Advanced MERN Stack Project Setup Guide from Scratch

Building a Production-Ready MERN Application with Next.js, GitLab, and npm

Created by Praveen Kumar
May 2025

Table of Contents

Introduction

This guide provides a comprehensive, step-by-step process to build a scalable, production-ready MERN (MongoDB, Express.js, React via Next.js, Node.js) stack project from scratch. It covers:

- Setting up development tools (VS Code, Git Bash, Postman).
- Configuring version control with GitLab.
- Building a Next.js frontend with Tailwind CSS.
- Creating an Express.js backend with MongoDB.
- Implementing JWT authentication and a todo feature.
- Testing, CI/CD with GitLab CI, and deployment to Vercel/Render.
- Using npm as the package manager.

The project includes a sample todo app with user authentication, making it ideal for beginners and intermediate developers. Git Bash is emphasized for Git operations, especially for Windows users.

Prerequisites

- Basic knowledge of JavaScript, HTML, CSS, and Git.
- A computer with internet access (Windows, macOS, or Linux).
- A GitLab account (Sign Up).
- A MongoDB Atlas account (optional, Register).

1. Installing Visual Studio Code (VS Code)

VS Code is a lightweight, customizable code editor for full-stack development.

- **Download**: https://code.visualstudio.com/
- **Steps**:
  1. Download the latest version for your OS (Windows, macOS, Linux).
  2. Run the installer and follow the prompts.
  3. Launch VS Code.
- **Recommended Extensions**:
  - Prettier: Code formatting.
  - ESLint: JavaScript linting.
  - GitLens: Enhanced Git integration.
  - JavaScript (ES6) Code Snippets: Quick snippets for JavaScript/React.
- **Configuration**:
  1. Open Extensions view (`Ctrl+Shift+X` or `Cmd+Shift+X` on macOS).
  2. Install the listed extensions.
  3. Enable format on save: `File > Preferences > Settings`, search for "format on save," and check the box.
  4. Set Prettier as the default formatter: Search for "default formatter" and select "Prettier."

2. Installing Git and Git Bash

Git is essential for version control, and Git Bash provides a Unix-like terminal for Windows users.

- **Download**: https://git-scm.com/downloads
- **Steps**:
  1. Download the Git installer for your OS.
  2. Run the installer, accepting default settings (ensure "Git Bash" is included for Windows).
  3. Complete the setup.
- **Verify Installation**:

```
git --version
```

Expected output: `git version 2.x.x`.

- **Configure Git**:

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

3. Setting Up GitLab Account

GitLab hosts your code and manages version control.

- **Link**: https://gitlab.com/users/sign_in
- **Steps**:
  1. Sign up or log in at GitLab.
  2. Create a new project:
     - Click *New Project* on the dashboard.
     - Name it (e.g., `mern-project`), set visibility (public/private), and create.
     - Copy the repository URL (e.g., `https://gitlab.com/your-username/mern-project.git`).
- **Set Up SSH (Recommended)**:
  1. Generate an SSH key:

     ```
     ssh-keygen -t rsa -b 4096 -C "your.email@example.com"
     cat ~/.ssh/id_rsa.pub
     ```

  2. Copy the public key and add it to GitLab (*Preferences* > *SSH Keys*).
  3. Verify SSH:

     ```
     ssh -T git@gitlab.com
     ```

     Expected output: `Welcome to GitLab, @your-username!`.

4. Installing Node.js and npm

Node.js is the runtime for JavaScript, and npm is the package manager.

- **Download**: https://nodejs.org/
- **Steps**:
  1. Download the LTS version (e.g., 20.x.x) for your OS.
  2. Run the installer and follow the prompts.
- **Verify Installation**:

```
node -v
npm -v
```

Expected output: `v20.x.x` (Node.js), `10.x.x` (npm).

- **What is npm?**:

- npm (Node Package Manager) is the default package manager for Node.js.
- It manages JavaScript packages (libraries/modules).
- Common commands:

```
npm install <package-name>   # Install a specific package
npm install                  # Install all dependencies listed in
package.json
npm uninstall <package-name> # Remove a package
```

5. Installing MongoDB

MongoDB is the NoSQL database for the MERN stack.

- **Download**: https://www.mongodb.com/try/download/community
- **Option 1: Local MongoDB**:
  1. Download and install MongoDB Community Server.
  2. Start the server:

     ```
     mongod
     ```

  3. Install MongoDB Compass (GUI): https://www.mongodb.com/try/download/compass
- **Option 2: MongoDB Atlas (Cloud)**:
  1. Sign up: https://www.mongodb.com/cloud/atlas/register
  2. Create a free cluster, whitelist your IP, and get the connection string (e.g., `mongodb+srv://<username>:<password>@cluster0.mongodb.net/<dbname>`).
- **Verify Connection**:
  - Use MongoDB Compass or the `mongo` shell to connect.

6. Installing Postman

Postman is used for testing RESTful APIs.

- **Download**: https://www.postman.com/downloads/
- **Steps**:
  1. Download and install Postman for your OS.
  2. Sign in (optional) to save workspaces.
- **Usage**:
  - Create GET, POST, PUT, DELETE requests to test APIs.
  - Save requests in collections for reuse.

7. Setting Up the Project Folder

Create a project folder to organize your MERN stack application.

```
mkdir mern-project
cd mern-project
```

- **Initialize Git**:

  ```
  git init
  echo "node_modules/" > .gitignore
  echo ".env" >> .gitignore
  ```

- **Project Structure**:

  ```
  mern-project/
  ├── client/       # Next.js frontend
  ├── server/       # Express.js backend
  ├── .gitignore
  ```

8. Frontend: Using Next.js (React Framework)

Next.js is a React framework for server-side rendering and static site generation.

```
npx create-next-app@latest client
cd client
npm run dev
```

- Ensure the app runs at `http://localhost:3000`.
- **Install Dependencies**:

  ```
  npm install axios
  ```

  - `axios`: For API requests.
- **Create a Sample Page**: Replace `client/app/page.js`:

  ```
  import axios from 'axios';
  import { useState, useEffect } from 'react';

  export default function Home() {
    const [message, setMessage] = useState('');

    useEffect(() => {
      axios.get('http://localhost:5000/api/sample')
        .then((res) => setMessage(res.data))
        .catch((err) => console.error(err));
    }, []);

    return (
      <div className="p-6">
        <h1 className="text-3xl font-bold">MERN Stack App</h1>
        <p className="mt-4">Backend Message: {message}</p>
      </div>
    );
  }
  ```

9. Installing Tailwind CSS in Next.js

Tailwind CSS is a utility-first CSS framework for styling.

```
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

- **Update** `client/tailwind.config.js`:

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    "./pages/**/*.{js,ts,jsx,tsx}",
    "./components/**/*.{js,ts,jsx,tsx}",
    "./app/**/*.{js,ts,jsx,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

- **Update** `client/app/globals.css`:

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

10. Backend: Setting Up Node.js with Express

The backend handles API logic and database interactions.

```
mkdir server
cd server
npm init -y
```

- **Install Dependencies**:

```
npm install express mongoose cors dotenv nodemon jsonwebtoken bcryptjs
npm install --save-dev jest supertest
```

  - `jsonwebtoken`, `bcryptjs`: For authentication.
  - `jest`, `supertest`: For testing.

- **Update** `server/package.json`:

```
{
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js",
    "test": "jest"
```

```
  }
}
```

11. Connecting MongoDB to Backend

Set up the Express server and connect to MongoDB.

- **Create** `server/index.js` :

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
require('dotenv').config();

const sampleRoutes = require('./routes/sample');
const authRoutes = require('./routes/auth');
const todoRoutes = require('./routes/todos');

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

app.use(cors());
app.use(express.json());

// MongoDB Connection
mongoose.connect(process.env.MONGO_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
  .then(() => console.log('MongoDB Connected'))
  .catch((err) => console.error(err));

// Routes
app.use('/api/sample', sampleRoutes);
app.use('/api/auth', authRoutes);
app.use('/api/todos', todoRoutes);

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

- **Create** `server/.env` :

```
MONGO_URI=your_mongodb_connection_string
PORT=5000
JWT_SECRET=your_jwt_secret
```

12. Sample API Setup

Create a sample API to test the backend.

- **Create** `server/routes/sample.js` :

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

router.get('/', (req, res) => {
  res.send('Hello from sample API!');
});

module.exports = router;
```

13. Implement Authentication

Add user registration and login with JWT.

13.1 User Model

- **Create** `server/models/User.js`:

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

const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true },
});

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

13.2 Auth Routes

- **Create** `server/routes/auth.js`:

```
const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const User = require('../models/User');

// Register
router.post('/register', async (req, res) => {
  const { username, password } = req.body;
  try {
    let user = await User.findOne({ username });
    if (user) return res.status(400).json({ msg: 'User already exists' });

    user = new User({ username, password: await bcrypt.hash(password, 10) });
    await user.save();

    const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET, {
expiresIn: '1h' });
    res.json({ token });
  } catch (err) {
    res.status(500).json({ msg: 'Server error' });
  }
```

```
});

// Login
router.post('/login', async (req, res) => {
  const { username, password } = req.body;
  try {
    const user = await User.findOne({ username });
    if (!user) return res.status(400).json({ msg: 'Invalid credentials' });

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).json({ msg: 'Invalid credentials' });

    const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET, {
expiresIn: '1h' });
    res.json({ token });
  } catch (err) {
    res.status(500).json({ msg: 'Server error' });
  }
});

module.exports = router;
```

13.3 Middleware for Protected Routes

- **Create** `server/middleware/auth.js` :

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

module.exports = async (req, res, next) => {
  const token = req.header('Authorization')?.replace('Bearer ', '');
  if (!token) return res.status(401).json({ msg: 'No token, authorization
denied' });

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    res.status(401).json({ msg: 'Token is not valid' });
  }
};
```

14. Build a Todo Feature (CRUD with Authentication)

Implement a todo feature to demonstrate full-stack integration.

14.1 Todo Model

- **Create** `server/models/Todo.js` :

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

```
const todoSchema = new mongoose.Schema({
  text: { type: String, required: true },
  completed: { type: Boolean, default: false },
  user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
});

module.exports = mongoose.model('Todo', todoSchema);
```

14.2 Todo Routes

- **Create** `server/routes/todos.js`:

```
const express = require('express');
const router = express.Router();
const auth = require('../middleware/auth');
const Todo = require('../models/Todo');

// Get all todos for user
router.get('/', auth, async (req, res) => {
  const todos = await Todo.find({ user: req.user.id });
  res.json(todos);
});

// Create a todo
router.post('/', auth, async (req, res) => {
  const { text } = req.body;
  const todo = new Todo({ text, user: req.user.id });
  await todo.save();
  res.json(todo);
});

module.exports = router;
```

14.3 Frontend Integration

- **Update** `client/app/page.js`:

```
'use client';
import axios from 'axios';
import { useState, useEffect } from 'react';

export default function Home() {
  const [todos, setTodos] = useState([]);
  const [text, setText] = useState('');
  const [token, setToken] = useState('');

  const login = async () => {
    const res = await axios.post('http://localhost:5000/api/auth/login', {
      username: 'testuser',
      password: 'password123',
    });
    setToken(res.data.token);
```

```
  };

  // Fetch todos
  useEffect(() => {
    if (token) {
      axios.get('http://localhost:5000/api/todos', {
        headers: { Authorization: `Bearer ${token}` },
      })
        .then((res) => setTodos(res.data))
        .catch((err) => console.error(err));
    }
  }, [token]);

  // Add todo
  const addTodo = async () => {
    const res = await axios.post(
      'http://localhost:5000/api/todos',
      { text },
      { headers: { Authorization: `Bearer ${token}` } }
    );
    setTodos([...todos, res.data]);
    setText('');
  };

  return (
    <div className="p-6 max-w-md mx-auto">
      <h1 className="text-3xl font-bold mb-4">Todo App</h1>
      <button onClick={login} className="bg-blue-500 text-white p-2 rounded mb-
4">
        Login (Test)
      </button>
      <div className="flex gap-2 mb-4">
        <input
          type="text"
          value={text}
          onChange={(e) => setText(e.target.value)}
          className="border p-2 rounded flex-1"
          placeholder="Enter todo"
        />
        <button onClick={addTodo} className="bg-green-500 text-white p-2
rounded">
          Add
        </button>
      </div>
      <ul className="space-y-2">
        {todos.map((todo) => (
          <li key={todo._id} className="border p-2 rounded">
            {todo.text}
          </li>
        ))}
      </ul>
    </div>
```

```
  );
}
```

14.4 Test the Feature

1. Start the backend: `npm run dev` in `server`.
2. Start the frontend: `npm run dev` in `client`.
3. Register a user via Postman (`POST /api/auth/register`), log in, and test the todo feature.

15. Git Commands to Push to GitLab

Push your project to GitLab using Git Bash for version control.

```
git init
git remote add origin https://gitlab.com/your-username/mern-project.git
git add .
git commit -m "Initial commit"
git branch -M main
git push -u origin main
```

- **Using SSH (Preferred)**:

```
git remote add origin git@gitlab.com:your-username/mern-project.git
git add .
git commit -m "Initial commit"
git branch -M main
git push -u origin main
```

- **Handle Authentication**:
  - For HTTPS, use a Personal Access Token:
    - Go to GitLab *Preferences* > *Access Tokens*.
    - Create a token with `api` and `write_repository` scopes.
    - Use the token as your password.
  - For SSH, ensure your SSH key is set up (see Section 3).
- **Verify Push**:
  - Visit your GitLab repository (`https://gitlab.com/your-username/mern-project`).
  - Confirm all files are uploaded.

16. Testing the Application

Test the backend and frontend to ensure reliability.

16.1 Backend Testing

- **Create** `server/tests/sample.test.js`:

```
const request = require('supertest');
const app = require('../index');
```

```
describe('Sample API', () => {
  it('should return hello message', async () => {
    const res = await request(app).get('/api/sample');
    expect(res.statusCode).toEqual(200);
    expect(res.text).toBe('Hello from sample API!');
  });
});
```

- **Run tests**:

```
npm test
```

16.2 Frontend Testing

- **Install testing libraries** in `client`:

```
npm install --save-dev @testing-library/react @testing-library/jest-dom
```

- **Create** `client/__tests__/page.test.js`:

```
import { render, screen } from '@testing-library/react';
import Home from '../app/page';

test('renders todo app heading', () => {
  render(<Home />);
  const heading = screen.getByText(/Todo App/i);
  expect(heading).toBeInTheDocument();
});
```

- **Run tests**:

```
npm test
```

17. Build for Production

Prepare the application for deployment.

- **Frontend Build**:

```
cd client
npm run build
```

  - Output: `client/out` (static) or `client/.next` (server).

- **Serve Frontend from Backend**:

  - Install `path` in `server`:

```
npm install path
```

  - **Update** `server/index.js`:

```
const path = require('path');
app.use(express.static(path.join(__dirname, '../client/out')));
```

```
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, '../client/out', 'index.html'));
});
```

- Copy `client/out` to `server`.

- **Test Production Build**:

```
cd server
npm start
```

Visit `http://localhost:5000`.

18. Deploy to Vercel (Frontend) and Render (Backend)

Deploy the application to cloud platforms.

- **Frontend (Vercel)**:
  1. Push code to GitLab (see Section 15).
  2. Sign up at https://vercel.com.
  3. Import the `client` folder's repository from GitLab:
     - Select *Import Git Repository* and authenticate with GitLab.
     - Choose the `mern-project/client` directory.
  4. Set environment variables (if any) and deploy.

- **Backend (Render)**:
  1. Sign up at https://render.com.
  2. Create a new Node.js service and link the `server` folder's repository.
  3. Set environment variables (`MONGO_URI`, `JWT_SECRET`, `PORT`).
  4. Deploy and note the backend URL.
  5. Update `client/app/page.js` to use the backend URL.

- **Test Deployment**:
  - Verify the app works on Vercel and communicates with the Render backend.

19. Set Up CI/CD with GitLab CI

Automate testing with GitLab CI.

- **Create** `.gitlab-ci.yml`:

```
stages:
  - test

test_job:
  stage: test
  image: node:20
  script:
    - cd client
    - npm install
    - npm test
    - cd ../server
```

```
    - npm install
    - npm test
  only:
    - main
    - merge_requests
```

- **Push to GitLab**:

```
git add .gitlab-ci.yml
git commit -m "Add GitLab CI configuration"
git push origin main
```

- **Verify the pipeline** in GitLab (*CI/CD* > *Pipelines*).

20. Useful Next.js Commands

```
npm run dev       # Run in development mode
npm run build     # Create production build
npm start         # Run production build
```

21. Useful Node Commands

```
npm install       # Install all dependencies
npm start         # Start the server
npm run dev       # Start server with nodemon
```

22. What You Need While Building a MERN Stack Project

- **Code Editor**: VS Code with extensions (Prettier, ESLint, GitLens).
- **Version Control**: Git and GitLab.
- **API Testing**: Postman.
- **Database**: MongoDB (local or Atlas).
- **Backend**: Express.js and Node.js.
- **Frontend**: Next.js for React-based UI.
- **Styling**: Tailwind CSS.
- **Environment Variables**: Managed via `.env`.
- **Code Quality**: ESLint and Prettier.
- **Package Manager**: npm.

23. Final Project Structure

```
mern-project/
├── client/
│   ├── app/
│   │   ├── page.js
│   │   └── globals.css
│   ├── components/
│   ├── __tests__/
│   ├── public/
│   ├── tailwind.config.js
```

```
│   ├── package.json
│   └── .gitignore
├── server/
│   ├── models/
│   │   ├── User.js
│   │   └── Todo.js
│   ├── routes/
│   │   ├── auth.js
│   │   ├── sample.js
│   │   └── todos.js
│   ├── middleware/
│   │   └── auth.js
│   ├── tests/
│   │   └── sample.test.js
│   ├── index.js
│   ├── .env
│   ├── package.json
│   └── .gitignore
├── .gitlab-ci.yml
├── .gitignore
└── README.md
```

24. Additional Tools and Best Practices

- **Code Quality**:
  - Configure `.eslintrc.json`:

```
{
  "env": {
    "browser": true,
    "es2021": true,
    "node": true
  },
  "extends": ["eslint:recommended", "plugin:react/recommended"],
  "parserOptions": {
    "ecmaVersion": 12,
    "sourceType": "module"
  },
  "plugins": ["react"],
  "rules": {}
}
```

  - Configure `.prettierrc`:

```
{
  "semi": true,
  "trailingComma": "es5",
  "singleQuote": true,
  "printWidth": 80,
  "tabWidth": 2
}
```

- **Security**:
  - Use `helmet`:

    ```
    npm install helmet
    ```

    ```
    const helmet = require('helmet');
    app.use(helmet());
    ```

  - Validate inputs with `express-validator`:

    ```
    npm install express-validator
    ```

- **Monitoring**: Use MongoDB Atlas or New Relic.
- **Documentation**: Create a `README.md` and use Swagger:

  ```
  npm install swagger-ui-express
  ```

25. Conclusion

You’ve built an advanced MERN stack application with:

- Next.js for a modern React frontend with Tailwind CSS.
- Express.js and Node.js for a secure backend.
- MongoDB for data persistence.
- Authentication with JWT.
- Testing with Jest and Testing Library.
- CI/CD with GitLab CI.
- Version Control with GitLab using Git Bash.
- Deployment to Vercel and Render.

This setup is production-ready and can be extended with features like WebSocket, file uploads, or advanced UI components. Explore further with Redux, TypeScript, or GraphQL.
Happy coding!