Nuestro Primer CRUD MEVN

El siguiente ejercicio 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). Asimismo, se hará uso de Axios (Cliente HTTP basado en promesas) para realizar solicitudes HTTP.

Para el desarrollo del ejercicio, utilizaremos:

- **Vue.js** 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** es un entorno de ejecución de JavaScript de código abierto, multiplataforma, que ejecuta código JavaScript fuera de un navegador.
- **Express.js** es un marco de aplicación web para Node.js.
- **MongoDB** es una base de datos NoSQL multiplataforma de código abierto utilizada por la mayoría de las aplicaciones modernas. MongoDB Compass proporcionará una GUI para MongoDB
- **Axios** es un cliente HTTP basado en promesas para el navegador y Node.js

Ahora, vamos a crear una sencilla aplicación de tareas pendientes con la funcionalidad CRUD básica.

Prerrequisitos

Tener instalado en el pc lo siguiente:

- Editor VSCode.
- Node.js.
- Vue CLI
- MongoDB.

Construyendo la aplicación

Cree un directorio llamado “Todo_tuts” y navegue hasta el directorio

```
mkdir Todo_tuts
```

Cambie a ese directorio

```
cd Todo_tuts
```

Creando el back-end

Cree un directorio llamado “server” dentro de “Todo_tuts”.

```
mkdir server
```

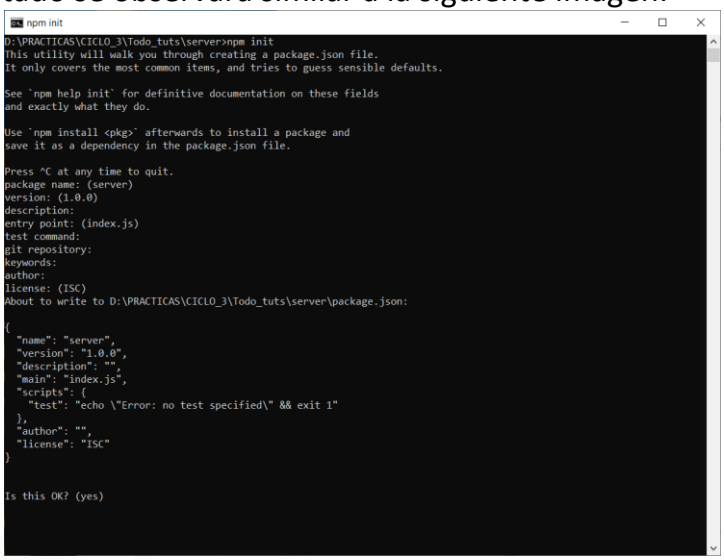
Cambie a ese directorio

```
cd server
```

Ejecute el siguiente comando para crear el archivo “package.json” del lado del servidor

```
npm init
```

A continuación, seleccione las opciones predeterminadas presionando la tecla <Intro>. El resultado se observará similar a la siguiente imagen:



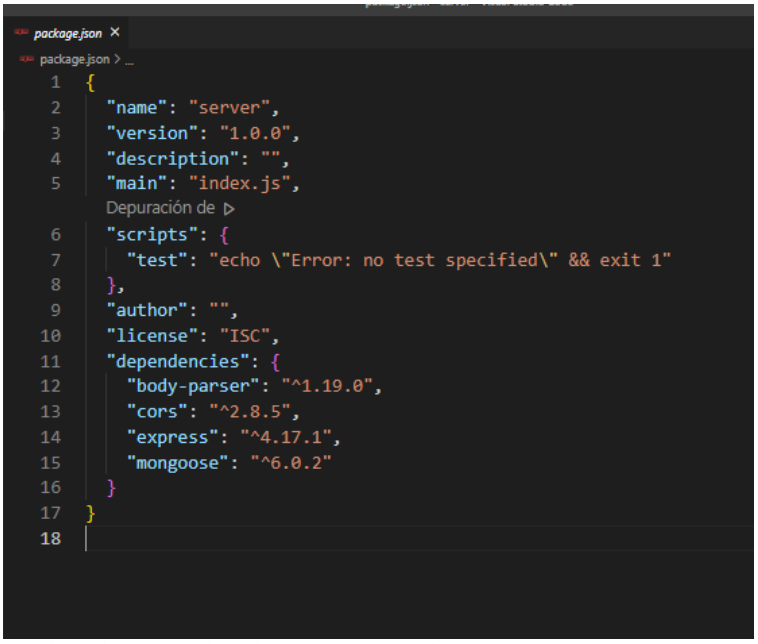
A continuación, instale las siguientes dependencias:

- Express.js: Es una infraestructura de aplicaciones web Node.js que proporciona capacidades para desarrollo de API Rest. Sirve como middleware o backend. Para más información, visite el siguiente sitio web: https://expressjs.com/es/
- BodyParser: Para poder acceder a los datos de una publicación tenemos que usar body-parser. Básicamente, lo que body-parser es lo que permite a Express leer el cuerpo y luego analizarlo en un objeto Json que podamos entender.
- Mongoose es una librería para Node. js que nos permite escribir consultas para una base de datos de MongooDB, con características como validaciones, construcción de queries, middlewares, conversión de tipos y algunas otras, que enriquecen la funcionalidad de la base de datos.

Ahora, ejecute el siguiente comando para instalar las dependencias:

```
npm install express body-parser mongoose cors --save
```

En consecuencia, se creará un archivo “package.json” en la ruta “Todo_tuts/server” que se verá de la siguiente manera:



Crear el servidor back-end

Cree un archivo dentro de la carpeta “server” llamado “index.js”. A continuación, pegue el siguiente código dentro de este archivo

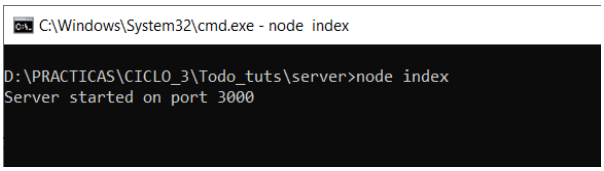
```
const express = require("express");
const cors = require("cors");
const app = express();
app.use(cors());
app.listen(3000, () => {
  console.log("Server started on port 3000");
});
```

En el código anterior, “Express” y “cors” se importan primero. Luego, se inicializa una aplicación rápida. app.use(cors()) que habilita el uso compartido de recursos entre orígenes para que las solicitudes HTTP entrantes no sean bloqueadas por la política de cors.
app.listen crea un servidor back-end en 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

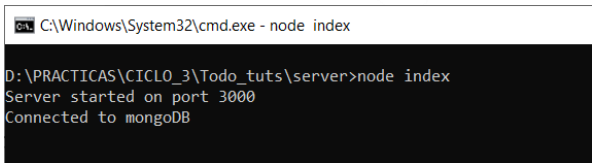
```
mongoose.connect("mongodb://localhost:27017/todoapp", {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

var db = mongoose.connection;
db.on("open", () => {
  console.log("Connected to mongoDB");
});

db.on("error", (error) => {
  console.log(error);
});
```

Ahora, index.js se verá de la siguiente manera:

```
const express = require("express");
const cors = require("cors");
const mongoose = require("mongoose");

const app = express();

app.use(cors());

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);
});

app.listen(3000, () => {
  console.log("Server started on port 3000");
```

```
});
```

Entonces, mongoose.connect se utiliza para conectarse a MongoDB. 27017 como 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



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 “todo_schema.js” dentro del directorio “server”. Nuestro esquema de tareas tiene los siguientes campos:

- Title
- Completed

Ahora, se importa “mongoose” de primero. Luego se crea un esquema usando mongoose.schema.

Agregue el siguiente código dentro del archivo todo_schema.js.

```
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 HTTP 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 y para obtener detalles de tareas pendientes cada vez que llegue una solicitud de publicación, para eso, agregue el siguiente código en “index.js”

```
const bodyParser = require("body-parser");


// parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: false }));

// parse application/json
app.use(bodyParser.json());
```

Ahora, necesitamos importar el modelo de esquema de tareas pendientes. Para eso, agregue el siguiente código en “index.js”.

```
// import todo schema as model
let todoModel = require("./todo_schema");
```

El código de “index.js” se verá de la siguiente forma:

```
const express = require("express");
const bodyParser = require("body-parser");
const mongoose = require("mongoose");
const cors = require("cors");

const app = express();

app.use(cors());

// parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: false }));

// parse application/json
```

```
app.use(bodyParser.json());

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);
});

// import todo schema as model
let todoModel = require("./todo_schema");

app.listen(3000, () => {
  console.log("Server started on port 3000");
});
```

Crear una tarea pendiente

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

```
// ROUTES
app.get("/", (req, res) => {
  res.send("hello");
});

// add todo
app.post("/todo/add", (req, res) => {
    let newTodo = new todoModel();
    newTodo.title = req.body.todo;
    newTodo.completed = false;
    newTodo.save(err => {
      if (err) {
        console.log(err);
        res.send("Error while adding Todo");
      } else {
        console.log(newTodo);
        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.

Actualice “index.js” con la ruta para las tareas pendientes completadas

```
// FETCH TO-DO

// completed
app.get("/todo/completed", (req, res) => {
  todoModel.find({ completed: true }, (err, todos) => {
    if (err) {
      res.send("Error while fetching Todos");
    } else {
      res.json(todos);
    }
  });
});
```

Ahora, actualice “index.js” con la ruta para las tareas pendientes incompletas

```
// uncompleted
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 “true”. 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 ”true”.

Ahora, actualice “index.js” con el siguiente código

```
// update
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. En consecuencia, agregue el siguiente código en “index.js”:

```
// delete todo
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");
    }
  });
});
```

Una vez integrado todas las API’s, el código de “index.js” quedará de la siguiente forma:

```
const express = require("express");
const bodyParser = require("body-parser");
const mongoose = require("mongoose");
const cors = require("cors");
```

```
const app = express();

app.use(cors());

// parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: false }));

// parse application/json
app.use(bodyParser.json());

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);
});

// import todo schema as model
let todoModel = require("./todo_schema");

// ROUTES

app.get("/", (req, res) => {
  res.send("hello");
});

// add todo
app.post("/todo/add", (req, res) => {
  let newTodo = new todoModel();
  newTodo.title = req.body.todo;
  newTodo.completed = false;
  newTodo.save(err => {
    if (err) {
      console.log(err);
      res.send("Error while adding Todo");
    } else {
      console.log(newTodo);
      res.send("Todo added");
    }
  });
});

// FETCH TO-DO

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

```
  todoModel.find({ completed: true }, (err, todos) => {
    if (err) {
      res.send("Error while fetching Todos");
    } else {
      res.json(todos);
    }
  });
});

// uncompleted
app.get("/todo/uncompleted", (req, res) => {
  todoModel.find({ completed: false }, (err, todos) => {
    if (err) {
      res.send("Error while fetching Todos");
    } else {
      res.json(todos);
    }
  });
});

// update
app.post("/todo/complete/:id", (req, res) => {
  todoModel.findByIdAndUpdate(
    req.params.id,
    { completed: true },
    (err, todo) => {
      if (!err) {
        res.send("Good Work");
      }
    }
  );
});

// delete todo
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");
    }
  });
});

app.listen(3000, () => {
  console.log("Server started on port 3000");
});
```

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

Creación del front-end

Dentro del directorio “Todo_tuts” ejecute el siguiente comando en la terminal:

```
vue create client
```

Seguidamente, le pedirá un preajuste, donde simplemente seleccione el preajuste predeterminado. Luego, Vue creará la carpeta “client” para nuestro código de front-end.

Para ver nuestra aplicación

```
cd client
```

Luego

```
npm run serve
```

Instalación de dependencias

Para el front-end estamos usando axios y vuetify. Por su parte, axios se utiliza para hacer una solicitud HTTP al back-end. Por otro lado, vuetify es un framework 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:3000 con 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 el archivo “App.vue” que se encuentra en la ruta “Todo_tuts/client/src/App.vue”.

```
<template>
  <v-app>
    <div class="d-flex justify-center">
      <h1 id="addTodo">Add ToDo</h1>
    </div>
    <div class="d-flex justify-center">
      <v-col cols="6" style="margin: 0px auto">
        <v-text-field v-model="newTodo" label="Add Todo" solo></v-text-field>
      </v-col>
    </div>
    <div class="d-flex justify-center">
      <v-btn @click="addToDo()" color="primary">Add ToDo</v-btn>
    </div>
  </v-app>
</template>
<script>
import axios from "axios";
export default {
  data: () => ({
    newToDo: "",
  }),
  methods: {
    addToDo() {
      axios.post("http://localhost:3000/todo/add", {
        todo: this.newTodo
     }).then(response => {
       this.message = response.data;
     });
  }
};
</script>
```

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 usando axios.

Buscar tareas pendientes

Agregue el siguiente código a “App.vue” dentro de la etiqueta v-app 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>
<h1 class="text-center white--text">Completed Todos</h1>
<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” que se encuentra dentro de la etiqueta “script”

```
  uncompletedTodos: [],
  completedTodos: []
```

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

Ahora, agregue esto a continuación de “methods” en la etiqueta “script”.

```
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 hook creado que es uno de los hook 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 directiva v-for renderizaremos tanto las tareas completadas como las no completadas.

Eliminar tarea pendiente

Agregue el siguiente código a continuación de “methods” en la etiqueta “script” en “App.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 del “method” de la etiqueta “script” en “App.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.

Finalmente, el código de “App.vue” se verá de la siguiente manera:

```
<template>
  <v-app>
    <div class="d-flex justify-center">
      <h1 id="addTodo">Add ToDo</h1>
    </div>
    <div class="d-flex justify-center">
      <v-col cols="6" style="margin: 0px auto;">
        <v-text-field v-model="newTodo" label="Add Todo" solo></v-text-field>
      </v-col>
    </div>
    <div class="d-flex justify-center">
      <v-btn @click="addToDo()" color="primary">Add ToDo</v-btn>
    </div>
    <!-- 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>
    <h1 class="text-center white--text">Completed Todos</h1>
    <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>
  </v-app>
</template>

<script>
import axios from "axios";
export default {
  name: "App",
  data: () => ({
    newToDo: "",
    uncompletedTodos: [],
    completedTodos: []
  }),
  methods: {
    addToDo() {
      axios
        .post("http://localhost:3000/todo/add", {
          todo: this.newTodo
        })
        .then(response => {
          this.message = response.data;
        });
      this.newTodo = "";
    },
```

```
    completeTodo(todoID) {
      axios
        .post("http://localhost:3000/todo/complete/" + todoID)
        .then(response => {
          console.log(response.data);
        });
    },
    deleteTodo(todoID) {
      axios.delete("http://localhost:3000/todo/" + todoID).then(response => {
        console.log(response.data);
      });
    }
  },
  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));
  }
};
</script>
```