Complete Express.js Full-Stack Development Guide

Table of Contents

Express.js Fundamentals

Installation & Setup

```bash
# Initialize project
npm init -y

# Install Express
npm install express

# Install development dependencies
npm install -D nodemon
```

Basic Express Server

```javascript
```

```javascript
// app.js
const express = require('express');
const app = express();
const PORT = process.env.PORT || 3000;

// Middleware
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Basic route
app.get('/', (req, res) => {
  res.json({ message: 'Hello World!' });
});

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

Essential Middleware

```javascript
const cors = require('cors');
const helmet = require('helmet');
const morgan = require('morgan');

// Security
app.use(helmet());

// CORS
app.use(cors());

// Logging
app.use(morgan('combined'));

// Rate limiting
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);
```

REST API Creation

RESTful Routes Structure

```
GET    /api/users      - Get all users
GET    /api/users/:id  - Get specific user
POST   /api/users      - Create new user
PUT    /api/users/:id  - Update entire user
PATCH  /api/users/:id  - Partial update user
DELETE /api/users/:id  - Delete user
```

Basic REST API Implementation

javascript

```
// routes/users.js
const express = require('express');
const router = express.Router();

let users = []; // In-memory storage (use database in production)

// GET all users
router.get('/', (req, res) => {
  res.json({
    success: true,
    data: users,
    count: users.length
  });
});

// GET single user
router.get('/:id', (req, res) => {
  const user = users.find(u => u.id === parseInt(req.params.id));

  if (!user) {
    return res.status(404).json({
      success: false,
      message: 'User not found'
    });
  }

  res.json({
    success: true,
    data: user
  });
});

// CREATE user
router.post('/', (req, res) => {
  const { name, email } = req.body;

  if (!name || !email) {
    return res.status(400).json({
      success: false,
      message: 'Name and email are required'
    });
  }

  const newUser = {
    id: users.length + 1,
    name,
```

```
    email,
    createdAt: new Date()
  };

  users.push(newUser);

  res.status(201).json({
    success: true,
    data: newUser
  });
});

// UPDATE user
router.put('/:id', (req, res) => {
  const userIndex = users.findIndex(u => u.id === parseInt(req.params.id));

  if (userIndex === -1) {
    return res.status(404).json({
      success: false,
      message: 'User not found'
    });
  }

  users[userIndex] = {
    ...users[userIndex],
    ...req.body,
    updatedAt: new Date()
  };

  res.json({
    success: true,
    data: users[userIndex]
  });
});

// DELETE user
router.delete('/:id', (req, res) => {
  const userIndex = users.findIndex(u => u.id === parseInt(req.params.id));

  if (userIndex === -1) {
    return res.status(404).json({
      success: false,
      message: 'User not found'
    });
  }

  const deletedUser = users.splice(userIndex, 1);
```

```
  res.json({
    success: true,
    data: deletedUser[0]
  });
});

module.exports = router;
```

Error Handling Middleware

javascript

```
// middleware/errorHandler.js
const errorHandler = (err, req, res, next) => {
  let error = { ...err };
  error.message = err.message;

  // Log error
  console.error(err);

  // Mongoose bad ObjectId
  if (err.name === 'CastError') {
    const message = 'Resource not found';
    error = { message, statusCode: 404 };
  }

  // Mongoose duplicate key
  if (err.code === 11000) {
    const message = 'Duplicate field value entered';
    error = { message, statusCode: 400 };
  }

  // Mongoose validation error
  if (err.name === 'ValidationError') {
    const message = Object.values(err.errors).map(val => val.message).join(', ');
    error = { message, statusCode: 400 };
  }

  res.status(error.statusCode || 500).json({
    success: false,
    error: error.message || 'Server Error'
  });
};

module.exports = errorHandler;
```

MVC Architecture

MVC (Model-View-Controller) is a design pattern that separates application logic into three components:

Project Structure

```
project/
├── controllers/     # Handle request/response logic
├── models/          # Data models and database schemas
├── routes/          # Route definitions
├── middleware/      # Custom middleware functions
├── views/           # Templates (if using server-side rendering)
├── config/          # Configuration files
├── utils/           # Utility functions
└── app.js           # Main application file
```

Model Example

javascript

```javascript
// models/User.js
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Please provide a name'],
    trim: true,
    maxlength: [50, 'Name cannot be more than 50 characters']
  },
  email: {
    type: String,
    required: [true, 'Please provide an email'],
    unique: true,
    match: [
      /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/,
      'Please provide a valid email'
    ]
  },
  role: {
    type: String,
    enum: ['user', 'admin'],
    default: 'user'
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
});

module.exports = mongoose.model('User', userSchema);
```

Controller Example

javascript

```
// controllers/userController.js
const User = require('../models/User');
const asyncHandler = require('../middleware/async');

// @desc    Get all users
// @route   GET /api/users
// @access  Public
exports.getUsers = asyncHandler(async (req, res, next) => {
  const users = await User.find();

  res.status(200).json({
    success: true,
    count: users.length,
    data: users
  });
});

// @desc    Get single user
// @route   GET /api/users/:id
// @access  Public
exports.getUser = asyncHandler(async (req, res, next) => {
  const user = await User.findById(req.params.id);

  if (!user) {
    return res.status(404).json({
      success: false,
      message: 'User not found'
    });
  }

  res.status(200).json({
    success: true,
    data: user
  });
});

// @desc    Create user
// @route   POST /api/users
// @access  Public
exports.createUser = asyncHandler(async (req, res, next) => {
  const user = await User.create(req.body);

  res.status(201).json({
    success: true,
    data: user
  });
```

```
});

// @desc    Update user
// @route   PUT /api/users/:id
// @access  Public
exports.updateUser = asyncHandler(async (req, res, next) => {
  const user = await User.findByIdAndUpdate(req.params.id, req.body, {
    new: true,
    runValidators: true
  });

  if (!user) {
    return res.status(404).json({
      success: false,
      message: 'User not found'
    });
  }

  res.status(200).json({
    success: true,
    data: user
  });
});

// @desc    Delete user
// @route   DELETE /api/users/:id
// @access  Public
exports.deleteUser = asyncHandler(async (req, res, next) => {
  const user = await User.findByIdAndDelete(req.params.id);

  if (!user) {
    return res.status(404).json({
      success: false,
      message: 'User not found'
    });
  }

  res.status(200).json({
    success: true,
    data: {}
  });
});
```

Routes with Controllers

javascript

```javascript
// routes/users.js
const express = require('express');
const {
  getUsers,
  getUser,
  createUser,
  updateUser,
  deleteUser
} = require('../controllers/userController');

const router = express.Router();

router.route('/')
  .get(getUsers)
  .post(createUser);

router.route('/:id')
  .get(getUser)
  .put(updateUser)
  .delete(deleteUser);

module.exports = router;
```

MongoDB & Mongoose

Installation

bash

```
npm install mongoose
```

Database Connection

javascript

```
// config/database.js
const mongoose = require('mongoose');

const connectDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGODB_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });

    console.log(`MongoDB Connected: ${conn.connection.host}`);
  } catch (error) {
    console.error(error);
    process.exit(1);
  }
};

module.exports = connectDB;
```

Mongoose Schema Types

javascript

```
const userSchema = new mongoose.Schema({
  // String
  name: {
    type: String,
    required: true,
    trim: true,
    minlength: 2,
    maxlength: 50
  },

  // Number
  age: {
    type: Number,
    min: 0,
    max: 120
  },

  // Boolean
  isActive: {
    type: Boolean,
    default: true
  },

  // Date
  createdAt: {
    type: Date,
    default: Date.now
  },

  // Array
  tags: [String],

  // Object ID (Reference)
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },

  // Embedded Document
  address: {
    street: String,
    city: String,
    zipCode: String
  },
```

```
  // Enum
  role: {
    type: String,
    enum: ['user', 'admin', 'moderator'],
    default: 'user'
  }
});
```

Mongoose Middleware (Hooks)

javascript

```
// Pre-save middleware
userSchema.pre('save', async function(next) {
  // Hash password before saving
  if (!this.isModified('password')) {
    next();
  }

  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
});

// Post-save middleware
userSchema.post('save', function(doc, next) {
  console.log('User saved:', doc.name);
  next();
});

// Pre-remove middleware
userSchema.pre('remove', async function(next) {
  // Remove user's posts when user is deleted
  await this.model('Post').deleteMany({ user: this._id });
  next();
});
```

Advanced Mongoose Queries

javascript

```javascript
// Find with conditions
const users = await User.find({ age: { $gte: 18 } });

// Find with select
const users = await User.find().select('name email');

// Find with populate (for references)
const posts = await Post.find().populate('user', 'name email');

// Find with pagination
const page = req.query.page || 1;
const limit = req.query.limit || 10;
const skip = (page - 1) * limit;

const users = await User.find()
  .skip(skip)
  .limit(limit)
  .sort({ createdAt: -1 });

// Aggregation pipeline
const stats = await User.aggregate([
  { $match: { isActive: true } },
  { $group: {
    _id: '$role',
    count: { $sum: 1 },
    avgAge: { $avg: '$age' }
  }},
  { $sort: { count: -1 } }
]);
```

Validation & Error Handling

javascript

```
// Custom validator
const userSchema = new mongoose.Schema({
  email: {
    type: String,
    validate: {
      validator: function(v) {
        return /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/.test(v);
      },
      message: props => `${props.value} is not a valid email address!`
    }
  },

  // Async validator
  username: {
    type: String,
    validate: {
      validator: async function(username) {
        const user = await mongoose.models.User.findOne({ username });
        return !user;
      },
      message: 'Username already exists'
    }
  }
});
```

PostgreSQL Fundamentals

Installation

bash

```
# Ubuntu/Debian
sudo apt-get update
sudo apt-get install postgresql postgresql-contrib

# macOS with Homebrew
brew install postgresql

# Start PostgreSQL service
sudo service postgresql start  # Linux
brew services start postgresql  # macOS
```

Basic PostgreSQL Setup

```bash
# Switch to postgres user
sudo -i -u postgres

# Create database
createdb myapp_db

# Access PostgreSQL prompt
psql

# Connect to specific database
\c myapp_db

# Exit
\q
```

SQL Commands Reference

Database Operations

```sql
-- Create Database
CREATE DATABASE myapp_db;

-- List Databases
\l

-- Connect to Database
\c myapp_db;

-- Drop Database
DROP DATABASE myapp_db;
```

Table Operations

```sql
```

```sql
-- Create Table
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    age INTEGER CHECK (age >= 0),
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- List Tables
\dt

-- Describe Table
\d users

-- Drop Table
DROP TABLE users;

-- Alter Table
ALTER TABLE users ADD COLUMN phone VARCHAR(20);
ALTER TABLE users DROP COLUMN phone;
ALTER TABLE users ALTER COLUMN name TYPE VARCHAR(150);
```

CRUD Operations

Create (INSERT)

sql

```sql
-- Insert single record
INSERT INTO users (name, email, age)
VALUES ('John Doe', 'john@example.com', 30);

-- Insert multiple records
INSERT INTO users (name, email, age) VALUES
    ('Jane Smith', 'jane@example.com', 25),
    ('Bob Johnson', 'bob@example.com', 35);

-- Insert with RETURNING
INSERT INTO users (name, email, age)
VALUES ('Alice Brown', 'alice@example.com', 28)
RETURNING id, name;
```

Read (SELECT)

sql

```sql
-- Select all
SELECT * FROM users;

-- Select specific columns
SELECT name, email FROM users;

-- Select with WHERE clause
SELECT * FROM users WHERE age > 25;

-- Select with multiple conditions
SELECT * FROM users WHERE age > 25 AND is_active = TRUE;

-- Select with LIKE (pattern matching)
SELECT * FROM users WHERE name LIKE 'John%';

-- Select with ORDER BY
SELECT * FROM users ORDER BY age DESC;

-- Select with LIMIT and OFFSET
SELECT * FROM users ORDER BY id LIMIT 5 OFFSET 10;

-- Select with COUNT
SELECT COUNT(*) FROM users;

-- Select with GROUP BY
SELECT is_active, COUNT(*) FROM users GROUP BY is_active;

-- Select with HAVING
SELECT age, COUNT(*) FROM users GROUP BY age HAVING COUNT(*) > 1;
```

Update (UPDATE)

sql

```sql
-- Update single record
UPDATE users SET age = 31 WHERE id = 1;

-- Update multiple columns
UPDATE users SET name = 'John Smith', age = 32 WHERE id = 1;

-- Update with condition
UPDATE users SET is_active = FALSE WHERE age < 18;

-- Update with RETURNING
UPDATE users SET age = age + 1 WHERE id = 1 RETURNING *;
```

Delete (DELETE)

sql

```sql
-- Delete specific record
DELETE FROM users WHERE id = 1;

-- Delete with condition
DELETE FROM users WHERE is_active = FALSE;

-- Delete all records (be careful!)
DELETE FROM users;

-- Delete with RETURNING
DELETE FROM users WHERE id = 1 RETURNING *;
```

Advanced SQL Operations

Joins

sql

```sql
-- Create posts table for join examples
CREATE TABLE posts (
    id SERIAL PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    content TEXT,
    user_id INTEGER REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- INNER JOIN
SELECT u.name, p.title
FROM users u
INNER JOIN posts p ON u.id = p.user_id;

-- LEFT JOIN
SELECT u.name, p.title
FROM users u
LEFT JOIN posts p ON u.id = p.user_id;

-- RIGHT JOIN
SELECT u.name, p.title
FROM users u
RIGHT JOIN posts p ON u.id = p.user_id;
```

Subqueries

```sql
-- Subquery in WHERE
SELECT * FROM users
WHERE id IN (SELECT user_id FROM posts WHERE title LIKE '%SQL%');

-- Subquery in FROM
SELECT avg_age.average
FROM (SELECT AVG(age) as average FROM users) as avg_age;
```

Window Functions

```sql
```

```sql
-- ROW_NUMBER
SELECT name, age,
    ROW_NUMBER() OVER (ORDER BY age DESC) as rank
FROM users;

-- RANK
SELECT name, age,
    RANK() OVER (ORDER BY age DESC) as rank
FROM users;

-- PARTITION BY
SELECT name, age, is_active,
    AVG(age) OVER (PARTITION BY is_active) as avg_age_by_status
FROM users;
```

Indexes

```sql
-- Create index
CREATE INDEX idx_users_email ON users(email);

-- Create unique index
CREATE UNIQUE INDEX idx_users_email_unique ON users(email);

-- Create partial index
CREATE INDEX idx_active_users ON users(name) WHERE is_active = TRUE;

-- Drop index
DROP INDEX idx_users_email;

-- List indexes
\di
```

PostgreSQL with Express (pg)

Installation

```bash
npm install pg
npm install -D @types/pg  # If using TypeScript
```

Database Connection

javascript

```javascript
// config/database.js
const { Pool } = require('pg');

const pool = new Pool({
  user: process.env.DB_USER || 'postgres',
  host: process.env.DB_HOST || 'localhost',
  database: process.env.DB_NAME || 'myapp_db',
  password: process.env.DB_PASSWORD || 'password',
  port: process.env.DB_PORT || 5432,
});

// Test connection
pool.query('SELECT NOW()', (err, res) => {
  if (err) {
    console.error('Database connection error:', err);
  } else {
    console.log('Database connected successfully');
  }
});

module.exports = pool;
```

User Model with pg

javascript

```
// models/User.js
const pool = require('../config/database');

class User {
  // Get all users
  static async getAll() {
    const query = 'SELECT * FROM users ORDER BY created_at DESC';
    const result = await pool.query(query);
    return result.rows;
  }

  // Get user by ID
  static async getById(id) {
    const query = 'SELECT * FROM users WHERE id = $1';
    const result = await pool.query(query, [id]);
    return result.rows[0];
  }

  // Create user
  static async create(userData) {
    const { name, email, age } = userData;
    const query = `
      INSERT INTO users (name, email, age)
      VALUES ($1, $2, $3)
      RETURNING *
    `;
    const result = await pool.query(query, [name, email, age]);
    return result.rows[0];
  }

  // Update user
  static async update(id, userData) {
    const { name, email, age } = userData;
    const query = `
      UPDATE users
      SET name = $1, email = $2, age = $3
      WHERE id = $4
      RETURNING *
    `;
    const result = await pool.query(query, [name, email, age, id]);
    return result.rows[0];
  }

  // Delete user
  static async delete(id) {
    const query = 'DELETE FROM users WHERE id = $1 RETURNING *';
```

```
    const result = await pool.query(query, [id]);
    return result.rows[0];
  }

  // Find by email
  static async findByEmail(email) {
    const query = 'SELECT * FROM users WHERE email = $1';
    const result = await pool.query(query, [email]);
    return result.rows[0];
  }

  // Get users with pagination
  static async getPaginated(page = 1, limit = 10) {
    const offset = (page - 1) * limit;
    const query = `
      SELECT * FROM users
      ORDER BY created_at DESC
      LIMIT $1 OFFSET $2
    `;
    const countQuery = 'SELECT COUNT(*) FROM users';

    const [users, count] = await Promise.all([
      pool.query(query, [limit, offset]),
      pool.query(countQuery)
    ]);

    return {
      users: users.rows,
      totalCount: parseInt(count.rows[0].count),
      totalPages: Math.ceil(count.rows[0].count / limit),
      currentPage: page
    };
  }
}

module.exports = User;
```

Controller with pg

```
javascript
```

```
// controllers/userController.js
const User = require('../models/User');

// Get all users
exports.getUsers = async (req, res) => {
  try {
    const page = parseInt(req.query.page) || 1;
    const limit = parseInt(req.query.limit) || 10;

    const result = await User.getPaginated(page, limit);

    res.status(200).json({
      success: true,
      data: result.users,
      pagination: {
        currentPage: result.currentPage,
        totalPages: result.totalPages,
        totalCount: result.totalCount,
        hasNext: result.currentPage < result.totalPages,
        hasPrev: result.currentPage > 1
      }
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: 'Server Error',
      error: error.message
    });
  }
};

// Get single user
exports.getUser = async (req, res) => {
  try {
    const user = await User.getById(req.params.id);

    if (!user) {
      return res.status(404).json({
        success: false,
        message: 'User not found'
      });
    }

    res.status(200).json({
      success: true,
      data: user
```

```
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: 'Server Error',
      error: error.message
    });
  }
};

// Create user
exports.createUser = async (req, res) => {
  try {
    const { name, email, age } = req.body;

    // Check if user already exists
    const existingUser = await User.findByEmail(email);
    if (existingUser) {
      return res.status(400).json({
        success: false,
        message: 'User with this email already exists'
      });
    }

    const user = await User.create({ name, email, age });

    res.status(201).json({
      success: true,
      data: user
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: 'Server Error',
      error: error.message
    });
  }
};

// Update user
exports.updateUser = async (req, res) => {
  try {
    const user = await User.update(req.params.id, req.body);

    if (!user) {
      return res.status(404).json({
        success: false,
```

```
        message: 'User not found'
      });
    }

    res.status(200).json({
      success: true,
      data: user
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: 'Server Error',
      error: error.message
    });
  }
};

// Delete user
exports.deleteUser = async (req, res) => {
  try {
    const user = await User.delete(req.params.id);

    if (!user) {
      return res.status(404).json({
        success: false,
        message: 'User not found'
      });
    }

    res.status(200).json({
      success: true,
      message: 'User deleted successfully',
      data: user
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: 'Server Error',
      error: error.message
    });
  }
};
```

Database Migrations

javascript

```
// migrations/001_create_users_table.js
const pool = require('../config/database');

const up = async () => {
  const query = `
    CREATE TABLE IF NOT EXISTS users (
      id SERIAL PRIMARY KEY,
      name VARCHAR(100) NOT NULL,
      email VARCHAR(255) UNIQUE NOT NULL,
      age INTEGER CHECK (age >= 0),
      is_active BOOLEAN DEFAULT TRUE,
      created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    );

    CREATE INDEX IF NOT EXISTS idx_users_email ON users(email);
    CREATE INDEX IF NOT EXISTS idx_users_active ON users(is_active);
  `;

  await pool.query(query);
  console.log('Users table created successfully');
};

const down = async () => {
  const query = 'DROP TABLE IF EXISTS users CASCADE;';
  await pool.query(query);
  console.log('Users table dropped successfully');
};

module.exports = { up, down };
```

Prisma ORM

Installation & Setup

```
bash

# Install Prisma
npm install prisma @prisma/client

# Initialize Prisma
npx prisma init
```

Prisma Schema