**Vue.js + Node.js + Express + PostgreSQL example: Build a full-stack CRUD Application**

[Last modified: April 30, 2021](https://www.bezkoder.com/vue-node-express-postgresql/)  [bezkoder](https://www.bezkoder.com/author/bezkoder/)  [Full Stack](https://www.bezkoder.com/category/full-stack/), [Node.js](https://www.bezkoder.com/category/node-js/), [Vue.js](https://www.bezkoder.com/category/vue/)

In this tutorial, I will show you how to build full-stack (Vue.js + Node.js + Express + PostgreSQL) example with a CRUD Application. The back-end server uses Node.js + Express for REST APIs, front-end side is a Vue client with Vue Router and axios.

More Practice: [Node.js Express + Vue.js: JWT Authentication & Authorization example](https://bezkoder.com/node-express-vue-jwt-auth/)

Run both projects (back-end & front-end) in one place:  
[How to serve/combine Vue App with Express](https://bezkoder.com/serve-vue-app-express/)

Serverless with Firebase:  
– [Vue Firebase Realtime Database: CRUD example](https://bezkoder.com/vue-firebase-realtime-database/)  
– [Vue Firestore: Build a CRUD App example](https://bezkoder.com/vue-firestore-crud/)

**Contents**[[hide](https://www.bezkoder.com/vue-node-express-postgresql/)]

* [Vue.js + Node.js + Express + PostgreSQL example Overview](https://www.bezkoder.com/vue-node-express-postgresql/#Vuejs_Nodejs_Express_PostgreSQL_example_Overview)
* [Full-stack CRUD App Architecture](https://www.bezkoder.com/vue-node-express-postgresql/#Full-stack_CRUD_App_Architecture)
* [Node.js Express Back-end](https://www.bezkoder.com/vue-node-express-postgresql/#Nodejs_Express_Back-end)
  + [Overview](https://www.bezkoder.com/vue-node-express-postgresql/#Overview)
  + [Project Structure](https://www.bezkoder.com/vue-node-express-postgresql/#Project_Structure)
  + [Implementation](https://www.bezkoder.com/vue-node-express-postgresql/#Implementation)
    - [Create Node.js App](https://www.bezkoder.com/vue-node-express-postgresql/#Create_Nodejs_App)
    - [Setup Express web server](https://www.bezkoder.com/vue-node-express-postgresql/#Setup_Express_web_server)
    - [Configure PostgreSQL database & Sequelize](https://www.bezkoder.com/vue-node-express-postgresql/#Configure_PostgreSQL_database_038_Sequelize)
    - [Initialize Sequelize](https://www.bezkoder.com/vue-node-express-postgresql/#Initialize_Sequelize)
    - [Define the Sequelize Model](https://www.bezkoder.com/vue-node-express-postgresql/#Define_the_Sequelize_Model)
    - [Create the Controller](https://www.bezkoder.com/vue-node-express-postgresql/#Create_the_Controller)
    - [Run the Node.js Express Server](https://www.bezkoder.com/vue-node-express-postgresql/#Run_the_Nodejs_Express_Server)
* [Vue.js Front-end](https://www.bezkoder.com/vue-node-express-postgresql/#Vuejs_Front-end)
  + [Overview](https://www.bezkoder.com/vue-node-express-postgresql/#Overview-2)
  + [Technology](https://www.bezkoder.com/vue-node-express-postgresql/#Technology)
  + [Project Structure](https://www.bezkoder.com/vue-node-express-postgresql/#Project_Structure-2)
  + [Implementation](https://www.bezkoder.com/vue-node-express-postgresql/#Implementation-2)
    - [Setup Vue.js Project](https://www.bezkoder.com/vue-node-express-postgresql/#Setup_Vuejs_Project)
    - [Add Vue Router to Vue.js 2 CRUD App](https://www.bezkoder.com/vue-node-express-postgresql/#Add_Vue_Router_to_Vuejs_2_CRUD_App)
    - [Add Navbar and Router View to Vue.js 2 CRUD App](https://www.bezkoder.com/vue-node-express-postgresql/#Add_Navbar_and_Router_View_to_Vuejs_2_CRUD_App)
    - [Initialize Axios for Vue.js 2 CRUD HTTP Client](https://www.bezkoder.com/vue-node-express-postgresql/#Initialize_Axios_for_Vuejs_2_CRUD_HTTP_Client)
    - [Create Data Service](https://www.bezkoder.com/vue-node-express-postgresql/#Create_Data_Service)
    - [Create Vue Components](https://www.bezkoder.com/vue-node-express-postgresql/#Create_Vue_Components)
    - [Run Vue.js Client](https://www.bezkoder.com/vue-node-express-postgresql/#Run_Vuejs_Client)
* [Conclusion](https://www.bezkoder.com/vue-node-express-postgresql/#Conclusion)

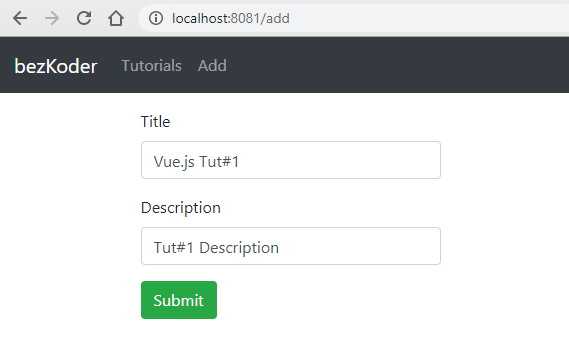
**Vue.js + Node.js + Express + PostgreSQL example Overview**

We will build a full-stack Tutorial Application in that:

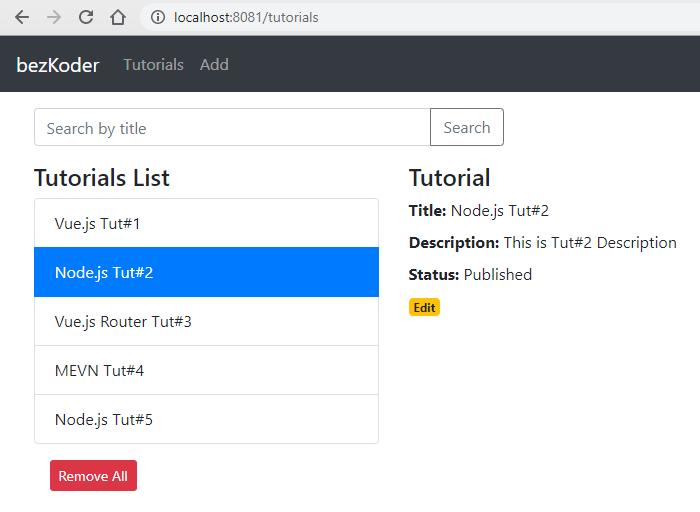
* Tutorial has id, title, description, published status.
* User can create, retrieve, update, delete Tutorials.
* There is a search box for finding Tutorials by title.

Here are screenshots of the example.

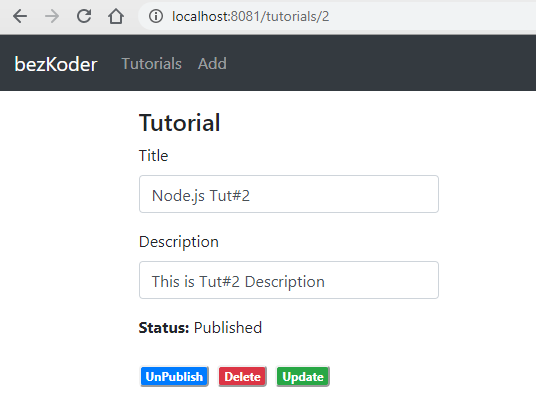
– Add an object:



– Show all objects:



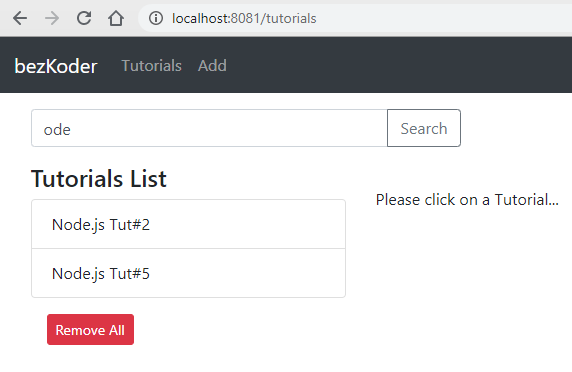
– Click on **Edit** button to update an object:



On this Page, you can:

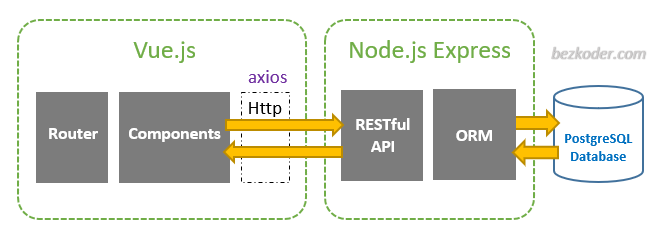
* change status to **Published**/**Pending** using **Publish**/**UnPublished** button
* remove the object from PostgreSQL Database using **Delete** button
* update this object’s details on Database with **Update** button

– Search objects by field ‘title’:



**Full-stack CRUD App Architecture**

We’re gonna build the application with following architecture:



– Node.js Express exports REST APIs & interacts with PostgreSQL Database using Sequelize ORM.  
– Vue Client sends HTTP Requests and retrieves HTTP Responses using *axios*, consume data on the components. Vue Router is used for navigating to pages.

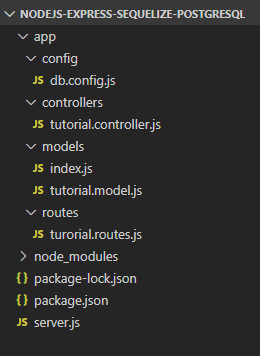
**Node.js Express Back-end**

**Overview**

These are APIs that Node.js Express App will export:

| **Methods** | **Urls** | **Actions** |
| --- | --- | --- |
| GET | api/tutorials | get all Tutorials |
| GET | api/tutorials/:id | get Tutorial by id |
| POST | api/tutorials | add new Tutorial |
| PUT | api/tutorials/:id | update Tutorial by id |
| DELETE | api/tutorials/:id | remove Tutorial by id |
| DELETE | api/tutorials | remove all Tutorials |
| GET | api/tutorials?title=[kw] | find all Tutorials which title contains 'kw' |

**Project Structure**



– *db.config.js* exports configuring parameters for PostgreSQL connection & Sequelize.  
– **Express** web server in *server.js* where we configure CORS, initialize & run Express REST APIs.  
– Next, we add configuration for PostgreSQL database in **models**/*index.js*, create **Sequelize** data model in **models**/*tutorial.model.js*.  
– Tutorial controller in **controllers**.  
– Routes for handling all CRUD operations (including custom finder) in *tutorial.routes.js*.

**Implementation**

**Create Node.js App**

First, we create a folder:

$ mkdir nodejs-express-sequelize-postgresql

$ cd nodejs-express-sequelize-postgresql

Next, we initialize the Node.js App with a *package.json* file:

npm init

name: (nodejs-express-sequelize-postgresql)

version: (1.0.0)

description: Node.js Rest Apis with Express, Sequelize & PostgreSQL.

entry point: (index.js) server.js

test command:

git repository:

keywords: nodejs, express, sequelize, postgresql, rest, api

author: bezkoder

license: (ISC)

Is this ok? (yes) yes

We need to install necessary modules: express, sequelize, pg, pg-hstore and body-parser.  
Run the command:

npm install express sequelize pg pg-hstore body-parser cors --save

\*pg for PostgreSQL and pg-hstore for converting data into the PostgreSQL hstore format.

**Setup Express web server**

In the root folder, let’s create a new *server.js* file:

const express = require("express");

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

const cors = require("cors");

const app = express();

var corsOptions = {

origin: "http://localhost:8081"

};

app.use(cors(corsOptions));

// parse requests of content-type - application/json

app.use(bodyParser.json());

// parse requests of content-type - application/x-www-form-urlencoded

app.use(bodyParser.urlencoded({ extended: true }));

// simple route

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

res.json({ message: "Welcome to bezkoder application." });

});

// set port, listen for requests

const PORT = process.env.PORT || 8080;

app.listen(PORT, () => {

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

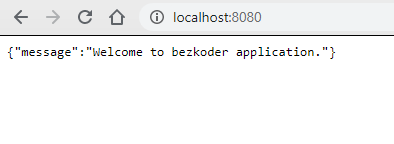
});

What we do are:  
– import express, body-parser and cors modules:

* Express is for building the Rest apis
* [body-parser](https://www.npmjs.com/package/body-parser) helps to parse the request and create the req.body object
* [cors](https://www.npmjs.com/package/cors) provides Express middleware to enable CORS with various options.

– create an Express app, then add body-parser and cors middlewares using app.use() method. Notice that we set origin: http://localhost:8081.  
– define a GET route which is simple for test.  
– listen on port 8080 for incoming requests.

Now let’s run the app with command: node server.js.  
Open your browser with url <http://localhost:8080/>, you will see:



Yeah, the first step is done. We’re gonna work with Sequelize in the next section.

**Configure PostgreSQL database & Sequelize**

In the *app* folder, we create a separate *config* folder for configuration with *db.config.js* file like this:

module.exports = {

HOST: "localhost",

USER: "postgres",

PASSWORD: "123",

DB: "testdb",

dialect: "postgres",

pool: {

max: 5,

min: 0,

acquire: 30000,

idle: 10000

}

};

First five parameters are for PostgreSQL connection.  
pool is optional, it will be used for Sequelize connection pool configuration:

* max: maximum number of connection in pool
* min: minimum number of connection in pool
* idle: maximum time, in milliseconds, that a connection can be idle before being released
* acquire: maximum time, in milliseconds, that pool will try to get connection before throwing error

For more details, please visit [API Reference for the Sequelize constructor](https://sequelize.org/master/class/lib/sequelize.js~Sequelize.html#instance-constructor-constructor).

**Initialize Sequelize**

We’re gonna initialize Sequelize in **app**/**models** folder that will contain model in the next step.

Now create **app**/**models**/*index.js* with the following code:

const dbConfig = require("../config/db.config.js");

const Sequelize = require("sequelize");

const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER, dbConfig.PASSWORD, {

host: dbConfig.HOST,

dialect: dbConfig.dialect,

operatorsAliases: false,

pool: {

max: dbConfig.pool.max,

min: dbConfig.pool.min,

acquire: dbConfig.pool.acquire,

idle: dbConfig.pool.idle

}

});

const db = {};

db.Sequelize = Sequelize;

db.sequelize = sequelize;

db.tutorials = require("./tutorial.model.js")(sequelize, Sequelize);

module.exports = db;

Don’t forget to call sync() method in *server.js*:

...

const app = express();

app.use(...);

const db = require("./app/models");

db.sequelize.sync();

...

In development, you may need to drop existing tables and re-sync database. Just use force: true as following code:

db.sequelize.sync({ force: true }).then(() => {

console.log("Drop and re-sync db.");

});

**Define the Sequelize Model**

In *models* folder, create *tutorial.model.js* file like this:

module.exports = (sequelize, Sequelize) => {

const Tutorial = sequelize.define("tutorial", {

title: {

type: Sequelize.STRING

},

description: {

type: Sequelize.STRING

},

published: {

type: Sequelize.BOOLEAN

}

});

return Tutorial;

};

This Sequelize Model represents **tutorials** table in PostgreSQL database. These columns will be generated automatically: *id*, *title*, *description*, *published*, *createdAt*, *updatedAt*.

After initializing Sequelize, we don’t need to write CRUD functions, Sequelize supports all of them:

* create a new Tutorial: [create](https://sequelize.org/master/class/lib/model.js~Model.html#static-method-create)(object)
* find a Tutorial by id: [findByPk](https://sequelize.org/master/class/lib/model.js~Model.html" \l "static-method-findByPk)(id)
* get all Tutorials: [findAll](https://sequelize.org/master/class/lib/model.js~Model.html" \l "static-method-findAll)()
* update a Tutorial by id: [update](https://sequelize.org/master/class/lib/model.js~Model.html#static-method-update)(data, where: { id: id })
* remove a Tutorial: [destroy](https://sequelize.org/master/class/lib/model.js~Model.html#static-method-destroy)(where: { id: id })
* remove all Tutorials: destroy(where: {})
* find all Tutorials by title: findAll({ where: { title: ... } })

These functions will be used in our Controller.

We can improve the example by adding Comments for each Tutorial. It is the One-to-Many Relationship and I write a tutorial for this at:  
[Node.js Sequelize Associations: One-to-Many example](https://bezkoder.com/sequelize-associate-one-to-many/)

Or you can add Tags for each Tutorial and add Tutorials to Tag (Many-to-Many Relationship):  
[Node.js Sequelize Associations: Many-to-Many example](https://bezkoder.com/sequelize-associate-many-to-many/)

**Create the Controller**

Inside **app**/**controllers** folder, let’s create *tutorial.controller.js* with these CRUD functions:

* create
* findAll
* findOne
* update
* delete
* deleteAll
* findAllPublished

const db = require("../models");

const Tutorial = db.tutorials;

const Op = db.Sequelize.Op;

// Create and Save a new Tutorial

exports.create = (req, res) => {

};

// Retrieve all Tutorials from the database.

exports.findAll = (req, res) => {

};

// Find a single Tutorial with an id

exports.findOne = (req, res) => {

};

// Update a Tutorial by the id in the request

exports.update = (req, res) => {

};

// Delete a Tutorial with the specified id in the request

exports.delete = (req, res) => {

};

// Delete all Tutorials from the database.

exports.deleteAll = (req, res) => {

};

// Find all published Tutorials

exports.findAllPublished = (req, res) => {

};

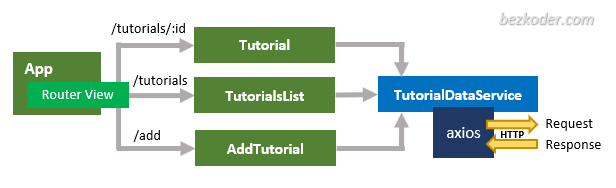
You can continue with step by step to implement this Node.js Express App in the post:  
[Node.js Express & PostgreSQL: CRUD Rest APIs example with Sequelize](https://bezkoder.com/node-express-sequelize-postgresql/)

**Run the Node.js Express Server**

Run our Node.js application with command: node server.js.

**Vue.js Front-end**

**Overview**



– The App component is a container with router-view. It has navbar that links to routes paths.

– TutorialsList component gets and displays Tutorials.  
– Tutorial component has form for editing Tutorial’s details based on :id.  
– AddTutorial component has form for submission new Tutorial.

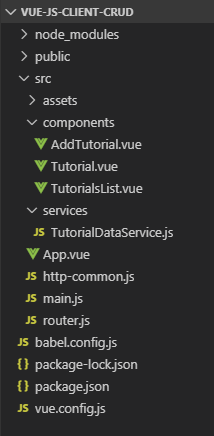
– These Components call TutorialDataService methods which use axios to make HTTP requests and receive responses.

**Technology**

* vue: 2.6.10
* vue-router: 3.1.3
* axios: 0.19.0

If you want to use Vue 3 instead, please visit:  
[Vue 3 CRUD example with Axios & Vue Router](https://bezkoder.com/vue-3-crud/)

**Project Structure**



– **package.json** contains 3 main modules: vue, vue-router, axios.  
– There are 3 components: TutorialsList, Tutorial, AddTutorial.  
– **router.js** defines routes for each component.  
– **http-common.js** initializes axios with HTTP base Url and headers.  
– TutorialDataService has methods for sending HTTP requests to the Apis.  
– **vue.config.js** configures *port* for this Vue Client.

**Implementation**

**Setup Vue.js Project**

Open cmd at the folder you want to save Project folder, run command:  
vue create vue-js-client-crud

You will see some options, choose **default (babel, eslint)**.  
After the process is done. We create new folders and files like the following tree:

 public

 index.html

 src

 components

 AddTutorial.vue

 Tutorial.vue

 TutorialsList.vue

 services

 TutorialDataService.js

 App.vue

 main.js

 package.json

Open **public**/*index.html*, add bootstrap inside <head> tag:

<!DOCTYPE html>

<html lang="en">

<head>

...

<title>vue-js-client-crud</title>

<link type="text/css" rel="stylesheet" href="//unpkg.com/bootstrap/dist/css/bootstrap.min.css" />

</head>

<body>

...

</body>

</html>

**Add Vue Router to Vue.js 2 CRUD App**

– Run the command: npm install vue-router.

– In **src** folder, create *router.js* and define Router as following code:

import Vue from "vue";

import Router from "vue-router";

Vue.use(Router);

export default new Router({

mode: "history",

routes: [

{

path: "/",

alias: "/tutorials",

name: "tutorials",

component: () => import("./components/TutorialsList")

},

{

path: "/tutorials/:id",

name: "tutorial-details",

component: () => import("./components/Tutorial")

},

{

path: "/add",

name: "add",

component: () => import("./components/AddTutorial")

}

]

});

– Open **src/***main.js*, then import router:

import Vue from 'vue'

import App from './App.vue'

import router from './router'

Vue.config.productionTip = false

new Vue({

router,

render: h => h(App),

}).$mount('#app')

**Add Navbar and Router View to Vue.js 2 CRUD App**

Let’s open **src**/*App.vue*, this App component is the root container for our application, it will contain a navbar.

<template>

<div id="app">

<nav class="navbar navbar-expand navbar-dark bg-dark">

<router-link to="/" class="navbar-brand">bezKoder</router-link>

<div class="navbar-nav mr-auto">

<li class="nav-item">

<router-link to="/tutorials" class="nav-link">Tutorials</router-link>

</li>

<li class="nav-item">

<router-link to="/add" class="nav-link">Add</router-link>

</li>

</div>

</nav>

<div class="container mt-3">

<router-view />

</div>

</div>

</template>

<script>

export default {

name: "app"

};

</script>

**Initialize Axios for Vue.js 2 CRUD HTTP Client**

Now we’re gonna install *axios* with command: npm install axios.  
Then, under **src** folder, we create *http-common.js* file like this:

import axios from "axios";

export default axios.create({

baseURL: "http://localhost:8080/api",

headers: {

"Content-type": "application/json"

}

});

Remember to change the baseURL, it depends on REST APIs url that your Server configures.

**Create Data Service**

Our service will use axios from HTTP client above to send HTTP requests.

**services**/*TutorialDataService.js*

import http from "../http-common";

class TutorialDataService {

getAll() {

return http.get("/tutorials");

}

get(id) {

return http.get(`/tutorials/${id}`);

}

create(data) {

return http.post("/tutorials", data);

}

update(id, data) {

return http.put(`/tutorials/${id}`, data);

}

delete(id) {

return http.delete(`/tutorials/${id}`);

}

deleteAll() {

return http.delete(`/tutorials`);

}

findByTitle(title) {

return http.get(`/tutorials?title=${title}`);

}

}

export default new TutorialDataService();

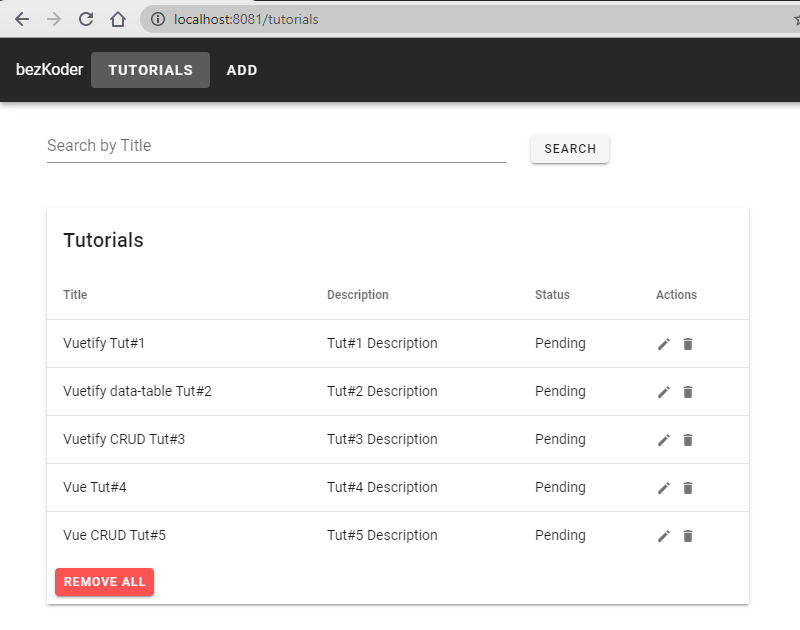
**Create Vue Components**

As I’ve said before, we have 3 components corresponding to 3 routes defined in Vue Router.

* Add new Item
* List of items
* Item details

You can continue with step by step to implement this Vue App in the post:  
[Vue.js CRUD App with Vue Router & Axios](https://bezkoder.com/vue-js-crud-app/)

Or using Vuetify: [Vuetify data-table example with a CRUD App](https://bezkoder.com/vuetify-data-table-example/)



**Run Vue.js Client**

You can run our Vue App with command: npm run serve.  
If the process is successful, open Browser with Url: http://localhost:8081/ and check it.

**Conclusion**

Now we have an overview of Vue.js + Node.js Express + PostgreSQL example when building a full-stack CRUD App.

We also take a look at client-server architecture for REST API using Express & Sequelize ORM, as well as Vue.js project structure for building a front-end app to make HTTP requests and consume responses.

Next tutorials show you more details about how to implement the system:  
– [Back-end](https://bezkoder.com/node-express-sequelize-postgresql/)  
– Front-end:

* [Vue 2](https://bezkoder.com/vue-js-crud-app/)
* [Vue 3](https://bezkoder.com/vue-3-crud/)

Run both projects (back-end & front-end) in one place:  
[How to serve/combine Vue App with Express](https://bezkoder.com/serve-vue-app-express/)

If you want a Typescript version for the Vue App, it is here:  
[Vue Typescript CRUD Application to consume Web API example](https://bezkoder.com/vue-typescript-crud/)

Serverless with Firebase:  
– [Vue Firebase Realtime Database: CRUD example](https://bezkoder.com/vue-firebase-realtime-database/)  
– [Vue Firestore: Build a CRUD App example](https://bezkoder.com/vue-firestore-crud/)

Happy learning, see you again!