1. Setup

Install Express

```
npm init -y               # Initialize Node.js project
npm install express       # Install Express
npm install nodemon --save-dev # Install Nodemon for development
```

Basic Server Setup

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

const PORT = process.env.PORT || 3000;

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

---

2. Middleware

Middleware functions are used to handle HTTP requests and perform various operations before sending a response.

Built-in Middleware

- **Serve Static Files:**

  ```
  app.use(express.static('public')); // To serve static files from a folder named 'public'
  ```

- **Parse JSON Request Body:**

  ```
  app.use(express.json()); // To parse JSON request body
  ```

- **Parse URL-encoded Form Data:**

  ```
  app.use(express.urlencoded({ extended: true }));
  ```

Custom Middleware

```
app.use((req, res, next) => {
  console.log(`${req.method} request for '${req.url}'`);
  next(); // Pass control to the next middleware function
});
```

---

3. Routing

Basic Routes

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

Route Parameters

```
app.get('/user/:id', (req, res) => {
  const userId = req.params.id;
  res.send(`User ID is ${userId}`);
});
```

Query Parameters

```
app.get('/search', (req, res) => {
  const query = req.query.q;
  res.send(`You searched for: ${query}`);
});
```

Route Chaining

Multiple routes with the same path but different HTTP methods.

```
app.route('/user')
  .get((req, res) => {
    res.send('Get a user');
  })
  .post((req, res) => {
    res.send('Create a user');
  })
  .put((req, res) => {
    res.send('Update a user');
  });
```

---

4. Handling HTTP Methods

GET Request

```
app.get('/users', (req, res) => {
  res.send('GET request to /users');
});
```

POST Request

```
app.post('/users', (req, res) => {
  const newUser = req.body; // Assuming req.body has JSON data
  res.status(201).json(newUser);
});
```

PUT Request

```
app.put('/users/:id', (req, res) => {
  const userId = req.params.id;
  res.send(`User ${userId} updated`);
});
```

DELETE Request

```
app.delete('/users/:id', (req, res) => {
  const userId = req.params.id;
  res.send(`User ${userId} deleted`);
});
```

---

5. Error Handling

Basic Error Handling Middleware

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Something broke!');
});
```

Throwing Errors

```
app.get('/error', (req, res, next) => {
  const err = new Error('Something went wrong!');
  next(err); // Pass error to error-handling middleware
});
```

---

6. Routing with Express Router

Using Express Router

```
const express = require('express');
const router = express.Router();

router.get('/profile', (req, res) => {
  res.send('User profile page');
});

router.post('/profile', (req, res) => {
  res.send('Create user profile');
});

app.use('/user', router); // Mount the router at /user path
```

Route Prefixes

```
app.use('/api/users', require('./routes/users')); // Prefix all routes in
users.js with /api/users
```

---

7. Static File Handling

Serving Static Files

```
app.use(express.static('public')); // Serve static files from the 'public'
directory
```

---

9. Handling Form Data

URL-encoded Form

```
app.use(express.urlencoded({ extended: true }));

app.post('/submit', (req, res) => {
  const { name, email } = req.body;
  res.send(`Name: ${name}, Email: ${email}`);
});
```

---

10. CORS (Cross-Origin Resource Sharing)

Enabling CORS

```
npm install cors
```

```
const cors = require('cors');
app.use(cors()); // Enable CORS for all routes
```

To enable CORS for specific domains:

```
app.use(cors({
  origin: ['http://example.com', 'http://another-domain.com']
}));
```

---

11. Connecting to MongoDB (Mongoose)

Install Mongoose

```
npm install mongoose
```

Connecting to MongoDB

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

mongoose.connect('mongodb://localhost:27017/mydb', { useNewUrlParser: true,
useUnifiedTopology: true })
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.error('MongoDB connection error', err));
```

Creating a Mongoose Model

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

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

---

12. RESTful API Example

CRUD Operations

- **Create** a new user (POST):

```
app.post('/users', async (req, res) => {
  const newUser = new User(req.body);
  await newUser.save();
  res.status(201).json(newUser);
});
```

- **Read** all users (GET):

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

- **Update** a user by ID (PUT):

```
app.put('/users/:id', async (req, res) => {
  const updatedUser = await User.findByIdAndUpdate(req.params.id,
req.body, { new: true });
  res.json(updatedUser);
});
```

- **Delete** a user by ID (DELETE):

```
app.delete('/users/:id', async (req, res) => {
  await User.findByIdAndDelete(req.params.id);
  res.json({ message: 'User deleted' });
});
```

---

13. Environment Variables

Using dotenv

1. **Install dotenv**:

   ```
   npm install dotenv
   ```

2. **Create `.env` File**:

   ```
   PORT=3000
   MONGO_URI=mongodb://localhost:27017/mydb
   ```

3. **Load Environment Variables**:

   ```
   require('dotenv').config();

   const PORT = process.env.PORT || 3000;
   const MONGO_URI = process.env.MONGO_URI;
   ```

Security Best Practices

Rate Limiting (express-rate-limit)

```
npm install express-rate-limit


const rateLimit = require('express-rate-limit');

const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100 // Limit each IP to 100 requests per windowMs
});

app.use(limiter);
```

---

15. Useful Tools

Nodemon

Install `nodemon` for auto-restarting the server when code changes:

```
npm install --save-dev nodemon
```

Run your application with `nodemon`:

```
npx nodemon app.js
```