Complete Express.js Playbook

Table of Contents

Introduction

Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It's the most popular Node.js framework and serves as the foundation for many other frameworks.

Key Features

- Fast, unopinionated, minimalist web framework
- Robust routing system
- Middleware support
- Template engine integration
- Static file serving

- Error handling

Getting Started

Installation

bash

```bash
# Create a new project
mkdir my-express-app
cd my-express-app

# Initialize npm
npm init -y

# Install Express
npm install express

# Install development dependencies
npm install --save-dev nodemon
```

Basic Server Setup

javascript

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

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

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

Package.json Scripts

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

Core Concepts

Application Object

The Express application object represents your web application and contains methods for:

- Routing HTTP requests
- Configuring middleware
- Rendering HTML views
- Registering template engines

```javascript
const app = express();

// Application settings
app.set('view engine', 'ejs');
app.set('views', './views');

// Application-level middleware
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
```

HTTP Methods

Express provides methods corresponding to HTTP verbs:

```javascript
app.get('/', handler);     // GET requests
app.post('/', handler);    // POST requests
app.put('/', handler);     // PUT requests
app.delete('/', handler);  // DELETE requests
app.patch('/', handler);   // PATCH requests
app.all('/', handler);     // All HTTP methods
```

Routing

Basic Routing

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

// POST route
app.post('/users', (req, res) => {
  res.json({ message: 'Create user', data: req.body });
});

// PUT route
app.put('/users/:id', (req, res) => {
  res.json({ message: `Update user ${req.params.id}` });
});

// DELETE route
app.delete('/users/:id', (req, res) => {
  res.json({ message: `Delete user ${req.params.id}` });
});
```

Route Parameters

```javascript
// Single parameter
app.get('/users/:id', (req, res) => {
  const userId = req.params.id;
  res.json({ userId });
});

// Multiple parameters
app.get('/users/:userId/posts/:postId', (req, res) => {
  const { userId, postId } = req.params;
  res.json({ userId, postId });
});

// Optional parameters
app.get('/posts/:year/:month?', (req, res) => {
  const { year, month } = req.params;
  res.json({ year, month: month || 'all' });
});
```

Query Parameters

```javascript
app.get('/search', (req, res) => {
  const { q, page = 1, limit = 10 } = req.query;
  res.json({ query: q, page: parseInt(page), limit: parseInt(limit) });
});
// URL: /search?q=nodejs&page=2&limit=20
```

Route Patterns

```javascript
// Wildcard
app.get('/files/*', (req, res) => {
  res.json({ path: req.params[0] });
});

// Regular expressions
app.get(/.*fly$/, (req, res) => {
  res.send('Ends with fly');
});

// String patterns
app.get('/ab?cd', (req, res) => {
  res.send('Matches /acd or /abcd');
});
```

Express Router

javascript

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

// Middleware specific to this router
router.use((req, res, next) => {
  console.log('Users router middleware');
  next();
});

router.get('/', (req, res) => {
  res.json({ message: 'Users list' });
});

router.get('/:id', (req, res) => {
  res.json({ message: `User ${req.params.id}` });
});

router.post('/', (req, res) => {
  res.json({ message: 'Create user', data: req.body });
});

module.exports = router;

// app.js
const userRoutes = require('./routes/users');
app.use('/users', userRoutes);
```

Middleware

Middleware functions execute during the request-response cycle and can:

- Execute code
- Modify request/response objects
- End the request-response cycle
- Call the next middleware function

Types of Middleware

Application-level Middleware

```javascript
// Executed for every request
app.use((req, res, next) => {
  console.log(`${req.method} ${req.url} - ${new Date().toISOString()}`);
  next();
});

// Executed for specific paths
app.use('/api', (req, res, next) => {
  console.log('API middleware');
  next();
});
```

Built-in Middleware

```javascript
// Parse JSON bodies
app.use(express.json());

// Parse URL-encoded bodies
app.use(express.urlencoded({ extended: true }));

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

Third-party Middleware

```javascript
// Install: npm install cors helmet morgan
const cors = require('cors');
const helmet = require('helmet');
const morgan = require('morgan');

app.use(cors());
app.use(helmet());
app.use(morgan('combined'));
```

Custom Middleware

```javascript
// Authentication middleware
const authenticate = (req, res, next) => {
  const token = req.headers.authorization;

  if (!token) {
    return res.status(401).json({ error: 'No token provided' });
  }

  // Verify token logic here
  req.user = { id: 1, name: 'John Doe' }; // Mock user
  next();
};

// Rate limiting middleware
const rateLimit = (windowMs, max) => {
  const requests = new Map();

  return (req, res, next) => {
    const ip = req.ip;
    const now = Date.now();
    const windowStart = now - windowMs;

    if (!requests.has(ip)) {
      requests.set(ip, []);
    }

    const requestTimes = requests.get(ip).filter(time => time > windowStart);

    if (requestTimes.length >= max) {
      return res.status(429).json({ error: 'Too many requests' });
    }

    requestTimes.push(now);
    requests.set(ip, requestTimes);
    next();
  };
};

// Usage
app.use('/api', rateLimit(15 * 60 * 1000, 100)); // 100 requests per 15 minutes
app.use('/api/protected', authenticate);
```

Error Handling Middleware

```javascript
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ error: 'Something went wrong!' });
});
```

Request and Response Objects

Request Object (req)

```javascript
app.get('/demo', (req, res) => {
  // Request properties
  console.log('Method:', req.method);
  console.log('URL:', req.url);
  console.log('Headers:', req.headers);
  console.log('Query:', req.query);
  console.log('Params:', req.params);
  console.log('Body:', req.body);
  console.log('IP:', req.ip);
  console.log('Protocol:', req.protocol);
  console.log('Hostname:', req.hostname);
  console.log('Path:', req.path);

  res.json({ message: 'Request received' });
});
```

Response Object (res)

```javascript
app.get('/response-demo', (req, res) => {
  // Set status code
  res.status(200);

  // Set headers
  res.set('Custom-Header', 'value');
  res.set({
    'Another-Header': 'another-value',
    'Content-Type': 'application/json'
  });

  // Send different types of responses
  // res.send('Plain text');
  // res.json({ key: 'value' });
  // res.render('template', { data });
  // res.redirect('/other-route');
  // res.download('/path/to/file');

  res.json({ message: 'Response sent' });
});

// Method chaining
app.get('/chain', (req, res) => {
  res.status(201).set('Location', '/users/123').json({ id: 123, name: 'John' });
});
```

Error Handling

Synchronous Error Handling

```javascript
app.get('/sync-error', (req, res, next) => {
  try {
    // Some operation that might throw
    const result = JSON.parse('invalid json');
    res.json(result);
  } catch (error) {
    next(error); // Pass error to error handler
  }
});
```

Asynchronous Error Handling

```javascript
// Async wrapper utility
const asyncHandler = (fn) => {
  return (req, res, next) => {
    Promise.resolve(fn(req, res, next)).catch(next);
  };
};

app.get('/async-error', asyncHandler(async (req, res) => {
  const data = await someAsyncOperation();
  res.json(data);
}));
```

Custom Error Classes

```javascript
class AppError extends Error {
  constructor(message, statusCode) {
    super(message);
    this.statusCode = statusCode;
    this.isOperational = true;

    Error.captureStackTrace(this, this.constructor);
  }
}

// Usage
app.get('/custom-error', (req, res, next) => {
  next(new AppError('User not found', 404));
});
```

Global Error Handler

```javascript
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 = new AppError(message, 404);
  }

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

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

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

app.use(errorHandler);
```

Static Files

Basic Static File Serving

javascript

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

// Multiple static directories
app.use(express.static('public'));
app.use(express.static('files'));

// Virtual path prefix
app.use('/static', express.static('public'));
```

Advanced Static File Configuration

javascript

```javascript
const path = require('path');

// Static file options
app.use('/static', express.static(path.join(__dirname, 'public'), {
  dotfiles: 'ignore',
  etag: false,
  extensions: ['htm', 'html'],
  index: false,
  maxAge: '1d',
  redirect: false,
  setHeaders: (res, path, stat) => {
    res.set('x-timestamp', Date.now());
  }
}));
```

Template Engines

EJS Setup

bash

```bash
npm install ejs
```

javascript

```javascript
app.set('view engine', 'ejs');
app.set('views', './views');

app.get('/template', (req, res) => {
  res.render('index', {
    title: 'My App',
    users: ['John', 'Jane', 'Bob']
  });
});
```

html

```html
<!-- views/index.ejs -->
<!DOCTYPE html>
<html>
<head>
  <title><%= title %></title>
</head>
<body>
  <h1>Welcome to <%= title %></h1>
  <ul>
    <% users.forEach(user => { %>
      <li><%= user %></li>
    <% }); %>
  </ul>
</body>
</html>
```

Handlebars Setup

bash

```bash
npm install express-handlebars
```

javascript

```javascript
const exphbs = require('express-handlebars');

app.engine('handlebars', exphbs());
app.set('view engine', 'handlebars');

app.get('/handlebars', (req, res) => {
  res.render('home', { name: 'World' });
});
```

Database Integration

MongoDB with Mongoose

```bash
npm install mongoose
```

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

// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/myapp', {
  useNewUrlParser: true,
  useUnifiedTopology: true
});

// User schema
const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  createdAt: { type: Date, default: Date.now }
});

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

// Routes
app.get('/users', async (req, res) => {
  try {
    const users = await User.find();
    res.json(users);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

app.post('/users', async (req, res) => {
  try {
    const user = new User(req.body);
    await user.save();
    res.status(201).json(user);
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});
```

PostgreSQL with Sequelize

```bash
npm install sequelize pg pg-hstore
```

javascript

```javascript
const { Sequelize, DataTypes } = require('sequelize');

const sequelize = new Sequelize('database', 'username', 'password', {
  host: 'localhost',
  dialect: 'postgres'
});

// User model
const User = sequelize.define('User', {
  name: {
    type: DataTypes.STRING,
    allowNull: false
  },
  email: {
    type: DataTypes.STRING,
    allowNull: false,
    unique: true
  }
});

// Sync database
sequelize.sync();

// Routes
app.get('/users', async (req, res) => {
  try {
    const users = await User.findAll();
    res.json(users);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

Authentication & Authorization

JWT Authentication

bash

```bash
npm install jsonwebtoken bcryptjs
```

javascript

```
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');

const JWT_SECRET = process.env.JWT_SECRET || 'your-secret-key';

// Register route
app.post('/register', async (req, res) => {
  try {
    const { name, email, password } = req.body;

    // Hash password
    const hashedPassword = await bcrypt.hash(password, 10);

    // Create user
    const user = new User({ name, email, password: hashedPassword });
    await user.save();

    // Generate token
    const token = jwt.sign({ userId: user._id }, JWT_SECRET, { expiresIn: '7d' });

    res.status(201).json({ token, user: { id: user._id, name, email } });
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});

// Login route
app.post('/login', async (req, res) => {
  try {
    const { email, password } = req.body;

    // Find user
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(401).json({ error: 'Invalid credentials' });
    }

    // Check password
    const isValidPassword = await bcrypt.compare(password, user.password);
    if (!isValidPassword) {
      return res.status(401).json({ error: 'Invalid credentials' });
    }

    // Generate token
    const token = jwt.sign({ userId: user._id }, JWT_SECRET, { expiresIn: '7d' });
    res.json({ token, user: { id: user._id, name: user.name, email } });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// Auth middleware
const authenticateToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {
    return res.status(401).json({ error: 'Access token required' });
  }

  jwt.verify(token, JWT_SECRET, async (err, decoded) => {
    if (err) {
      return res.status(403).json({ error: 'Invalid token' });
    }

    try {
      const user = await User.findById(decoded.userId);
      req.user = user;
      next();
    } catch (error) {
      res.status(500).json({ error: error.message });
    }
  });
};

// Protected route
app.get('/profile', authenticateToken, (req, res) => {
  res.json({ user: req.user });
});
```

Role-based Authorization

javascript

```javascript
const authorize = (...roles) => {
  return (req, res, next) => {
    if (!req.user) {
      return res.status(401).json({ error: 'Unauthorized' });
    }

    if (!roles.includes(req.user.role)) {
      return res.status(403).json({ error: 'Forbidden' });
    }

    next();
  };
};

// Usage
app.delete('/users/:id', authenticateToken, authorize('admin'), (req, res) => {
  // Only admins can delete users
  res.json({ message: 'User deleted' });
});
```

File Upload

Using Multer

bash

```bash
npm install multer
```

```
javascript
```

```
const multer = require('multer');
const path = require('path');

// Storage configuration
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'uploads/');
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + '-' + Math.round(Math.random() * 1E9) + path.extname(file.originalnam
  }
});

// File filter
const fileFilter = (req, file, cb) => {
  if (file.mimetype.startsWith('image/')) {
    cb(null, true);
  } else {
    cb(new Error('Only image files are allowed'), false);
  }
};

const upload = multer({
  storage,
  fileFilter,
  limits: { fileSize: 5 * 1024 * 1024 } // 5MB
});

// Single file upload
app.post('/upload', upload.single('image'), (req, res) => {
  if (!req.file) {
    return res.status(400).json({ error: 'No file uploaded' });
  }

  res.json({
    message: 'File uploaded successfully',
    file: req.file
  });
});

// Multiple files upload
app.post('/upload-multiple', upload.array('images', 5), (req, res) => {
  res.json({
    message: 'Files uploaded successfully',
    files: req.files
    });
  });
```

API Development

RESTful API Structure

javascript

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

exports.getUsers = async (req, res) => {
  try {
    const page = parseInt(req.query.page) || 1;
    const limit = parseInt(req.query.limit) || 10;
    const skip = (page - 1) * limit;

    const users = await User.find().skip(skip).limit(limit);
    const total = await User.countDocuments();

    res.json({
      users,
      pagination: {
        page,
        limit,
        total,
        pages: Math.ceil(total / limit)
      }
    });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

exports.getUser = async (req, res) => {
  try {
    const user = await User.findById(req.params.id);
    if (!user) {
      return res.status(404).json({ error: 'User not found' });
    }
    res.json(user);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

exports.createUser = async (req, res) => {
  try {
    const user = new User(req.body);
    await user.save();
    res.status(201).json(user);
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
};

exports.updateUser = async (req, res) => {
  try {
    const user = await User.findByIdAndUpdate(
      req.params.id,
      req.body,
      { new: true, runValidators: true }
    );
    if (!user) {
      return res.status(404).json({ error: 'User not found' });
    }
    res.json(user);
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
};

exports.deleteUser = async (req, res) => {
  try {
    const user = await User.findByIdAndDelete(req.params.id);
    if (!user) {
      return res.status(404).json({ error: 'User not found' });
    }
    res.status(204).send();
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

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

router.get('/', userController.getUsers);
router.get('/:id', userController.getUser);
router.post('/', userController.createUser);
router.put('/:id', userController.updateUser);
router.delete('/:id', userController.deleteUser);

module.exports = router;
```

API Validation

bash

```bash
npm install joi
```

javascript

```javascript
const Joi = require('joi');

const validateUser = (req, res, next) => {
  const schema = Joi.object({
    name: Joi.string().min(2).max(50).required(),
    email: Joi.string().email().required(),
    age: Joi.number().integer().min(0).max(120)
  });

  const { error } = schema.validate(req.body);
  if (error) {
    return res.status(400).json({ error: error.details[0].message });
  }

  next();
};

app.post('/users', validateUser, userController.createUser);
```

Security Best Practices

Essential Security Middleware

bash

```bash
npm install helmet cors express-rate-limit express-mongo-sanitize xss-clean hpp
```

```javascript
const helmet = require('helmet');
const cors = require('cors');
const rateLimit = require('express-rate-limit');
const mongoSanitize = require('express-mongo-sanitize');
const xss = require('xss-clean');
const hpp = require('hpp');

// Set security headers
app.use(helmet());

// Enable CORS
app.use(cors({
  origin: process.env.FRONTEND_URL || 'http://localhost:3000',
  credentials: true
}));

// Rate limiting
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100 // limit each IP to 100 requests per windowMs
});
app.use('/api/', limiter);

// Data sanitization against NoSQL query injection
app.use(mongoSanitize());

// Data sanitization against XSS
app.use(xss());

// Prevent parameter pollution
app.use(hpp());
```

Input Validation and Sanitization

javascript

```javascript
const validator = require('validator');

const sanitizeInput = (req, res, next) => {
  if (req.body.email) {
    req.body.email = validator.normalizeEmail(req.body.email);
  }

  if (req.body.name) {
    req.body.name = validator.escape(req.body.name);
  }

  next();
};

app.use(sanitizeInput);
```

Testing

Unit Testing with Jest

bash

```bash
npm install --save-dev jest supertest
```

```javascript
// tests/app.test.js
const request = require('supertest');
const app = require('../app');

describe('User API', () => {
  test('GET /users should return users list', async () => {
    const response = await request(app)
      .get('/users')
      .expect(200);

    expect(response.body).toHaveProperty('users');
    expect(Array.isArray(response.body.users)).toBe(true);
  });

  test('POST /users should create a new user', async () => {
    const userData = {
      name: 'John Doe',
      email: 'john@example.com'
    };

    const response = await request(app)
      .post('/users')
      .send(userData)
      .expect(201);

    expect(response.body).toHaveProperty('name', userData.name);
    expect(response.body).toHaveProperty('email', userData.email);
  });
});
```

Integration Testing

javascript

```javascript
// tests/integration/users.test.js
const request = require('supertest');
const mongoose = require('mongoose');
const app = require('../../app');
const User = require('../../models/User');

beforeAll(async () => {
  await mongoose.connect(process.env.TEST_DB_URL);
});

beforeEach(async () => {
  await User.deleteMany({});
});

afterAll(async () => {
  await mongoose.connection.close();
});

describe('User Integration Tests', () => {
  test('should create and retrieve user', async () => {
    const userData = { name: 'John Doe', email: 'john@example.com' };

    // Create user
    const createResponse = await request(app)
      .post('/users')
      .send(userData)
      .expect(201);

    const userId = createResponse.body._id;

    // Retrieve user
    const getResponse = await request(app)
      .get(`/users/${userId}`)
      .expect(200);

    expect(getResponse.body.name).toBe(userData.name);
    expect(getResponse.body.email).toBe(userData.email);
  });
});
```

Deployment

Environment Configuration

```javascript
// config/config.js
require('dotenv').config();

module.exports = {
  port: process.env.PORT || 3000,
  mongodb: {
    url: process.env.MONGODB_URL || 'mongodb://localhost:27017/myapp'
  },
  jwt: {
    secret: process.env.JWT_SECRET || 'your-secret-key',
    expiresIn: process.env.JWT_EXPIRES_IN || '7d'
  },
  redis: {
    url: process.env.REDIS_URL || 'redis://localhost:6379'
  }
};
```

Production Setup

```javascript
// app.js
const express = require('express');
const compression = require('compression');
const morgan = require('morgan');

const app = express();

// Production middleware
if (process.env.NODE_ENV === 'production') {
  app.use(compression());
  app.use(morgan('combined'));
} else {
  app.use(morgan('dev'));
}

// Graceful shutdown
process.on('SIGTERM', () => {
  console.log('SIGTERM received. Shutting down gracefully...');
  server.close(() => {
    console.log('Process terminated');
  });
});

process.on('SIGINT', () => {
  console.log('SIGINT received. Shutting down gracefully...');
  server.close(() => {
    console.log('Process terminated');
  });
});
```

Docker Configuration

dockerfile

```dockerfile
# Dockerfile
FROM node:16-alpine

WORKDIR /app

COPY package*.json ./
RUN npm ci --only=production

COPY . .

EXPOSE 3000

USER node

CMD ["npm", "start"]
```

yaml

```yaml
# docker-compose.yml
version: '3.8'
services:
  app:
    build: .
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
      - MONGODB_URL=mongodb://mongo:27017/myapp
    depends_on:
      - mongo

  mongo:
    image: mongo:4.4
    volumes:
      - mongo_data:/data/db
    ports:
      - "27017:27017"

volumes:
  mongo_data:
```

Performance Optimization

Caching with Redis

```bash
npm install redis
```

```javascript
const redis = require('redis');
const client = redis.createClient({
  url: process.env.REDIS_URL
});

client.on('error', (err) => console.log('Redis Client Error', err));
client.connect();

// Cache middleware
const cache = (duration = 300) => {
  return async (req, res, next) => {
    const key = req.originalUrl;

    try {
      const cached = await client.get(key);
      if (cached) {
        return res.json(JSON.parse(cached));
      }

      // Store original res.json
      const originalJson = res.json;

      // Override res.json
      res.json = function(data) {
        // Cache the response
        client.setEx(key, duration, JSON.stringify(data));
        // Call original json method
        originalJson.call(this, data);
      };

      next();
    } catch (error) {
      console.error('Cache error:', error);
      next();
    }
  };
};

// Usage
app.get('/users', cache(600), userController.getUsers); // Cache for 10 minutes
```

Database Optimization

javascript

```javascript
// MongoDB optimization
const userSchema = new mongoose.Schema({
  name: { type: String, required: true, index: true },
  email: { type: String, required: true, unique: true },
  status: { type: String, enum: ['active', 'inactive'], index: true },
  createdAt: { type: Date, default: Date.now, index: true }
});

// Compound index
userSchema.index({ status: 1, createdAt: -1 });

// Text search index
userSchema.index({ name: 'text', email: 'text' });

// Optimized queries
exports.getUsers = async (req, res) => {
  try {
    const { page = 1, limit = 10, status, search } = req.query;
    const skip = (page - 1) * limit;

    let query = {};

    if (status) {
      query.status = status;
    }

    if (search) {
      query.$text = { $search: search };
    }

    const users = await User.find(query)
      .select('name email status createdAt') // Only select needed fields
      .skip(skip)
      .limit(parseInt(limit))
      .sort({ createdAt: -1 });

    const total = await User.countDocuments(query);

    res.json({
      users,
      pagination: {
        page: parseInt(page),
        limit: parseInt(limit),
        total,
        pages: Math.ceil(total / limit)
      }
    });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};
```

Request Optimization

javascript

```javascript
// Compression middleware
const compression = require('compression');
app.use(compression());

// Response time tracking
app.use((req, res, next) => {
  const start = Date.now();

  res.on('finish', () => {
    const duration = Date.now() - start;
    console.log(`${req.method} ${req.url} - ${duration}ms`);
  });

  next();
});

// Pagination helper
const paginate = (model) => {
  return async (req, res, next) => {
    const page = parseInt(req.query.page) || 1;
    const limit = parseInt(req.query.limit) || 10;
    const skip = (page - 1) * limit;

    req.pagination = { page, limit, skip };

    // Add pagination info to response
    const originalJson = res.json;
    res.json = function(data) {
      if (data && Array.isArray(data)) {
        originalJson.call(this, {
          data,
          pagination: {
            page,
            limit,
            hasNext: data.length === limit,
            hasPrev: page > 1
          }
        });
      } else {
        originalJson.call(this, data);
      }
    };

    next();
    };
  };
```

Common Patterns

Repository Pattern

javascript

```javascript
// repositories/UserRepository.js
class UserRepository {
  constructor(model) {
    this.model = model;
  }

  async findAll(options = {}) {
    const { page = 1, limit = 10, filter = {} } = options;
    const skip = (page - 1) * limit;

    return await this.model.find(filter)
      .skip(skip)
      .limit(limit)
      .sort({ createdAt: -1 });
  }

  async findById(id) {
    return await this.model.findById(id);
  }

  async create(data) {
    const document = new this.model(data);
    return await document.save();
  }

  async update(id, data) {
    return await this.model.findByIdAndUpdate(id, data, {
      new: true,
      runValidators: true
    });
  }

  async delete(id) {
    return await this.model.findByIdAndDelete(id);
  }

  async count(filter = {}) {
    return await this.model.countDocuments(filter);
  }
}

module.exports = UserRepository;

// Usage in controller
const UserRepository = require('../repositories/UserRepository');
const User = require('../models/User');
const userRepo = new UserRepository(User);

exports.getUsers = async (req, res) => {
  try {
    const options = {
      page: parseInt(req.query.page) || 1,
      limit: parseInt(req.query.limit) || 10,
      filter: req.query.status ? { status: req.query.status } : {}
    };

    const users = await userRepo.findAll(options);
    const total = await userRepo.count(options.filter);

    res.json({
      users,
      pagination: {
        page: options.page,
        limit: options.limit,
        total,
        pages: Math.ceil(total / options.limit)
      }
    });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};
```

Service Layer Pattern

```
javascript
```

```javascript
// services/UserService.js
const UserRepository = require('../repositories/UserRepository');
const EmailService = require('./EmailService');
const bcrypt = require('bcryptjs');

class UserService {
  constructor() {
    this.userRepo = new UserRepository(require('../models/User'));
    this.emailService = new EmailService();
  }

  async createUser(userData) {
    // Validate email uniqueness
    const existingUser = await this.userRepo.findByEmail(userData.email);
    if (existingUser) {
      throw new Error('Email already exists');
    }

    // Hash password
    if (userData.password) {
      userData.password = await bcrypt.hash(userData.password, 10);
    }

    // Create user
    const user = await this.userRepo.create(userData);

    // Send welcome email
    await this.emailService.sendWelcomeEmail(user.email, user.name);

    return user;
  }

  async updateUser(id, updateData) {
    const user = await this.userRepo.findById(id);
    if (!user) {
      throw new Error('User not found');
    }

    // Hash password if provided
    if (updateData.password) {
      updateData.password = await bcrypt.hash(updateData.password, 10);
    }

    return await this.userRepo.update(id, updateData);
  }
  async deleteUser(id) {
    const user = await this.userRepo.findById(id);
    if (!user) {
      throw new Error('User not found');
    }

    // Send goodbye email
    await this.emailService.sendGoodbyeEmail(user.email, user.name);

    return await this.userRepo.delete(id);
  }
}

module.exports = UserService;

// Usage in controller
const UserService = require('../services/UserService');

const userService = new UserService();

exports.createUser = async (req, res) => {
  try {
    const user = await userService.createUser(req.body);
    res.status(201).json(user);
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
};
```

Dependency Injection Pattern

javascript

```
// di/Container.js
class Container {
  constructor() {
    this.services = new Map();
    this.singletons = new Map();
  }

  register(name, factory, options = {}) {
    this.services.set(name, { factory, options });
  }

  get(name) {
    const service = this.services.get(name);
    if (!service) {
      throw new Error(`Service ${name} not found`);
    }

    if (service.options.singleton) {
      if (!this.singletons.has(name)) {
        this.singletons.set(name, service.factory(this));
      }
      return this.singletons.get(name);
    }

    return service.factory(this);
  }
}

// di/setup.js
const Container = require('./Container');
const UserRepository = require('../repositories/UserRepository');
const UserService = require('../services/UserService');
const User = require('../models/User');

const container = new Container();

// Register services
container.register('userModel', () => User);
container.register('userRepository', (c) => new UserRepository(c.get('userModel')));
container.register('userService', (c) => new UserService(c.get('userRepository')), { singleton:

module.exports = container;

// Usage in routes
const container = require('../di/setup');
```

```
exports.createUser = async (req, res) => {
  try {
    const userService = container.get('userService');
    const user = await userService.createUser(req.body);
    res.status(201).json(user);
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
};
```

Event-Driven Architecture

javascript

```javascript
// events/EventEmitter.js
const EventEmitter = require('events');

class AppEventEmitter extends EventEmitter {
  constructor() {
    super();
    this.setMaxListeners(20);
  }
}

const eventEmitter = new AppEventEmitter();

module.exports = eventEmitter;

// events/listeners/UserListeners.js
const eventEmitter = require('../EventEmitter');
const EmailService = require('../../services/EmailService');
const Logger = require('../../utils/Logger');

const emailService = new EmailService();
const logger = new Logger();

// User created event
eventEmitter.on('user.created', async (user) => {
  try {
    await emailService.sendWelcomeEmail(user.email, user.name);
    logger.info(`Welcome email sent to ${user.email}`);
  } catch (error) {
    logger.error(`Failed to send welcome email: ${error.message}`);
  }
});

// User updated event
eventEmitter.on('user.updated', async (user) => {
  try {
    logger.info(`User ${user.id} updated`);
    // Additional logic here
  } catch (error) {
    logger.error(`Error handling user update: ${error.message}`);
  }
});

// Usage in service
const eventEmitter = require('../events/EventEmitter');

class UserService {
  async createUser(userData) {
    const user = await this.userRepo.create(userData);

    // Emit event
    eventEmitter.emit('user.created', user);

    return user;
  }
}
```

API Versioning

javascript

```javascript
// v1/routes/users.js
const express = require('express');
const router = express.Router();

router.get('/', (req, res) => {
  res.json({ version: 'v1', message: 'Users list v1' });
});

module.exports = router;

// v2/routes/users.js
const express = require('express');
const router = express.Router();

router.get('/', (req, res) => {
  res.json({
    version: 'v2',
    data: { users: [] },
    meta: { total: 0, page: 1 }
  });
});

module.exports = router;

// app.js
const v1Routes = require('./v1/routes/users');
const v2Routes = require('./v2/routes/users');

app.use('/api/v1/users', v1Routes);
app.use('/api/v2/users', v2Routes);

// Version middleware
const versionMiddleware = (req, res, next) => {
  const version = req.headers['api-version'] || req.query.version || 'v1';
  req.apiVersion = version;
  next();
};

app.use('/api', versionMiddleware);
```

WebSocket Integration

```bash
npm install socket.io
```

```
javascript
```

```
// websocket/socket.js
const socketIo = require('socket.io');

const initializeSocket = (server) => {
  const io = socketIo(server, {
    cors: {
      origin: process.env.FRONTEND_URL,
      methods: ["GET", "POST"]
    }
  });

  // Authentication middleware
  io.use((socket, next) => {
    const token = socket.handshake.auth.token;
    // Verify token
    next();
  });

  io.on('connection', (socket) => {
    console.log(`User connected: ${socket.id}`);

    socket.on('join-room', (room) => {
      socket.join(room);
      socket.to(room).emit('user-joined', socket.id);
    });

    socket.on('send-message', (data) => {
      socket.to(data.room).emit('receive-message', {
        message: data.message,
        sender: socket.id,
        timestamp: new Date()
      });
    });

    socket.on('disconnect', () => {
      console.log(`User disconnected: ${socket.id}`);
    });
  });

  return io;
};

module.exports = initializeSocket;

// app.js
const http = require('http');
const initializeSocket = require('./websocket/socket');

const server = http.createServer(app);
const io = initializeSocket(server);

// Make io available in routes
app.set('io', io);

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

Advanced Topics

GraphQL Integration

```bash
npm install apollo-server-express graphql
```

```
javascript
```

```javascript
// graphql/schema.js
const { gql } = require('apollo-server-express');

const typeDefs = gql`
  type User {
    id: ID!
    name: String!
    email: String!
    createdAt: String!
  }

  type Query {
    users: [User!]!
    user(id: ID!): User
  }

  type Mutation {
    createUser(name: String!, email: String!): User!
    updateUser(id: ID!, name: String, email: String): User!
    deleteUser(id: ID!): Boolean!
  }
`;

module.exports = typeDefs;

// graphql/resolvers.js
const User = require('../models/User');

const resolvers = {
  Query: {
    users: async () => {
      return await User.find();
    },
    user: async (_, { id }) => {
      return await User.findById(id);
    }
  },

  Mutation: {
    createUser: async (_, { name, email }) => {
      const user = new User({ name, email });
      return await user.save();
    },
    updateUser: async (_, { id, name, email }) => {
      return await User.findByIdAndUpdate(
        id,
        { name, email },
        { new: true }
      );
    },
    deleteUser: async (_, { id }) => {
      await User.findByIdAndDelete(id);
      return true;
    }
  }
};

module.exports = resolvers;

// app.js
const { ApolloServer } = require('apollo-server-express');
const typeDefs = require('./graphql/schema');
const resolvers = require('./graphql/resolvers');

const server = new ApolloServer({
  typeDefs,
  resolvers,
  context: ({ req }) => ({
    user: req.user // Pass authenticated user
  })
});

await server.start();
server.applyMiddleware({ app, path: '/graphql' });
```

Microservices Architecture

```
javascript
```

```javascript
// services/user-service/app.js
const express = require('express');
const axios = require('axios');

const app = express();
app.use(express.json());

// Service discovery
const services = {
  'notification-service': process.env.NOTIFICATION_SERVICE_URL,
  'auth-service': process.env.AUTH_SERVICE_URL
};

// Inter-service communication
const callService = async (service, endpoint, data) => {
  try {
    const response = await axios.post(`${services[service]}${endpoint}`, data);
    return response.data;
  } catch (error) {
    console.error(`Error calling ${service}:`, error.message);
    throw error;
  }
};

app.post('/users', async (req, res) => {
  try {
    // Create user
    const user = await User.create(req.body);

    // Call notification service
    await callService('notification-service', '/send-welcome', {
      email: user.email,
      name: user.name
    });

    res.status(201).json(user);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// Health check
app.get('/health', (req, res) => {
  res.json({ status: 'healthy', service: 'user-service' });
});
```

```
module.exports = app;
```

Message Queues with Bull

bash

```bash
npm install bull redis
```

```
javascript
```

```
// queues/emailQueue.js
const Queue = require('bull');
const redis = require('redis');

const emailQueue = new Queue('email processing', {
  redis: {
    port: process.env.REDIS_PORT,
    host: process.env.REDIS_HOST,
  }
});

// Process jobs
emailQueue.process('welcome-email', async (job) => {
  const { email, name } = job.data;

  // Send email logic here
  console.log(`Sending welcome email to ${email}`);

  // Simulate email sending
  await new Promise(resolve => setTimeout(resolve, 2000));

  return { status: 'sent', email };
});

// Add job to queue
const sendWelcomeEmail = async (email, name) => {
  await emailQueue.add('welcome-email', { email, name }, {
    attempts: 3,
    backoff: 'exponential',
    delay: 5000
  });
};

module.exports = { emailQueue, sendWelcomeEmail };

// Usage in service
const { sendWelcomeEmail } = require('../queues/emailQueue');

class UserService {
  async createUser(userData) {
    const user = await this.userRepo.create(userData);

    // Queue welcome email
    await sendWelcomeEmail(user.email, user.name);

    return user;
    }
  }
```

Best Practices Summary

Code Organization

- Use MVC or layered architecture
- Separate concerns (routes, controllers, services, repositories)
- Use dependency injection for better testability
- Implement proper error handling
- Use environment variables for configuration

Security

- Always validate and sanitize input
- Use HTTPS in production
- Implement proper authentication and authorization
- Use security headers (helmet)
- Rate limit API endpoints
- Keep dependencies updated

Performance

- Use caching strategies (Redis, in-memory)
- Optimize database queries
- Implement pagination
- Use compression middleware
- Monitor and profile your application

Testing

- Write unit tests for business logic
- Write integration tests for API endpoints
- Use test databases
- Mock external dependencies
- Maintain good test coverage

Deployment

- Use environment-specific configurations

- Implement proper logging
- Set up monitoring and alerting
- Use process managers (PM2)
- Implement graceful shutdowns
- Use Docker for containerization

This playbook covers the essential aspects of Express.js development from basic concepts to advanced patterns. Use it as a reference guide and adapt the examples to your specific use cases.