🚀 MERN Project Generator CLI

> Create production-ready MERN stack projects in seconds!

NPM Package Website mern-project-cli

Website https://devcli.vercel.app

MERN Project Generator CLI is a powerful tool designed to simplify the process of setting up a complete, production-ready MERN stack project in seconds.

This tool eliminates the need for manual configurations, boilerplate code copying, and repetitive tasks, allowing developers to start building their apps right away with best practices in place. Perfect for both beginners and experienced developers, it saves time and ensures a solid project foundation.

✨ Features

- **One Command Setup**: Generate both frontend and backend with a single command
- **Industry-Standard Structure**: Pre-configured folder structure following best practices
- **Create frontend with shadcn and vite**, a new React project with either Shadcn UI + Tailwind CSS or just Vite + Tailwind CSS using a single command.
- **Instant MongoDB Integration**: Connect to MongoDB with zero configuration
- **Generate Mongoose Schema**: Generate Mongoose Schema with just one command
- **Development Ready**: Hot-reloading enabled for both frontend and backend
- **Pre-configured Environment**: `.env.example` files included with sensible defaults
- **Git Ready**: Initialized Git repository with proper `.gitignore` files

Index

Requirements

Before you begin, ensure your system meets these requirements:

- **Node.js**: Version 14.x or higher
- **npm**: Version 6.x or higher
- **MongoDB**: Local or remote installation

Installation

Install the CLI tool globally to use it from anywhere in your system:

```
npm install -g mern-project-cli
```

To check installation version:

```
devcli --version
```

Commands

1. Create MERN Project

```
devcli create <your_project_name>
```

What This Command Does:

1. Creates Project Structure:

The generated project follows the MVC (Model-View-Controller) pattern, a battle-tested architecture that separates your application into three main components:

```
your-project-name/
├── backend/
│   ├── controllers/         # Handle business logicdocumentation
│   ├── db/                  # Database configuration
│   ├── middlewares/         # Custom middleware functionsdocumentation
│   ├── models/              # MongoDB Schema model
│   ├── routes/              # API route definitions
```

```
│   ├── utils/              # Helper functionsdocumentation
│   ├── .env.example        # Environment variables template
│   ├── .gitignore          # Git ignore rules
│   ├── constants.js        # Application constants
│   ├── package.json        # Dependencies and scripts
│   ├── README.md           # Backend documentation
│   └── server.js           # Server entry point
└── frontend/
    ├── public/             # Static files
    ├── src/                # React source code
    │   ├── components/     # React components
    │   ├── pages/          # Page components
    │   ├── utils/          # Helper functions
    │   └── App.js          # Root component
    ├── .env.example        # Frontend environment template
    └── package.json        # Frontend dependencies
```

2. Installs Dependencies:

- Backend: Express, Mongoose, CORS, dotenv, nodemon.
- Frontend: React, React Router, Axios, Other Create React App dependencies.

After Creation:

Start Backend Development:

```
cd your-project-name/backend
```

```
npm run dev             # Start development server with nodemon
```

Start Frontend Development:

```
cd your-project-name/frontend
```

```
npm start               # Start React App
```

Option:

```
devcli create my_project --next
```

2. Connect MongoDB

- Create database as your_project_name_db

```
devcli mongodb-connect
```

- Or with custom database name

```
devcli mongodb-connect --project custom-name
```

Options:

- `-p, --project <name>`: Specify custom database name
- No options: Uses project folder name as database name

What This Command Does:

1. Creates Database Connection:

- Generates `connection.js` in the `db` folder
- Sets up Mongoose connection with error handling
- Configures connection string based on environment variables

2. Updates Server Configuration:

- Adds database connection import to `server.js`
- Sets up connection status logging

Usage Examples:

```
# Using project name
devcli mongodb-connect

# Using custom database name
devcli mongodb-connect --project custom_name
```

Generated Files:

```
// db/connection.js
require('dotenv').config();
const mongoose = require('mongoose');

const dburl = process.env.DB_URL || 'mongodb://localhost:27017/your_db_name';
```

```
mongoose
  .connect(dburl)
  .then(() => console.log('Connected to DB Successfully'))
  .catch((err) => console.log(err.message));
```

3. Generate Mongoose Schema

- Create mongoose schema for your backend.

```
devcli devcli mongoose-schema <schema-name> <fieldName:fieldType
fieldName:fieldType ...>
```

Usage Example

```
devcli mongoose-schema User name:String email:String password:String
```

This will create a `User.js` file with a Mongoose schema inside the `models/` directory:

```
//models/User.js
import mongoose from 'mongoose';

const UserSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true },
  password: { type: String, required: true },
});

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

Explanation:

The `mongoose-schema` command takes a model name (User) and field definitions (name:String, email:String, password:String), generating a Mongoose model file in the `models/` folder.

4. Add Redux

Set up Redux in your project or add new Redux slices.

Initialize Redux

```
devcli add-redux --init
```

What does this command do:

- Sets up Redux store configuration
- Creates necessary store files and directories
- Installs required dependencies (@reduxjs/toolkit and react-redux)
- Creates hooks for easier Redux usage

Create Redux Slice

```
devcli add-redux --slice <sliceName> --actions="action1,action2" --
state="field1:type,field2:type"
```

Options:

- `--slice`: Name of the slice to create
- `--actions`: Comma-separated list of actions for the slice
- `--state`: Initial state fields with types (string, boolean, array)

Usage Example:

```
devcli add-redux --slice user --actions="login,logout" --
state="username:string,isLoggedIn:boolean"
```

This creates:

- A new slice file in `src/store/slices`
- Boilerplate for specified actions
- Initial state with typed fields
- Automatic integration with the Redux store

Example Generated Redux Slice

When you run the command:

```
devcli add-redux --slice user --actions="login,logout" --
state="username:string,isLoggedIn:boolean"
```

It generates the following slice in `src/store/slices/userSlice.js`:

```
import { createSlice } from '@reduxjs/toolkit';

const initialState = {
```

```
  username: '',
  isLoggedIn: false,
};

const userSlice = createSlice({
  name: 'user',
  initialState,
  reducers: {
    login: (state, action) => {
      // Implement login logic here
    },
    logout: (state, action) => {
      // Implement logout logic here
    },
  },
});

export const { login, logout } = userSlice.actions;
export default userSlice.reducer;
```

5. Create Frontend Project

Create a new React project with either Shadcn UI + Tailwind CSS or just Vite + Tailwind CSS using a single command.

```
# Create project with Shadcn UI
devcli create-frontend <project_name> --shadcn

# Create project with Vite + Tailwind CSS
devcli create-frontend <project_name> --vite
```

Features

With --shadcn flag:

- Creates a Vite + React project
- Installs and configures Tailwind CSS
- Sets up Shadcn UI with New York style and Zinc color scheme
- Configures project structure with best practices
- Adds initial button component as example
- Sets up path aliases for better imports
- Includes all necessary configuration files

With --vite flag:

- Creates a basic Vite + React project
- Installs and configures Tailwind CSS
- Sets up minimal project structure

- Includes starter template with modern styling

Options

- `--shadcn`: Include Shadcn UI setup with Tailwind CSS
- `--vite`: Create basic Vite project with Tailwind CSS only

Usage Examples

```
# Create a new React project with Shadcn UI
devcli create-frontend my-app --shadcn

# Create a new React project with just Vite + Tailwind
devcli create-frontend my-app --vite

# Navigate to project
cd my-app

# Start development server
npm run dev
```

Generated Project Structure with --shadcn

```
your-project/
├── src/
│   ├── components/
│   │   └── ui/
│   │       └── button.jsx
│   ├── lib/
│   │   └── utils.js
│   ├── App.jsx
│   └── index.css
├── jsconfig.json
├── tailwind.config.js
├── vite.config.js
└── components.json
```

After Creation with --shadcn

- Add more Shadcn components using:

  ```
  npx shadcn@latest add <component-name>
  ```

- Available components can be found at shadcn/ui components
- Customize theme in `tailwind.config.js`

- Add your own components in `src/components`

6. Initialize Docker Files

Generate Dockerfiles for both backend and frontend, along with a docker-compose.yml file for your MERN stack project.

```
devcli init-dockerfiles
```

What This Command Does:

1. **Creates Backend Dockerfile**:
    - Uses Node.js 20 Alpine image
    - Sets up working directory
    - Installs dependencies
    - Configures for development mode
    - Exposes port 5000
2. **Creates Frontend Dockerfile**:
    - Uses Node.js 20 Alpine image
    - Sets up working directory
    - Installs dependencies
    - Exposes port 3000
    - Configures for development mode
3. **Generates docker-compose.yml**:
    - Configures services for backend, frontend, and MongoDB
    - Sets up proper networking between services
    - Configures volumes for development
    - Sets environment variables
    - Establishes service dependencies

Requirements:

- Project must have `backend` and `frontend` directories in root
- Docker must be installed on your system

Generated Files:

```
your-project/
├── backend/
│   ├── Dockerfile
│   └── .dockerignore
├── frontend/
│   ├── Dockerfile
```

```
│   └── .dockerignore
└── docker-compose.yml
```

Usage:

```
# Navigate to your project root
cd your-project

# Generate Docker files
devcli init-dockerfiles

# Start the containerized application
docker-compose up
```

This will start your application with:

- Backend running on `http://localhost:5000`
- Frontend running on `http://localhost:3000`
- MongoDB running on port `27017`

7. Add ESLint and Prettierrc

Initialize ESLint in the specified directory (frontend, backend, or the current directory) to ensure consistent code quality with tailored configurations based on the project type.

```
devcli add-eslint [directory]   # Set up ESLint in the specified directory
(defaults to current directory)
```

What This Command Does:

- **Automatically Detects Project Type**: Determines if the project is a React, Vue, TypeScript, Node.js, or plain JavaScript application.
- **Configures ESLint**: Creates a `.eslintrc.json` file specific to the detected environment (e.g., browser for React, Node.js for backend).
- **Installs Dependencies**: Automatically installs ESLint, Prettier, and their necessary plugins as development dependencies in the specified directory.
- **Supports Multiple File Extensions**: Handles various file types based on the project structure.

Example Usage

- To set up ESLint in the backend directory:

  ```
  devcli add-eslint backend
  ```

- To set up ESLint in the frontend directory:

```
devcli add-eslint frontend
```

- To set up ESLint in the current directory (default):

```
devcli add-eslint
```

Example Generated ESLint Configuration

This command generates a basic ESLint configuration file (`.eslintrc.json`) that looks like this:

For Backend Directory:

```
{
  "env": {
    "browser": false,
    "node": true,
    "es2021": true
  },
  "extends": ["eslint:recommended", "plugin:prettier/recommended"],
  "parserOptions": {
    "ecmaVersion": 12
  },
  "rules": {}
}
```

For Frontend Directory:

```
{
  "env": {
    "browser": true,
    "node": false,
    "es2021": true
  },
  "extends": [
    "eslint:recommended",
    "plugin:react/recommended",
    "plugin:prettier/recommended"
  ],
  "parserOptions": {
    "ecmaVersion": 12,
    "ecmaFeatures": {
      "jsx": true
    }
  },
```

```
  "rules": {}
}
```

For Arbitrary Folders (Defaulting to Node):

```
{
  "env": {
    "browser": false,
    "node": true,
    "es2021": true
  },
  "extends": ["eslint:recommended", "plugin:prettier/recommended"],
  "parserOptions": {
    "ecmaVersion": 12
  },
  "rules": {}
}
```

Benefits

- **Automates Setup**: Saves time by automating the ESLint configuration process based on project type.
- **Ensures Consistency**: Helps maintain consistent linting rules across backend and frontend codebases.
- **Supports Arbitrary Setup**: Allows for easy ESLint configuration in any directory, defaulting to Node.js environment.

8. Add JWT Authetication and Authorization

Here is the content for the 8th command, "Add JWT Authentication":

8. Add JWT Authentication and Authorization

Add JWT authentication boilerplate to your backend project.

```
devcli add-jwt-auth
```

What This Command Does:

1. **Creates Necessary Directories**:
    - `controllers/authController.js`
    - `middlewares/authMiddleware.js`
    - `models/userModel.js`
    - `routes/authRoutes.js`
2. **Generates Authentication Logic**:
    - `authController.js` - Handles user registration and login with JWT token generation.

- `authMiddleware.js` - Implements middleware to authenticate and authorize requests using JWT tokens.
- `userModel.js` - Defines a Mongoose schema for the User model.
- `authRoutes.js` - Defines API routes for authentication, including register, login, and a protected route.

3. **Installs Required Dependencies**:
   - `bcryptjs` - For password hashing
   - `jsonwebtoken` - For generating and verifying JWT tokens
4. **Integrates Authentication Routes**:
   - Adds the authentication routes to the `server.js` file.
5. **Provides Next Steps**:
   - Update the `.env` file with a secure `JWT_SECRET`.
   - Start the server and test the authentication routes:
     - `POST /api/auth/register`: Register a new user
     - `POST /api/auth/login`: Log in and get the JWT token
     - `GET /api/auth/protected`: Access the protected route with the JWT token

Usage:

1. Run the command in your project's `backend` directory:

```
devcli add-jwt-auth
```

2. Update the `.env` file in the `backend` directory with a secure `JWT_SECRET`.
3. Start the server and test the authentication routes.

Generated Files:

```
backend/
├── controllers/
│   └── authController.js
├── middlewares/
│   └── authMiddleware.js
├── models/
│   └── userModel.js
├── routes/
│   └── authRoutes.js
```

The generated files implement the following functionality:

1. **authController.js**: Handles user registration and login, generating JWT tokens.

2. **authMiddleware.js**: Middleware to authenticate and authorize requests using JWT tokens.
3. **userModel.js**: Mongoose schema and model for the User.
4. **authRoutes.js**: API routes for authentication, including register, login, and a protected route.

After running this command, you can start using the authentication system in your backend application.

📖 Complete User Journey Example

Let's create a blog application from scratch:

```
# Step 1: Install CLI globally
npm install -g mern-project-cli

# Step 2: Create new project
devcli create my-blog-app

# Step 3: Set up backend
cd my-blog-app/backend
npm run dev

# Step 4: Set up frontend (new terminal)
cd ../frontend
npm start

# Step 5: Connect MongoDB (optional)
cd ../backend
devcli mongodb-connect

# Step 6: Generate Mongoose Scheama (optional)
devcli mongoose-schema Blog name:String category:String


# Step 7: Set up Redux
cd ../frontend
devcli add-redux --init

# Step 8: Set up Es-lint and prettierrc
cd ../backend
devcli add-eslint

cd ../frontend
devcli add-eslint
# Step 9: Create blog slice for Redux
devcli add-redux --slice blog --actions="addPost,deletePost,updatePost" --
state="posts:array,loading:boolean"

# Step 10: Add jwt authetication
cd ..backend
devcli add-jwt-auth

🎉 Congratulations! Your blog application structure is ready with:
- Backend running on `http://localhost:5000`
```

```
- Frontend running on `http://localhost:3000`
- MongoDB connected and ready to use
```

⚙️ Environment Configuration

Backend (.env)

```
# Server Configuration
PORT=5000

# Database Configuration
DB_URI=mongodb://localhost:27017/your_db_name
```

Frontend (.env)

```
# API Configuration
REACT_APP_API_URL=http://localhost:5000
```

🔧 Available Commands

CLI Commands

Project Setup

```
npm install -g mern-project-cli    # Install CLI globally
devcli --version                   # Check CLI version
devcli create <project-name>       # Create new MERN project
```

```
OR [Create frontend with shadcn+tailwind/ vite+tailwind]

devcli create-frontend <project-name> --shadcn    # shadcn-frontend
devcli create-frontend <project-name> --vite      # vite-frontend
```

Backend CLI Commands

```
# Database Connection
devcli mongodb-connect                                    # Connect MongoDB
using project name
devcli mongodb-connect -p custom-name                     # Connect with
custom database name
```

```
# Schema Generation
devcli mongoose-schema <schema-name> <fieldName:fieldType ...>  # Generate
Mongoose schema
# Example: devcli mongoose-schema User name:String email:String password:String
```

Frontend CLI Commands

```
# Redux Setup
devcli add-redux --init                                          # Initialize
Redux in frontend
devcli add-redux --slice <name> --actions="action1,action2" --
state="field1:type,field2:type"       #Create Slice
# Example: devcli add-redux --slice user --actions="login,logout" --
state="username:string,isLoggedIn:boolean"
```

Docker CLI Commands

```
# Docker Configuration
devcli init-dockerfiles            # Generate Dockerfiles and docker-compose.yml
```

Development Commands

Backend Development

```
cd backend                # Navigate to backend directory
npm install               # Install dependencies
npm run dev               # Start with auto-reload (development)
npm start                 # Start without auto-reload (production)
```

Frontend Development

```
cd frontend               # Navigate to frontend directory
npm install               # Install dependencies
npm start                 # Start development server
```

Docker Development

```
docker-compose up         # Start all services (backend, frontend, mongodb)
docker-compose down       # Stop all services
```

```
docker-compose up --build  # Rebuild and start all services
```

Future Enhancements

1. **Code Generation** More Code-Snippets

Contribute to the Project

We welcome and appreciate contributions to MERN Project Generator CLI! If you'd like to help improve this tool, feel free to do so.

License

This project is licensed under the MIT License - see the LICENSE file for details.

Support the Project

If you find this tool helpful, please consider:

- Giving it a star on GitHub
- View on NPM mern-project-cli
- Sharing it with your fellow developers
- Contributing to its development

Build with by Manish Raj

Portfolio • GitHub • LinkedIn • Twitter