# **MERN Stack**

In this project, you are required to implement a web solution based on **MERN** stack on a Linux server.

**MERN** stack is a web development framework. It consists of MongoDB, ExpressJS, ReactJS, and NodeJS as its working components.

_images/MERN-stack.png

As shown in the illustration above, the user interacts with the ReactJS UI components at the application front-end residing in the browser. This frontend is served by the application backend residing in a server, through ExpressJS running on top of NodeJS.

Any interaction that causes a data change request is sent to the NodeJS based Express server, which grabs data from the MongoDB database if required, and returns the data to the frontend of the application, which is then presented to the user.

1. **MongoDB**: A document-oriented, No-SQL database used to store the application data.
2. **NodeJS**: The JavaScript runtime environment. It is used to run JavaScript on a machine rather than in a browser.
3. **ExpressJS**: A server side framework for NodeJS. A framework on the other side is a compilation of code libraries that a developer can use to speed up development work. Rather than writing code completely from scratch.
4. **ReactJS**: A frontend framework developed by Facebook. It is based on javascript, used to build UI components.

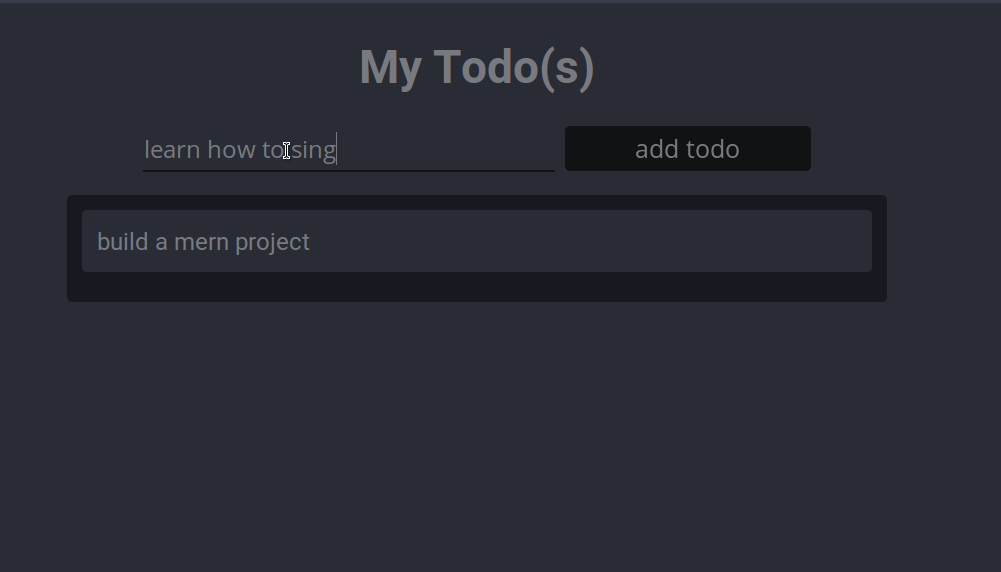
## **Project Prerequisites**

In order to complete this project, you will need to have a fresh install of Ubuntu 20.04 in your Virtual Box. Do Not use the same server from Project 1.

1. [Here is how to configure Virtual Box](https://www.youtube.com/watch?v=wql6adU2JeE&list=PLtPuNR8I4TvkwU7Zu0l0G_uwtSUXLckvh&index=3)
2. [Here is how to install and configure Ubuntu 20.04](https://www.youtube.com/watch?v=Lw-VAYWJumo&list=PLtPuNR8I4TvkwU7Zu0l0G_uwtSUXLckvh&index=4)

### **Requirement**

Deploy an application that is used to create **TODO** lists. like what we have in the image below.



### **Lets get started.**

### **Prepare the backend Linux server**

Update ubuntu

sudo apt-get update

Upgrade ubuntu

sudo apt-get upgrade

Lets get the location of nodejs software from ubuntu repositories over the internet.

curl -sL https://deb.nodesource.com/setup\_12.x | sudo -E bash -

## **First we will install *nodejs* on the server.**

Install Node.js with the command below

sudo apt-get install -y nodejs

**Note**: The command above will install both nodejs and npm. NPM is a package manager. Similar to what apt does on Ubuntu, NPM is used to install node modules or softwares, and manages dependency conflicts intelligently.

Verify the node installation with the command below

node -v

Verify the node installation with the command below

npm -v

### **Application Code Setup**

Let’s start with the setup processes. Open your terminal and create a new file directory in any convienient location on your Ubuntu Desktop. Name it todo. Run the command below to achieve this:

$ mkdir Todo

Run the command below to verify that the Todo directory is created

$ ls

Now enter into that file directory with the command below

$ cd Todo

Next, you will use the command npm init to initialise your project, so that a new file named package.json will be created. This file will normally contain information about your app and the dependencies that it needs to run. Follow the prompts after running the command. You can hit enter to accept default values, then accept to write out the package.json file by typing yes

_images/npm-init.png

Run the command ls to confirm that you have package.json file created

### **Install ExpressJS**

Remember that express is a framework for nodejs, therefore a lot of things developers would have programmed is already taken care of out of the box. Therefore it simplifies development, and abstracts a lot of low level details. For example, express helps to define routes of your application based on HTTP methods and URLs.

To use express, install it using npm:

$ npm install express

Now create a file index.js with the command below

$ touch index.js

Run ls to confirm that your index.js file is successfully created

Install the dotenv module

npm install dotenv

Open the index.js file with the command below

$ vim index.js

Type the code below into it and save. Do not get overwhelmed by the code you see. For now, simply paste the code into the file.

const express = require('express');

require('dotenv').config();

const app = express();

const port = process.env.PORT || 5000;

app.use((req, res, next) => {

res.header("Access-Control-Allow-Origin", "\\*");

res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

next();

});

app.use((req, res, next) => {

res.send('Welcome to Express');

});

app.listen(port, () => {

console.log(`Server running on port ${port}`)

});

Notice that we have specified to use port 5000 in the code. This will be required later when we go on the browser.

Use :w to save in vim and use :qa to exit vim

It’s time to start our server to see if it works. Open your terminal in the same directory as your index.js file and type:

$ node index.js

If every thing goes well, you should see **Server running on port 5000** in your terminal.

_images/node-index.png

Open up your browser, localhost:5000 and you should be able to see below message in the browser.

Welcome to Express

_images/welcomeExpress.JPG

### **Routes**

There are three things that the app needs to do:

1. Create a task
2. View all tasks
3. Delete a completed task.

For each task, we need to create routes that will define various endpoints that the todo app will depend on. So let’s create a folder routes

$ mkdir routes

Change directory into the routes folder.

$ cd routes

Now, create a file api.js with the command below

touch api.js

Open the file with the command below

vim api.js

input below code in it. (*Do not be overwhelmed with the code*)

const express = require ('express');

const router = express.Router();

router.get('/todos', (req, res, next) => {

});

router.post('/todos', (req, res, next) => {

});

router.delete('/todos/:id', (req, res, next) => {

})

module.exports = router;

### **Models**

Now comes the interesting part, since the app is going to make use of [Mongodb](https://www.mongodb.com/) which is a NoSql database \*\*\*(Its fine if you dont have much knowledge in databases, just follow along for now.)\*\*\*, we need to create a model.

A model is at the heart of Javascript based applications, and it is what makes it interactive.

We will also use models to define the database schema . This is important so that we will be able to define the fields stored in each Mongodb document. ***(Seems like a lot of information, but not to worry, everything will become clear to you overtime. I promise!!!)***

In essence the Schema is a blueprint of how the database will be constructed, including other data fields that may not be required to be stored in the database. These are known as *virtual properties*

To create a Schema and a model, install [*mongoose*](https://mongoosejs.com/) which is a node package that makes working with mongodb easier.

Change directory back Todo folder with cd .. and install Mongoose

$ npm install mongoose

Now we will do 3 things at once by chaining 2 commands into one with the && directive.

1. Create a new folder with mkdir models
2. And change directory into the newly created models folder with cd models.

Inside the models folder, Create a file and name it todo.js

touch todo.js

Open the file created with vim todo.js then paste the code below in the file:

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

//create schema **for** todo

const TodoSchema = new Schema({

action: {

type: String,

required: [true, 'The todo text field is required']

}

})

//create model **for** todo

const Todo = mongoose.model('todo', TodoSchema);

module.exports = Todo;

Now we need to update our routes from the file api.js in Routes directory to make use of the new model.

In Routes directory, open api.js with vim api.js, delete the code inside with :%d and paste ther code below into it then save and exit

const express = require ('express');

const router = express.Router();

const Todo = require('../models/todo');

router.get('/todos', (req, res, next) => {

//this will **return** all the data, exposing only the id **and** action field to the client

Todo.find({}, 'action')

.then(data => res.json(data))

.catch(next)

});

router.post('/todos', (req, res, next) => {

**if**(req.body.action){

Todo.create(req.body)

.then(data => res.json(data))

.catch(next)

}**else** {

res.json({

error: "The input field is empty"

})

}

});

router.delete('/todos/:id', (req, res, next) => {

Todo.findOneAndDelete({"\_id": req.params.id})

.then(data => res.json(data))

.catch(next)

})

module.exports = router;

### **MongoDB Database**

We need a database where we will store our data. For this we will make use of **mLab**. mLab provides MogoDB database as a service solution, so to make life easy, you will need to sign up for the shared clusters free account which is ideal for our use case. [Sign up here](https://www.mongodb.com/atlas-signup-from-mlab). Follow the sign up process, select **AWS** as the cloud provider, and choose a region near you.

Complete the get started checklist as shown in the image below

_images/MLab-dashboard.png

Ensure to allow access to the MongoDB database from anywhere (*Not secure, but it is ideal for testing*)

**IMPORTANT NOTE** In the image below, make sure you change the time of deleting the entry from 6 Hours to 1 Week

_images/MogoDB-Network-Access.png

Create a MongoDB database and collection inside mLab

_images/Mongo-create-DB-1.png

_images/Mongo-create-DB-2.png

_images/Mongo-create-DB-2.png

In the index.js file, we used process.env to access environment variables, but we have not yet created it. So we need to do that now.

Create a file in your TODO directory with and name it .env. Add the connection string to access the database in it, just as below:

DB = 'mongodb+srv://<username>:<password>@<network-address>/<dbname>?retryWrites=true&w=majority'

Ensure to update <username>, <password>, <network-address> and <database> according to your setup

Here is how to get your connection string

_images/Mongo-connect1.png _images/Mongo-connect2.png _images/Mongo-connect3.png

Now we need to update the index.js to reflect the use of dotenv so that nodejs can connect to the database.

Simply delete existing content in the file, and update it with the entire code below.

To do that using vim, follow below steps

1. Open the file with vim index.js
2. Press esc
3. Type :
4. Type %d
5. Hit enter

The entire content will be deleted, then,

1. Press i to enter the *insert* mode in *vim*
2. Now, paste the entire code below in the file.

const express = require('express');

const bodyParser = require('body-parser');

const mongoose = require('mongoose');

const routes = require('./routes/api');

const path = require('path');

require('dotenv').config();

const app = express();

const port = process.env.PORT || 5000;

//connect to the database

mongoose.connect(process.env.DB, { useNewUrlParser: true })

.then(() => console.log(`Database connected successfully`))

.catch(err => console.log(err));

//since mongoose promise is depreciated, we overide it with node's promise

mongoose.Promise = global.Promise;

app.use((req, res, next) => {

res.header("Access-Control-Allow-Origin", "\\*");

res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

next();

});

app.use(bodyParser.json());

app.use('/api', routes);

app.use((err, req, res, next) => {

console.log(err);

next();

});

app.listen(port, () => {

console.log(`Server running on port ${port}`)

});

Using environment variables to store information is considered more secure and best practice to separate configuration and secret data from the application, instead of writing connection strings directly inside the index.js application file.

Start your server using the command:

node index.js

### **Testing The Code Without A Frontend User Interface**

So far we have written our TODO application, and configured backend database, but we do not have a frontend UI yet. We need ReactJS code to achieve that. But during development, we will need a way to test our code using RESTFUL API. Therefore, we will need to make use of some api development clients to test our code.

In this project, we will use [Postman](https://www.getpostman.com/) to test for our API. Click [**Install Postman**](https://www.getpostman.com/downloads/) to download and install postman in your machine.

Click [**HERE**](https://www.youtube.com/watch?v=FjgYtQK_zLE) to learn how perform CRUD operartion on Postman

You should test all the api endpoints and make sure they are working. For the endpoints that require body, you should send json back with the necessary fields since it’s what we setup in our code.

Now open your Postman, create a post request to the API http://localhost:5000/api/todos. Check the image below:

_images/post-request.JPG

Create get request to your API on http://localhost:5000/api/todos

_images/get-request.JPG

### **Creating the Frontend**

Since we are done with the functionality we want from our api, it is time to create an interface for the client to interact with the api. To start out with the frontend of the todo app, we will use the create-react-app command to scaffold our app.

In the same root directory as your backend code, which is the Todo directory, run:

$ npx create-react-app client

This will create a new folder in your Todo directory called client, where you will add all the react code.

### **Running the React App**

Before testing the react app, there are a number of dependencies that need to be installed.

1. Install concurrently. It is used to run more than one command simultaneously from the same terminal window.

$ npm install concurrently --save-dev

1. Install nodemon. It is used to run and monitor the server. If there is any change in the server code, nodemon will restart it automatically and load the new changes.

$ npm install nodemon --save-dev

1. In the Todo folder and open the package.json file. Change the highlighted part of the below screenshot and replace with the code below.

"scripts": {

"start": "node index.js",

"start-watch": "nodemon index.js",

"dev": "concurrently **\"**npm run start-watch**\"** **\"**cd client && npm start**\"**"

},

_images/script.JPG

## **Configure Proxy in package.json**

1. Enter into the client folder from Todo directory

cd client

1. Open the package.json file

vim package.json

1. Add the key value pair in the package.json file "proxy": "http://localhost:5000".

The whole purpose of adding the proxy configuration in number 3 above is to make it possible to access the application directly from the browser by simply calling the server url like http://localhost:5000 rather than always including the entire path like http://localhost:5000//api/todos

Now, ensure you are inside the Todo directory, and simply do:

npm run dev

Your app should open and running on localhost:3000

### **Creating your React Components**

One of the advantages of react is that it makes use of components, which are reusable and also makes code modular. For our todo app, there will be two state components and one stateless component. From your Todo directory run

cd client

move to the src directory

cd src

Inside your src folder create another folder called components

mkdir components

Move into the components directory with

cd components

inside the components directory create three files Input.js, ListTodo.js and Todo.js.

touch Input.js ListTodo.js Todo.js

Open Input.js file

vim Input.js

Copy and paste the following

**import** **React**, { Component } **from** 'react';

**import** **axios** **from** 'axios';

**class** **Input** extends Component {

state = {

action: ""

}

addTodo = () => {

const task = {action: this.state.action}

**if**(task.action && task.action.length > 0){

axios.post('/api/todos', task)

.then(res => {

**if**(res.data){

this.props.getTodos();

this.setState({action: ""})

}

})

.catch(err => console.log(err))

}**else** {

console.log('input field required')

}

}

handleChange = (e) => {

this.setState({

action: e.target.value

})

}

render() {

let { action } = this.state;

**return** (

<div>

<input type="text" onChange={this.handleChange} value={action} />

<button onClick={this.addTodo}>add todo</button>

</div>

)

}

}

export default Input

To make use of axios, which is a Promise based HTTP client for the browser and node.js, you need to cd into your client from your terminal and run yarn add axios or npm install axios.

Move to the src folder

cd ..

Move to clients folder

cd ..

Install Axios

$ npm install axios

Go to components directory

cd src/components

After that open your ListTodo.js

vim ListTodo.js

in the ListTodo.js copy and paste the following code

import React from 'react';

const ListTodo = ({ todos, deleteTodo }) => {

return (

<ul>

{

todos &&

todos.length > 0 ?

(

todos.map(todo => {

return (

<li key={todo.\_id} onClick={() => deleteTodo(todo.\_id)}>{todo.action}</li>

)

})

)

:

(

<li>No todo(s) left</li>

)

}

</ul>

)

}

export default ListTodo

Then in your Todo.js file you write the following code

import React, {Component} from 'react';

import axios from 'axios';

import Input from './Input';

import ListTodo from './ListTodo';

class Todo extends Component {

state = {

todos: []

}

componentDidMount(){

this.getTodos();

}

getTodos = () => {

axios.get('/api/todos')

.then(res => {

if(res.data){

this.setState({

todos: res.data

})

}

})

.catch(err => console.log(err))

}

deleteTodo = (id) => {

axios.delete(`/api/todos/${id}`)

.then(res => {

if(res.data){

this.getTodos()

}

})

.catch(err => console.log(err))

}

render() {

let { todos } = this.state;

return(

<div>

<h1>My Todo(s)</h1>

<Input getTodos={this.getTodos}/>

<ListTodo todos={todos} deleteTodo={this.deleteTodo}/>

</div>

)

}

}

export default Todo;

We need to make little adjustment to our react code. Delete the logo and adjust our App.js to look like this.

Move to the src folder

cd ..

Make sure that you are in the src folder and do

vim App.js

Copy and paste the code below into it

**import** **React** **from** 'react';

**import** **Todo** **from** './components/Todo';

**import** './App.css';

const App = () => {

**return** (

<div className="App">

<Todo />

</div>

);

}

export default App;

After pasting, exit the editor.

In the src directory open the App.css

vim App.css

Then paste the following code into App.css:

.App {

text-align: center;

font-size: calc(10px + 2vmin);

width: 60%;

margin-left: auto;

margin-right: auto;

}

input {

height: 40px;

width: 50%;

border: none;

border-bottom: 2px *#101113 solid;*

background: none;

font-size: 1.5rem;

color: *#787a80;*

}

input:focus {

outline: none;

}

button {

width: 25%;

height: 45px;

border: none;

margin-left: 10px;

font-size: 25px;

background: *#101113;*

border-radius: 5px;

color: *#787a80;*

cursor: pointer;

}

button:focus {

outline: none;

}

ul {

list-style: none;

text-align: left;

padding: 15px;

background: *#171a1f;*

border-radius: 5px;

}

li {

padding: 15px;

font-size: 1.5rem;

margin-bottom: 15px;

background: *#282c34;*

border-radius: 5px;

overflow-wrap: **break**-word;

cursor: pointer;

}

@media only screen **and** (min-width: 300px) {

.App {

width: 80%;

}

input {

width: 100%

}

button {

width: 100%;

margin-top: 15px;

margin-left: 0;

}

}

@media only screen **and** (min-width: 640px) {

.App {

width: 60%;

}

input {

width: 50%;

}

button {

width: 30%;

margin-left: 10px;

margin-top: 0;

}

}

Exit

In the src directory open the index.css

vim index.css

Copy and paste the code below:

body {

margin: 0;

padding: 0;

font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", "Roboto", "Oxygen",

"Ubuntu", "Cantarell", "Fira Sans", "Droid Sans", "Helvetica Neue",

sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

box-sizing: border-box;

background-color: *#282c34;*

color: *#787a80;*

}

code {

font-family: source-code-pro, Menlo, Monaco, Consolas, "Courier New",

monospace;

}

Go to the Todo directory

cd ../..

When you are in the Todo directory run:

npm run dev

Assuming no errors when saving all these files, the todo app should be ready and fully functional with the functionality discussed earlier: creating a task, deleting a task and viewing all your tasks.

_images/alldone.JPG

### **Conclusion**

In this tutorial, you created a todo app using the MERN stack. You wrote a frontend application using React that communicates with a backend application written using expressjs. You also created a Mongodb backend for storing tasks in a database.

### **Instructions On How To Submit Your Work For Review And Feedback**

Follow the steps below to submit your work for review

1. Create a word document on Google Drive
2. Write down the steps you took to accomplish your work
3. Make screenshots wherever possible.
   1. On Windows you can use Snipping Tool
   2. On Mac you can use Lightshot
4. Name your document according to the Project work
5. Click the Submit your project for review in DAREY.IO project dashboard, and paste the link to your project, and submit.

Credits:

1. [This guide was inspired by Digital Ocean](https://www.digitalocean.com/community/tutorials/getting-started-with-the-mern-stack)
2. [Educative.io](https://www.educative.io/edpresso/what-is-mern-stack)