Este tutorial lo guiará a través de los pasos para crear una aplicación CRUD simple usando la pila MEVN (MongoDB, Express.js, Vue.js y Node.js). Axios se utiliza para realizar solicitudes HTTP.

[**Vue.js**](https://vuejs.org/)es un marco de JavaScript para crear interfaces de usuario. La ventaja de vue sobre otros trabajos de marcos frontales es su facilidad de aprendizaje.

[**Node.js**](https://nodejs.org/en/) es un entorno de ejecución de JavaScript de código abierto, multiplataforma, que ejecuta código JavaScript fuera de un navegador.

[**Express.js**](https://expressjs.com/) es un marco de aplicación web para Node.js.

[**MongoDB**](https://www.mongodb.com/) es una base de datos NoSQL multiplataforma de código abierto utilizada por la mayoría de las aplicaciones modernas. [MongoDB Compass](https://www.mongodb.com/products/compass) proporcionará una GUI para MongoDB

[**Axio s**](https://github.com/axios/axios)es un cliente HTTP basado en promesas para el navegador y Node.js

En este tutorial vamos a crear una sencilla aplicación de tareas pendientes con la funcionalidad CRUD básica.

Antes de comenzar con este tutorial, asegúrese de haber instalado Node.js, Vue CLI y MongoDB en su sistema.

**Construyendo la aplicación**

1. Cree un directorio llamado 'Todo\_tuts' y navegue hasta el directorio

mkdir Todo\_tuts   
cd Todo\_tuts

## **Empecemos creando back-end**

1. Cree un directorio llamado server dentro de 'Todo\_tuts'.
2. Navegue al directorio server y ejecute el siguiente comando.

npm init

Seleccione las opciones predeterminadas presionando la tecla Intro.

Texto

Descripción generada automáticamente

Instale las dependencias 'Express.js', 'BodyParser' y 'Mongoose'. BodyParser se usa para analizar la solicitud del cliente, Mongoose para el modelado de objetos y cors se usa para evitar el error Cross-Origin Request Block mientras intentamos comunicarnos desde el front-end hasta el back-end

Ejecute el siguiente comando para instalar dependencias.

npm install express body-parser mongoose cors --save

Se creará un archivo package.json en el Todo\_tuts/server que se verá así

Texto

Descripción generada automáticamente

## **Ahora es el momento de crear un servidor back-end**

Cree un archivo llamado directorio index.jsinterno server. Dentro del index.jsarchivo pegue el siguiente código

xpress y cors se importan primero. Luego, estamos inicializando una aplicación rápida. app.use(cors())habilita el uso compartido de recursos entre orígenes para que las solicitudes entrantes no sean bloqueadas por la política de cors. app.listencrea un servidor back-end en[http://localhost:3000](http://localhost:3000/)

Para iniciar el servidor, ejecute el siguiente comando en la terminal.

node index

Verá la siguiente salida en la terminal si se inicia el servidor



## **Conexión a la base de datos**

Antes de crear las rutas, conectemos nuestra aplicación a MongoDB. Para eso primero necesitamos importar mongoose.

const mongoose = require('mongoose')

Para conectarse a MongoDB, pegue el siguiente código en index.js

mongoose.connect("mongodb://localhost:27017/todoapp", {  
 useNewUrlParser: true,  
 useUnifiedTopology: true  
});var db = mongoose.connection;db.on('open', () => {  
 console.log('Connected to mongoDB');  
});db.on('error', (error) => {  
 console.log(error)  
})

mongoose.connect se utiliza para conectarse a MongoDB. 27017 es el puerto predeterminado para MongoDB y todoapp es el nombre de nuestra base de datos.

Tras una conexión exitosa a MongoDB, recibirá el siguiente mensaje en la terminal

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

## **Creación de esquemas**

Ahora necesitamos crear un esquema para nuestra base de datos. El esquema define la estructura de los datos almacenados en la base de datos. Cree un archivo llamado directorio todo\_schema.js interno server.

Nuestro esquema de tareas tiene los siguientes campos:

* Título
* Terminado

Primero deberíamos importar mongoose. Luego crea un esquema usando mongoose.schema. Agregue el siguiente código al todo\_schema.jsarchivo.

const mongoose = require("mongoose");  
const todoSchema = mongoose.Schema({  
title: {  
type: String,  
required: true},   
completed: {  
type: Boolean,  
required: true}  
});  
const todo = (module.exports = mongoose.model("todo", todoSchema));

Finalmente exportamos el esquema como modelo de esquema usando mongoose.model ().

## **Definición de rutas**

Las rutas se utilizan para manejar la solicitud del cliente. Necesitamos establecer rutas para agregar una tarea pendiente, actualizar las tareas pendientes como completadas, recuperar las tareas pendientes creadas y eliminar las tareas pendientes.

Necesitamos importar body-parser para analizar el cuerpo de la solicitud para obtener detalles de tareas pendientes cada vez que llegue una solicitud de publicación, para eso, agregue el siguiente código a index.js

Primero necesitamos importar el modelo de esquema de tareas pendientes. Para eso, agregue el siguiente código aindex.js

let todoModel = require('./todo\_schema')

**Crear una tarea pendiente**

Para crear una tarea pendiente, copie el siguiente código en index.js

app.post('/todo/add', (req, res) => {

    let newTodo = new todoModel;

    newTodo.title = req.body.todo;

    newTodo.completed = false;

    newTodo.save((err) => {

      if(err){

        res.send("Error while adding Todo");

      }else{

        res.send("Todo added");

      }

  })

El código anterior crea un nuevo modelo de tareas pendientes llamado 'newTodo' y guarda los detalles de las tareas pendientes al analizar la solicitud y luego los guarda en la base de datos.

**Obteniendo tareas pendientes**

La obtención de las tareas pendientes y el envío al cliente se realiza en dos partes. es decir, necesitamos buscar tanto las tareas pendientes completadas como las incompletas. Para facilitar la comprensión crearemos dos rutas diferentes para ello. es decir, una ruta para las tareas pendientes completadas y la otra para las tareas pendientes incompletas.

Ruta para las tareas pendientes completadas

app.get('/todo/completed', (req, res) => {

    todoModel.find({ completed: true }, (err, todos) => {

      if (err) {

        res.send("Error while fetching Todos");

      } else {

        res.json(todos)

      }

    })

  })

Ruta para tareas pendientes incompletas

app.get('/todo/uncompleted', (req, res) => {

    todoModel.find({completed:false},(err, todos) => {

      if(err){

        res.send("Error while fetching Todos");

      }else{

        res.json(todos)

      }

    })

  })

**Actualización de tareas pendientes**

Para actualizar una tarea pendiente como completada, cambiamos el campo completado de tarea pendiente a verdadero. Para cada documento en MongoDB, MongoDB proporcionará un ID de objeto por sí mismo. Pasaremos este ID de objeto a la API de back-end y actualizaremos el campo completo de la tarea pendiente como verdadero.

app.post('/todo/complete/:id',(req, res) => {

    todoModel.findByIdAndUpdate(req.params.id, {completed: true},         (err, todo) =>{

      if(!err){

        res.send("Good Work");

      }

    })

  })

**Eliminar tareas pendientes**

Para eliminar la tarea pendiente, recibiremos el ID de objeto de la tarea pendiente que se eliminará del front-end y borraremos la tarea pendiente que tenga el ID de objeto correspondiente.

app.delete('/todo/:id', (req, res) => {

    let query = { \_id: req.params.id }

    todoModel.deleteOne(query, (err) => {

      if(err){

        res.send("Error while deleting todo")

      }else{

        res.send("Todo deleted")

      }

    })

  })

Nuestro servidor back-end está listo. Ahora pasemos al front-end

## **Comencemos a crear front-end**

1. Dentro del directorio 'Todo\_tuts' ejecute el siguiente comando en la terminal

vue create client

Le pedirá un preajuste. Simplemente seleccione el preajuste predeterminado. Luego, vue creará la carpeta 'cliente' para nuestro código de front-end.

Para ver nuestra aplicación

cd client  
npm run serve

## **comencemos instalando dependencias**

Para el front-end estamos usando axios y vuetify. axios es para hacer una solicitud HTTP al back-end y vuetify es un marco CSS.Dentro de la carpeta del cliente, ejecute los siguientes comandos en una terminal

Primero instalaremos vuetify usando el siguiente comando

vue add vuetify

Luego instale axios usando el siguiente comando

npm install axios –save

Para comunicarnos con el back-end, realizaremos todas nuestras solicitudes http://localhost:3000con diferentes endpoints. Porque nuestro servidor back-end se ejecutará en http://localhost:3000. Para esto, nuestro servidor back-end debería estar ejecutándose siempre que ocurra cualquier solicitud.

**Crear tarea**

Pegue el siguiente código en Todo\_tuts/client/src/App.vue

Captura de pantalla de un celular

Descripción generada automáticamente

Al hacer clic en el botón Add ToDo, se llamará a la función addTodo. Dentro de la función, estamos haciendo una solicitud de publicación para http://localhost:3000/todo/addusar axios.

**Buscar tareas pendientes**

Agregue el siguiente código a App.vue dentro de la v-app etiqueta debajo del código agregado anteriormente.

<!-- uncompleted todos -->

<div class="d-flex justify-center">

  <h1>Unompleted ToDo</h1>

</div>

<div v-for="todo in uncompletedTodos" :key="todo.\_id">

  <v-card class="mx-auto" color="white" dark max-width="800">

    <v-card-text class="font-weight-bold title blue--text">

    {{ todo.title }}

      <v-list-item id="todo-list-item" class="grow">

        <v-btn @click="completeTodo(todo.\_id)" class="mx-2" small

        color="green"> Done </v-btn>

       <v-btn @click="deleteTodo(todo.\_id)" class="mx-2" small       color="red"> Delete </v-btn>

      </v-list-item>

    </v-card-text>

  </v-card>

</div>

<!-- completed todos -->

<div class="d-flex justify-center">

  <h1>Completed ToDo</h1>

</div>

<div v-for="todo in completedTodos" :key="todo.\_id">

  <v-card class="mx-auto" color="blue" dark max-width="800">

    <v-card-text class="font-weight-bold title white--text">

    {{ todo.title }}

      <v-list-item id="todo-list-item" class="grow">

        <v-btn @click="deleteTodo(todo.\_id)" class="mx-2" small

        color="red"> Delete </v-btn>

      </v-list-item>

    </v-card-text>

  </v-card>

</div>

Aquí estamos renderizando las tareas pendientes completadas y no completadas por separado almacenándolas en dos matrices diferentes.

Agregue esto dentro data de la script etiqueta

UnpletedTodos: [],   
completeTodos: []

Estas dos matrices almacenarán nuestras tareas pendientes completadas y no completadas. Los renderizaremos usando la v-fordirectiva.

Agregue esto a continuación methodsen la scriptetiqueta

created() {

  // fetch uncompleted task

  axios.get("http://localhost:3000/todo/uncompleted")

  .then(response => (this.uncompletedTodos = response.data))

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

  // fetch completed task

  axios.get("http://localhost:3000/todo/completed")

  .then(response => (this.completedTodos = response.data))

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

}

Este es un gancho creado que es uno de los ganchos del ciclo de vida en vue. Cuando se crea nuestra aplicación, estas solicitudes de obtención de axios llegarán a la API de back-end y recuperarán las tareas completadas y no completadas. Agregaremos estas tareas pendientes a dos matrices, una para tareas completadas y otra para tareas no completadas, y usando la v-fordirectiva renderizaremos tanto las tareas completadas como las no completadas.

**Eliminar tarea pendiente**

Agregue el siguiente código a continuación methods en la scriptetiqueta enApp.vue

deleteTodo(todoID) {

  axios

  .delete("http://localhost:3000/todo/" + todoID).then(response => {

   console.log(response.data);

  });

}

Estamos realizando una solicitud de eliminación a http://localhost:3000/todo/'+todoID. La función deleteTodo se llamará cuando se presione el botón Delete. El todoID es el ID de objeto mongodb de cada tarea pendiente, que se pasará a la función que se utiliza para eliminar la tarea pendiente correspondiente.

**Actualizar tareas pendientes**

Agregue este código dentro methodde la scriptetiqueta enApp.vue

completeTodo(todoID) {

  axios.post("http://localhost:3000/todo/complete/" + todoID)

  .then(response => {

    console.log(response.data);

  });

}

Para actualizar la tarea pendiente como completada, estamos haciendo una solicitud de publicación a http://localhost:3000/todo/complete/+todoID. La tarea con todoID correspondiente se actualizará en el back-end.

Actualice el sitio después de cada cambio. Entonces solo el cambio se reflejará en el sitio web.

Viva…. Habíamos creado una aplicación CRUD simple usando la pila MEVN.

El código completo de este tutorial se puede encontrar [**aquí.**](https://github.com/albinj12/todoapp-medium-tutorial)

<https://albinjose.medium.com/create-a-crud-app-using-vue-js-and-node-js-mevn-stack-5410680450e4>