of each request. The next() function passes control to the next middleware.

**3. Modules**

In Express.js, you can organize your application into modules to keep your code clean and maintainable. Modules can be used to group related routes and functionality.

**Example: Modular Routing**

Create a file named routes/users.js:

const express = require('express');

const router = express.Router();

// Route for getting all users

router.get('/', (req, res) => {

res.send('List of users');

});

// Route for creating a user

router.post('/', (req, res) => {

res.send('User created');

});

module.exports = router;

In your main application file (app.js):

javascript

Copy code

const express = require('express');

const app = express();

const port = 3000;

const userRoutes = require('./routes/users');

app.use('/users', userRoutes);

app.listen(port, () => {

console.log(`Server is running at http://localhost:${port}`);

});

In this example, routes related to users are separated into a module (routes/users.js). The main application file (app.js) imports and uses these routes.

**4. async.each**

async.each is part of the async library, which provides utility functions for asynchronous programming. async.each is used to iterate over a list of items and execute an asynchronous function on each item.

**Example: Using async.each**

First, install the async library:

bash

Copy code

npm install async

Then, use it in your code:

javascript

Copy code

const async = require('async');

const items = [1, 2, 3, 4, 5];

async.each(items, (item, callback) => {

// Simulate an asynchronous operation

setTimeout(() => {

console.log(`Processing item ${item}`);

callback(); // Call callback to indicate completion

}, 1000);

}, (err) => {

if (err) {

console.error('An error occurred:', err);

} else {

console.log('All items have been processed');

}

});

In this example, async.each processes each item in the items array with a simulated asynchronous operation. The final callback is called when all items are processed or if an error occurs.

**5. Express Routing**

Routing in Express.js is used to define the endpoints of your application and the corresponding request handlers.

**Example: Express Routing**

const express = require('express');

const app = express();

const port = 3000;

// Route for the root URL

app.get('/', (req, res) => {

res.send('Welcome to the root page!');

});

// Route for the /about URL

app.get('/about', (req, res) => {

res.send('This is the about page.');

});

// Route with a parameter

app.get('/user/:id', (req, res) => {

const userId = req.params.id;

res.send(`User ID is ${userId}`);

});

app.listen(port, () => {

console.log(`Server is running at http://localhost:${port}`);

});

In this example:

* / responds with a welcome message.
* /about responds with an about page message.
* /user/:id uses a route parameter (:id) to capture and display a dynamic value.

These examples should help you get started with Express.js, middleware, modules, async.each, and routing!

Integrating MongoDB with an Express.js application involves connecting to a MongoDB database, defining schemas, and interacting with the database for CRUD operations. Let's walk through a basic example using the mongoose library, which provides a straightforward way to work with MongoDB in Node.js.

**1. Setup**

First, ensure you have the necessary packages installed. You need express, mongoose, and body-parser for handling request bodies.

npm install express mongoose body-parser

**2. Create the Application**

Let's create a simple Express application that connects to MongoDB and provides basic CRUD operations.

**Structure:**

Copy code

myapp/

│

├── models/

│ └── user.js

│

├── routes/

│ └── userRoutes.js

│

└── app.js

**3. Setup MongoDB Connection**

**app.js**

javascript

Copy code

const express = require('express');

const mongoose = require('mongoose');

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

const userRoutes = require('./routes/userRoutes');

const app = express();

const port = 3000;

// Middleware

app.use(bodyParser.json());

// MongoDB connection

mongoose.connect('mongodb://localhost:27017/myapp', {

useNewUrlParser: true,

useUnifiedTopology: true

})

.then(() => console.log('MongoDB connected'))

.catch(err => console.error('MongoDB connection error:', err));

// Use routes

app.use('/users', userRoutes);

app.listen(port, () => {

console.log(`Server is running at http://localhost:${port}`);

});

**4. Define a Mongoose Model**

**models/user.js**

javascript

Copy code

const mongoose = require('mongoose');

// Define user schema

const userSchema = new mongoose.Schema({

name: String,

email: String,

age: Number

});

// Create user model

const User = mongoose.model('User', userSchema);

module.exports = User;

**5. Create Routes for CRUD Operations**

**routes/userRoutes.js**

javascript

Copy code

const express = require('express');

const router = express.Router();

const User = require('../models/user');

// Create a new user

router.post('/', async (req, res) => {

try {

const user = new User(req.body);

await user.save();

res.status(201).send(user);

} catch (error) {

res.status(400).send(error);

}

});

// Get all users

router.get('/', async (req, res) => {

try {

const users = await User.find();

res.send(users);

} catch (error) {

res.status(500).send(error);

}

});

// Get a user by ID

router.get('/:id', async (req, res) => {

try {

const user = await User.findById(req.params.id);

if (!user) {

return res.status(404).send('User not found');

}

res.send(user);

} catch (error) {

res.status(500).send(error);

}

});

// Update a user by ID

router.put('/:id', async (req, res) => {

try {

const user = await User.findByIdAndUpdate(req.params.id, req.body, { new: true, runValidators: true });

if (!user) {

return res.status(404).send('User not found');

}

res.send(user);

} catch (error) {

res.status(400).send(error);

}

});

// Delete a user by ID

router.delete('/:id', async (req, res) => {

try {

const user = await User.findByIdAndDelete(req.params.id);

if (!user) {

return res.status(404).send('User not found');

}

res.send(user);

} catch (error) {

res.status(500).send(error);

}

});

module.exports = router;

**6. Testing the Application**

You can test the CRUD operations using tools like Postman or curl. Here are some example curl commands:

* **Create a User:**

bash

Copy code

curl -X POST http://localhost:3000/users -H "Content-Type: application/json" -d '{"name":"John Doe", "email":"john@example.com", "age":30}'

* **Get All Users:**

bash

Copy code

curl http://localhost:3000/users

* **Get User by ID:**

bash

Copy code

curl http://localhost:3000/users/<user\_id>

* **Update User by ID:**

bash

Copy code

curl -X PUT http://localhost:3000/users/<user\_id> -H "Content-Type: application/json" -d '{"name":"Jane Doe"}'

* **Delete User by ID:**

bash

Copy code

curl -X DELETE http://localhost:3000/users/<user\_id>

In this example, we've created a simple Express.js application with a MongoDB backend, demonstrating how to set up a connection, define models, and implement CRUD operations using Express routing.