MERN Stack with Next.js Portfolio Project Structure Guide

Introduction

This comprehensive guide provides a production-ready project structure for building a modern portfolio website using the MERN stack (MongoDB, Express, React/Next.js, Node.js). This architecture follows industry best practices for scalability, maintainability, and deployment[1][2].

Complete Project Structure

Root Directory Overview

```
/portfolio-website
├── /client # Next.js Frontend (Port 3000)
├── /server # Express Backend (Port 5000)
├── .gitignore # Git ignore configuration
├── README.md # Project documentation
└── package.json # Optional root package.json
```

Frontend Structure (Next.js 14/15)

Client Directory - Full Structure

```
/client
├── /public # Static assets
│ ├── /images
│ │ ├── profile.jpg
│ │ ├── /projects # Project screenshots
│ │ │ ├── project1.png
│ │ │ ├── project2.png
│ │ │ └── project3.png
│ │ └── /icons # Icon assets
│ │ ├── github.svg
│ │ ├── linkedin.svg
│ │ └── email.svg
│ ├── /assets
│ │ ├── resume.pdf # Downloadable resume
│ │ └── /certificates # Certificate PDFs
│ ├── favicon.ico
│ └── robots.txt # SEO configuration
│
├── /src
│ ├── /app # Next.js 13+ App Router
│ │ ├── layout.js # Root layout (Navbar, Footer)
```

```
│   │   ├── page.js # Homepage
│   │   ├── globals.css # Global styles
│   │   ├── loading.js # Loading UI
│   │   ├── error.js # Error handling
│   │   │
│   │   ├── /about
│   │   │   └── page.js # About page
│   │   │
│   │   ├── /projects
│   │   │   ├── page.js # Projects listing
│   │   │   └── /[id] # Dynamic routes
│   │   │   └── page.js # Individual project page
│   │   │
│   │   ├── /experience
│   │   │   └── page.js # Work experience page
│   │   │
│   │   ├── /skills
│   │   │   └── page.js # Skills showcase
│   │   │
│   │   ├── /contact
│   │   │   └── page.js # Contact form page
│   │   │
│   │   ├── /blog # Optional blog section
│   │   │   ├── page.js
│   │   │   └── /[slug]
│   │   │   └── page.js
│   │   │
│   │   └── /api # API routes (optional)
│   │   └── /contact
│   │   └── route.js # Contact form handler
│   │
│   ├── /components # Reusable UI components
│   │   ├── /layout
│   │   │   ├── Header.jsx # Site header
│   │   │   ├── Footer.jsx # Site footer
│   │   │   ├── Navbar.jsx # Navigation bar
│   │   │   └── Sidebar.jsx # Mobile sidebar
│   │   │
│   │   ├── /sections # Page sections
│   │   │   ├── Hero.jsx # Landing hero section
│   │   │   ├── About.jsx # About section
│   │   │   ├── Skills.jsx # Skills grid
│   │   │   ├── ProjectsSection.jsx # Projects showcase
│   │   │   ├── Experience.jsx # Timeline component
│   │   │   ├── Testimonials.jsx # Client testimonials
│   │   │   └── ContactSection.jsx # Contact CTA
│   │   │
│   │   ├── /ui # Basic UI components
│   │   │   ├── Button.jsx # Custom button
│   │   │   ├── Card.jsx # Card wrapper
│   │   │   ├── Modal.jsx # Modal dialog
```

```
│   │   │   ├── Input.jsx # Form input
│   │   │   ├── Badge.jsx # Skill badges
│   │   │   └── Loader.jsx # Loading spinner
│   │   │
│   │   └── /common # Domain components
│   │   ├── ProjectCard.jsx # Project display card
│   │   ├── SkillBadge.jsx # Technology badge
│   │   ├── Timeline.jsx # Experience timeline
│   │   ├── SocialLinks.jsx # Social media icons
│   │   └── ThemeToggle.jsx # Dark/Light mode toggle
│   │
│   ├── /lib # Utilities & API clients
│   │   ├── api.js # Backend API calls
│   │   ├── constants.js # App constants
│   │   ├── utils.js # Helper functions
│   │   └── validations.js # Form validations
│   │
│   ├── /styles # Additional styles
│   │   ├── variables.css # CSS variables
│   │   └── /components # Component styles
│   │   ├── hero.css
│   │   └── projects.css
│   │
│   ├── /hooks # Custom React hooks
│   │   ├── useProjects.js # Fetch projects data
│   │   ├── useTheme.js # Theme management
│   │   ├── useForm.js # Form handling
│   │   └── useScrollPosition.js # Scroll tracking
│   │
│   ├── /context # React Context API
│   │   ├── ThemeContext.js # Theme provider
│   │   └── AppContext.js # Global state
│   │
│   └── /types # TypeScript types (if TS)
│   ├── project.ts
│   ├── skill.ts
│   └── experience.ts
│
├── .env.local # Environment variables
├── .eslintrc.json # ESLint configuration
├── .prettierrc # Prettier configuration
├── next.config.js # Next.js configuration
├── tailwind.config.js # Tailwind CSS config
├── postcss.config.js # PostCSS config
├── jsconfig.json # JavaScript config
├── package.json # Dependencies
└── README.md # Frontend documentation
```

Key Frontend Features

- **App Router Structure**: Next.js 13+ uses the `/app` directory for file-based routing[1][3]
- **Component Organization**: Separated by purpose (layout, sections, UI, common)
- **API Integration**: Centralized in `/lib/api.js` for clean backend communication
- **Custom Hooks**: Reusable logic for data fetching and UI state
- **Responsive Design**: Built with Tailwind CSS for mobile-first approach

Backend Structure (Express + MongoDB)

Server Directory - Full Structure

```
/server
├── /src
│   ├── /models # Mongoose schemas
│   │   ├── Project.js # Project schema
│   │   ├── Skill.js # Skill schema
│   │   ├── Experience.js # Work experience schema
│   │   ├── Education.js # Education schema
│   │   ├── Contact.js # Contact message schema
│   │   ├── Blog.js # Blog post schema (optional)
│   │   └── User.js # Admin user schema (optional)
│   │
│   ├── /routes # API endpoints
│   │   ├── index.js # Route aggregator
│   │   ├── projectRoutes.js # /api/projects
│   │   ├── skillRoutes.js # /api/skills
│   │   ├── experienceRoutes.js # /api/experience
│   │   ├── educationRoutes.js # /api/education
│   │   ├── contactRoutes.js # /api/contact
│   │   ├── blogRoutes.js # /api/blog (optional)
│   │   └── authRoutes.js # /api/auth (optional)
│   │
│   ├── /controllers # Business logic
│   │   ├── projectController.js # Project CRUD operations
│   │   ├── skillController.js # Skill CRUD operations
│   │   ├── experienceController.js # Experience CRUD operations
│   │   ├── educationController.js # Education CRUD operations
│   │   ├── contactController.js # Contact form handling
│   │   ├── blogController.js # Blog operations (optional)
│   │   └── authController.js # Authentication (optional)
│   │
│   ├── /middleware # Custom middleware
│   │   ├── errorHandler.js # Global error handling
│   │   ├── validator.js # Request validation
│   │   ├── auth.js # JWT authentication
│   │   ├── cors.js # CORS configuration
│   │   └── logger.js # Request logging
│   │
│   ├── /config # Configuration files
│   │   ├── db.js # MongoDB connection
```

```
│   │   ├── config.js # App configuration
│   │   └── constants.js # Backend constants
│   │
│   ├── /utils # Utility functions
│   │   ├── emailService.js # Email sending (Nodemailer)
│   │   ├── logger.js # Winston logger
│   │   ├── fileUpload.js # Multer file handling
│   │   └── helpers.js # Helper functions
│   │
│   ├── /validators # Validation schemas
│   │   ├── projectValidator.js
│   │   ├── contactValidator.js
│   │   └── authValidator.js
│   │
│   └── server.js # Main server entry point
│
├── /uploads # Uploaded files (if any)
│   ├── /projects
│   └── /documents
│
├── .env # Environment variables
├── .env.example # Environment template
├── .gitignore # Git ignore
├── package.json # Dependencies
└── README.md # Backend documentation
```

Key Backend Features

- **MVC Architecture**: Models, Routes, Controllers separation[4][5]
- **Middleware Stack**: Error handling, validation, authentication
- **Database Layer**: Mongoose ODM for MongoDB operations
- **API Structure**: RESTful endpoints with proper HTTP methods
- **Security**: CORS, helmet, rate limiting, input validation

Environment Variables Configuration

Frontend (.env.local)

API Configuration

```
NEXT_PUBLIC_API_URL=http://localhost:5000/api
NEXT_PUBLIC_SITE_URL=https://tejaskamble.in
```

Analytics (Optional)

```
NEXT_PUBLIC_GA_ID=G-XXXXXXXXXX
```

Email Service (Optional)

NEXT_PUBLIC_EMAILJS_SERVICE_ID=your_service_id
NEXT_PUBLIC_EMAILJS_TEMPLATE_ID=your_template_id
NEXT_PUBLIC_EMAILJS_PUBLIC_KEY=your_public_key

Backend (.env)

Server Configuration

PORT=5000
NODE_ENV=development

Database

MONGODB_URI=mongodb://localhost:27017/portfolio

OR for MongoDB Atlas:

MONGODB_URI=mongodb+srv://username:password@cluster.mongodb.net/portfolio

Frontend URL

FRONTEND_URL=http://localhost:3000

JWT Authentication (if using admin panel)

JWT_SECRET=your_super_secret_jwt_key_here
JWT_EXPIRE=7d

Email Service (Nodemailer)

EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=587
EMAIL_USER=your_email@gmail.com
EMAIL_PASSWORD=your_app_password

File Upload

MAX_FILE_SIZE=5242880
ALLOWED_FILE_TYPES=image/jpeg,image/png,application/pdf

Rate Limiting

RATE_LIMIT_WINDOW=15
RATE_LIMIT_MAX_REQUESTS=100

Database Schema Design

Project Model Example

```
// server/src/models/Project.js
import mongoose from 'mongoose';

const projectSchema = new mongoose.Schema({
title: {
type: String,
required: true,
trim: true
},
description: {
type: String,
required: true
},
longDescription: String,
technologies: [{
type: String,
required: true
}],
githubUrl: String,
liveUrl: String,
images: [String],
featured: {
type: Boolean,
default: false
},
category: {
type: String,
enum: ['web', 'mobile', 'ai', 'cloud', 'other'],
default: 'web'
},
metrics: {
impact: String,
timeline: String,
team: String
},
```

```
order: {
type: Number,
default: 0
}
}, {
timestamps: true
});

export default mongoose.model('Project', projectSchema);
```

Contact Model Example

```
// server/src/models/Contact.js
import mongoose from 'mongoose';

const contactSchema = new mongoose.Schema({
name: {
type: String,
required: true,
trim: true
},
email: {
type: String,
required: true,
lowercase: true
},
subject: {
type: String,
required: true
},
message: {
type: String,
required: true
},
status: {
type: String,
enum: ['new', 'read', 'replied'],
default: 'new'
},
ipAddress: String
}, {
timestamps: true
});

export default mongoose.model('Contact', contactSchema);
```

API Endpoints Structure

RESTful API Routes

Method	Endpoint	Description
GET	/api/projects	Get all projects
GET	/api/projects/:id	Get single project
POST	/api/projects	Create project (admin)
PUT	/api/projects/:id	Update project (admin)
DELETE	/api/projects/:id	Delete project (admin)
GET	/api/skills	Get all skills
POST	/api/skills	Add skill (admin)
GET	/api/experience	Get work experience
POST	/api/experience	Add experience (admin)
POST	/api/contact	Submit contact form
GET	/api/contact	Get all messages (admin)
POST	/api/auth/login	Admin login
POST	/api/auth/logout	Admin logout

Table 1: API endpoint structure for portfolio backend

Installation and Setup Guide

Prerequisites

- Node.js (v18 or higher)
- MongoDB (local or Atlas)
- Git
- Code editor (VS Code recommended)

Step 1: Project Initialization

Create root directory

mkdir portfolio-website
cd portfolio-website

Initialize Git

git init

Create .gitignore

echo "node_modules/\n.env\n.env.local\n.DS_Store" > .gitignore

Step 2: Frontend Setup (Next.js)

Create Next.js app with Tailwind CSS

npx create-next-app@latest client

Follow prompts:

✔ Would you like to use TypeScript? No

✔ Would you like to use ESLint? Yes

✔ Would you like to use Tailwind CSS? Yes

✔ Would you like to use `src/` directory? Yes

✔ Would you like to use App Router? Yes

✔ Would you like to customize the default import alias? No

cd client

Install additional dependencies

npm install axios react-icons framer-motion

Development server

npm run dev

Runs on http://localhost:3000

Step 3: Backend Setup (Express + MongoDB)

Navigate to root

cd ..

Create server directory

mkdir server
cd server

Initialize Node.js project

npm init -y

Install dependencies

npm install express mongoose dotenv cors helmet express-rate-limit
npm install nodemon --save-dev

Create folder structure

mkdir -p src/{models,routes,controllers,middleware,config,utils,validators}

Create main server file

touch src/server.js

Create environment file

touch .env

Step 4: Configure package.json (Backend)

```
{
"name": "portfolio-backend",
"version": "1.0.0",
"type": "module",
"scripts": {
"start": "node src/server.js",
"dev": "nodemon src/server.js"
},
"dependencies": {
"express": "^4.18.2",
"mongoose": "^8.0.0",
"dotenv": "^16.3.1",
"cors": "^2.8.5",
"helmet": "^7.1.0",
"express-rate-limit": "^7.1.5"
},
"devDependencies": {
"nodemon": "^3.0.2"
}
}
```

Step 5: Basic Server Setup

```
// server/src/server.js
import express from 'express';
import dotenv from 'dotenv';
import cors from 'cors';
import helmet from 'helmet';
import connectDB from './config/db.js';

dotenv.config();

const app = express();
const PORT = process.env.PORT || 5000;

// Database connection
connectDB();

// Middleware
app.use(helmet());
app.use(cors({
origin: process.env.FRONTEND_URL,
credentials: true
}));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

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

```
res.json({ message: 'Portfolio API Running' });
});

// Error handling
app.use((err, req, res, next) => {
console.error(err.stack);
res.status(500).json({
success: false,
message: 'Something went wrong!'
});
});

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

Step 6: Database Connection

```
// server/src/config/db.js
import mongoose from 'mongoose';

const connectDB = async () => {
try {
const conn = await mongoose.connect(process.env.MONGODB_URI);
console.log(MongoDB Connected: ${conn.connection.host});
} catch (error) {
console.error(Error: ${error.message});
process.exit(1);
}
};

export default connectDB;
```

Development Workflow

Running Both Servers

Terminal 1 - Frontend (from /client)

```
cd client
npm run dev
```

Runs on http://localhost:3000

Terminal 2 - Backend (from /server)

cd server
npm run dev

Runs on http://localhost:5000

Git Workflow

Initialize repository

git add .
git commit -m "Initial project structure"

Create GitHub repository and push

git remote add origin https://github.com/yourusername/portfolio.git
git branch -M main
git push -u origin main

Deployment Strategy

Frontend Deployment (Vercel)

1. Push code to GitHub
2. Connect repository to Vercel
3. Configure environment variables
4. Deploy automatically on push to main branch

Backend Deployment (Railway/Render)

1. Push server code to GitHub
2. Create new project on Railway/Render
3. Connect MongoDB Atlas database
4. Configure environment variables
5. Deploy and get production URL

Database Setup (MongoDB Atlas)

1. Create account at mongodb.com/cloud/atlas
2. Create free cluster
3. Add database user
4. Whitelist IP addresses (0.0.0.0/0 for all)
5. Get connection string
6. Add to .env file

Best Practices and Tips

Code Organization

- Keep components small and focused (single responsibility)
- Use meaningful names for files and folders
- Group related files together
- Separate business logic from UI components
- Use constants file for magic numbers and strings

Performance Optimization

- Optimize images (use WebP format, lazy loading)
- Implement code splitting in Next.js
- Use React.memo for expensive components
- Minimize bundle size (analyze with webpack-bundle-analyzer)
- Enable caching for API responses

Security Considerations

- Never commit .env files to Git
- Use environment variables for sensitive data
- Implement rate limiting on API endpoints
- Validate and sanitize all user inputs
- Use HTTPS in production
- Implement CORS properly

Testing Strategy

- Write unit tests for utility functions
- Test API endpoints with Postman or Jest
- Implement integration tests for critical flows
- Use React Testing Library for component tests
- Perform manual testing on different devices

Common Pitfalls to Avoid

1. Not separating frontend and backend properly
2. Hardcoding API URLs instead of using environment variables
3. Not implementing proper error handling
4. Forgetting to add .env files to .gitignore
5. Not optimizing images before deployment
6. Skipping mobile responsive testing
7. Not implementing loading states
8. Poor folder organization leading to spaghetti code

Useful Commands Reference

Frontend Commands

npm run dev # Start development server
npm run build # Create production build
npm run start # Start production server
npm run lint # Run ESLint

Backend Commands

npm run dev # Start with nodemon
npm start # Start production server
npm test # Run tests

Database Commands

mongosh # Open MongoDB shell
mongodump # Backup database
mongorestore # Restore database

Git Commands

git status # Check status
git add . # Stage all changes
git commit -m "msg" # Commit changes
git push # Push to remote

Additional Resources and Learning

Official Documentation

- Next.js Documentation: https://nextjs.org/docs
- Express.js Guide: https://expressjs.com
- MongoDB Manual: https://docs.mongodb.com
- Mongoose Docs: https://mongoosejs.com/docs
- Tailwind CSS: https://tailwindcss.com/docs

Recommended Tools

- VS Code Extensions: ESLint, Prettier, ES7 React snippets
- API Testing: Postman, Insomnia
- Database GUI: MongoDB Compass
- Version Control: GitHub Desktop or GitKraken
- Deployment: Vercel (frontend), Railway/Render (backend)

Conclusion

This project structure provides a solid foundation for building a professional, scalable portfolio website using the MERN stack with Next.js. The separation of concerns between frontend and backend allows for independent development, testing, and deployment. Following this structure and the best practices outlined will help you create a maintainable codebase that can grow with your needs.

Remember to start simple and add complexity as needed. Focus on getting the core functionality working first, then enhance with additional features like admin panels, blogs, or AI chatbots as you become more comfortable with the stack.

Quick Start Checklist

- [ ] Install Node.js and MongoDB
- [ ] Create project root directory
- [ ] Set up Next.js frontend with Tailwind CSS
- [ ] Set up Express backend with basic structure
- [ ] Configure environment variables
- [ ] Create basic models and routes
- [ ] Test API endpoints with Postman
- [ ] Connect frontend to backend
- [ ] Build and deploy to production
- [ ] Set up continuous deployment

References

[1] Next.js Documentation. (2024). Project Structure and File Conventions. https://nextjs.org/docs/app/getting-started/project-structure

[2] Strapi. (2025). MERN Stack Guide: Components, Setup & Best Practices. https://strapi.io/blog/mern-stack-guide-components-setup-best-practices

[3] Dev.to. (2025). Best Practices for Organizing Your Next.js 15. https://dev.to/bajrayejoon/best-practices-for-organizing-your-nextjs-15-2025-53ji

[4] Fadamakis, F. (2023). Express + Mongo Application Architecture and Folder Structure. Medium. https://fadamakis.com/express-mongo-application-architecture-and-folder-structure-1f95274c28fe

[5] freeCodeCamp. (2025). Intro to Backend Web Development – Node.js, Express, MongoDB. https://www.freecodecamp.org/news/intro-to-backend-web-development-nodejs-express-mongodb/