Developing and Building MEVN Stack

Jagadeesh Kancharana

Abstract: ***This article provides a comprehensive guide to developing and deploying a MEVN stack application, using a task management project as an example. It details the setup of essential tools and technologies like Node. js, Express, MongoDB, and Vue. js, and explores various methods to automate the build and deployment processes, including the use of Webpack, Gulp, and Docker. The article aims to streamline the development process and ensure the successful deployment of scalable web applications. This paper provides a comprehensive guide to building a MEVN stack application and deploying it in a production environment. The guide covers the essential steps involved in development, packaging, and deployment, highlighting various methods to streamline and automate the processes. The development section begins with setting up the required prerequisites, including Node. js, Express, Mongoose, MongoDB, and various tools such as Vue CLI and Postman. An example project, specifically a simple task management application, is used to demonstrate the practical implementation of the stack. This project involves creating, retrieving, editing, and deleting tasks through API calls to a Node. js server, storing data in MongoDB. Several approaches for building and packaging the application are explored. The manual implementation section walks through the process of manually building the Vue. js application, placing the static assets in a production - ready directory, and running the Node. js server. However, this method is time - consuming and prone to errors, leading to the exploration of more efficient techniques. The paper then introduces Webpack, a module bundler that simplifies the build process by combining the server code and signific assets into a single file. This approach reduces the need for manual installation of dependencies, but still requires some manual steps. To achieve full automation, the guide demonstrates how to use Gulp, a toolkit that automates repetitive tasks. By setting up a Gulpfile, developers can automate the entire build process, including cleaning directories, building the Vue. js code, bundling the server code, and zipping the final package. This method significantly improves productivity by eliminating manual steps and reducing the potential for errors. Finally, the paper discusses the use of Docker for packaging and deploying the MEVN stack application. Docker allows developers to containerize the application, creating an isolated environment that includes all dependencies and configurations. This container can then be deployed on various container platforms, ensuring consistency across different environments. The purpose of this article is to guide developers through the process of building, automating, and deploying a MEVN stack application for efficient web development.***

Keywords: MongoDB, Express. js, Vue. js, Node. js, Server - side, Type - Script, Bootstrap, MEVN stack, web development, Node. js, Vue. js, Docker

There are so many ways we can build VueJS apps and ship them for production. One way is to build the VueJS app with NodeJS and MongoDB as a database. The MEVN stack is popular because it allows developers to use JavaScript across all components. The four things are MongoDB, VueJS, Express, and NodeJS. This stack can be used for a lot of uses cases in web development.

Developing the application is one part and you need to package it based on your deployment needs once the development part is completed. There are so many ways we can package and ship

MEVN Stack to production: manual, with webpack, with Gulp, etc. This article explores these approaches in detail.
- Prerequisites
- Example Project
- MEVN Stack Development
- Manual Implementation
- With Webpack
- Packaging With Gulp
- With Docker
- Summary
- Conclusion

1. Prerequisites

There are some prerequisites for this post. You need to have a NodeJS installed on your machine and some other tools that are required to complete this project.
- NodeJS
- Express Framework
- Mongoose
- MongoDB
- VSCode
- Postman
- nodemon
- dotenv
- Vue CLI
- Typescript
- BootstrapVue
- gulp. js
- Docker

NodeJS: As an asynchronous event - driven JavaScript runtime, Node. js is designed to build scalable network applications.

Express Framework: Express is a minimal and flexible Node. js web application framework that provides a robust set of features for web and mobile applications.

Mongoose: elegant MongoDB object modeling for node. js

MongoDB: MongoDB is a general - purpose, document - based, distributed database built for modern application developers and for the cloud era.

VSCode: The editor we are using for the project. It's open - source and you can download it here.

Postman: Manual testing your APIs
nodemon: To speed up the development

If you are a complete beginner and don't know how to build from scratch, I would recommend going through the below articles. We used these projects from this article as a basis for this post.

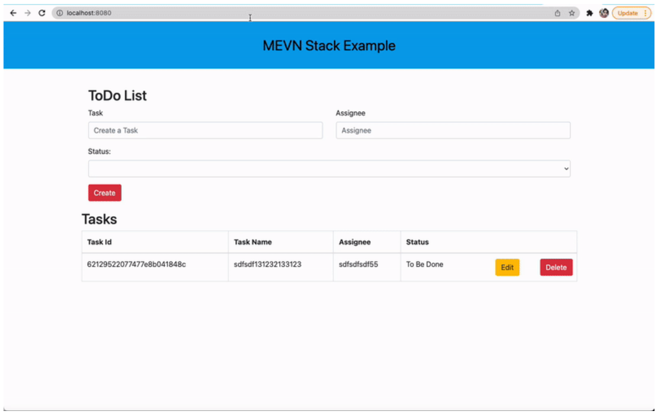
How To Develop and Build Vue. js App with NodeJS

How to Build NodeJS REST API with Express and MongoDB

How to write production - ready Node. js Rest API — Javascript version

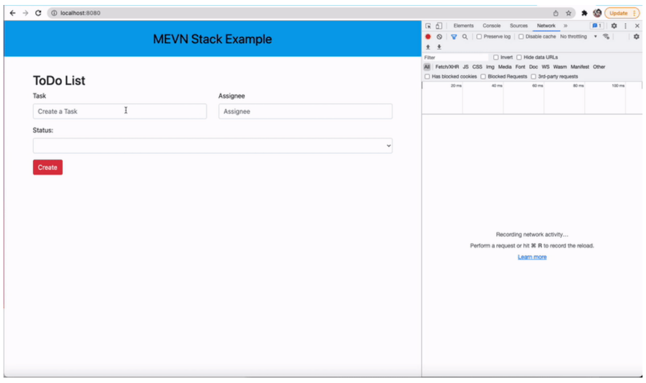
2. Example Project

Here is an example of a simple tasks application that creates, retrieves, edits, and deletes tasks. We actually run the API on the NodeJS server and you can use MongoDB to save all these tasks.



Task Id	Task Name	Assignee	Status
62129522077477e8b041848c	sdfsdf131232133123	sdfsdfsdf55	To Be Done

As you add users we are making an API call to the nodejs server to store them and get the same data from the server when we retrieve them. You can see network calls in the following video.



Network Calls
Here is a Github link to this project. You can clone it and run it on your machine.

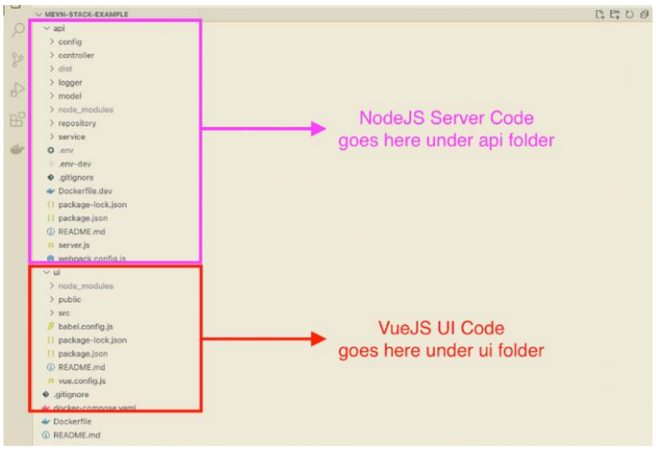
// clone the project
git clone https: //github. com/bbachi/mevn - stack - example. git

// Vue Code
cd ui
npm install
npm run serve

// API code
cd api
npm install
npm run dev

3. Project Structure

Let's understand the project structure for this project. We will have two packages. json: one for the VueJS and another for nodejs API. It's always best practice to have completely different node_modules for each one. In this way, you won't get merging issues or any other problems regarding web and server node modules collision. It's easier to convert your MEVN Stack into any other stack later such as replacing the API code with microservices and serving your UI through the NGINX web server.



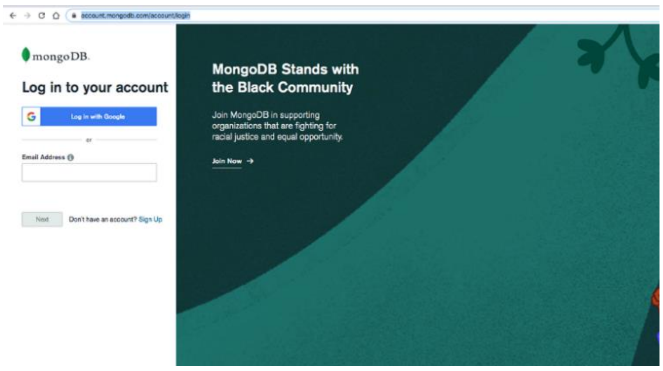
4. Project Structure

If you look at the above project structure, all the Vue app resides under the **ui** folder and nodejs API resides under the **api** folder.

1) Set up a MongoDB Atlas

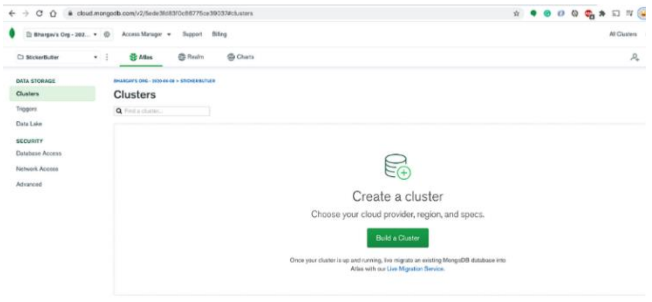
The core of MongoDB Cloud is MongoDB Atlas, a fully managed cloud database for modern applications. Atlas is the best way to run MongoDB, the leading modern database.

Let's create your MongoDB Account here. You can either log in with any of your Gmail accounts or you can provide any other email address to create the account.


account.mongodb.com/account/login
mongoDB.
Log in to your account
Log in with Google
or
Email Address
Next
Don't have an account? Sign Up
MongoDB Stands with the Black Community
Join MongoDB in supporting organizations that are fighting for racial justice and equal opportunity.
Join Now →

2) MongoDB Atlas login

Once you log in with your account you will see the dashboard below where you can create clusters.



Clusters

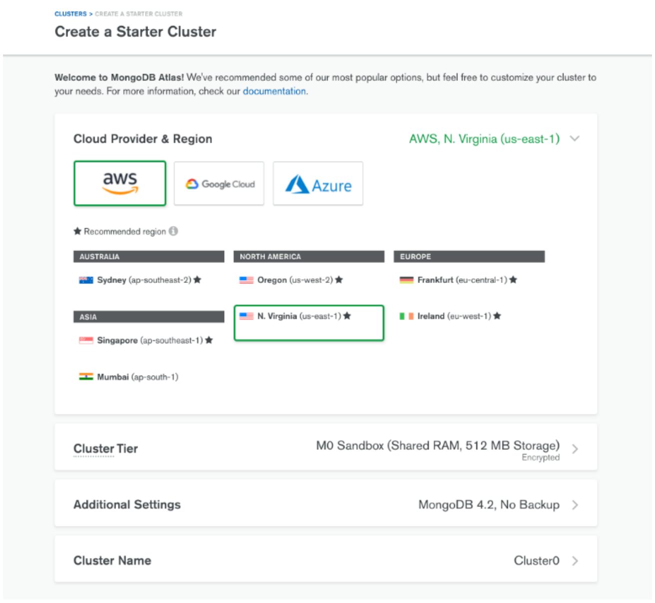
Create a cluster

Choose your cloud provider, region, and specs.

Build a Cluster

MongoDB Dashboard

Let's create a cluster called todo - cluster by clicking on the build a cluster and selecting all the details below.



CLUSTERS > CREATE A STARTER CLUSTER

Create a Starter Cluster

Welcome to MongoDB Atlas! We've recommended some of our most popular options, but feel free to customize your cluster to your needs. For more information, check our documentation.

Cloud Provider & Region — AWS, N. Virginia (us-east-1)

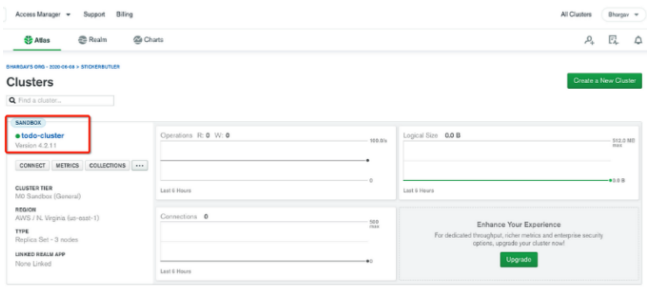
Cluster Tier — M0 Sandbox (Shared RAM, 512 MB Storage) Encrypted

Additional Settings — MongoDB 4.2, No Backup

Cluster Name — Cluster0

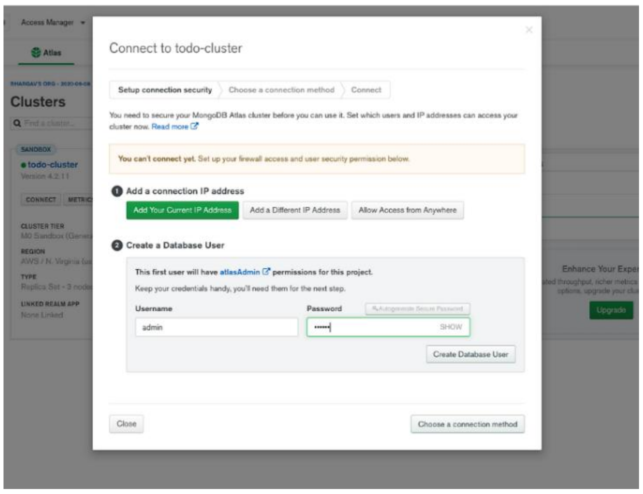
Creating a Cluster

Here is the cluster we created below.



todo - cluster

You can click on the connect button to see the details about connecting to the cluster. You need to create a user and Allow Access from anywhere for now.



Connect to todo-cluster

Setup connection security › Choose a connection method › Connect

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. Read more

You can't connect yet. Set up your firewall access and user security permission below.

1 Add a connection IP address

Add Your Current IP Address | Add a Different IP Address | Allow Access from Anywhere

2 Create a Database User

This first user will have atlasAdmin permissions for this project.

Keep your credentials handy, you'll need them for the next step.

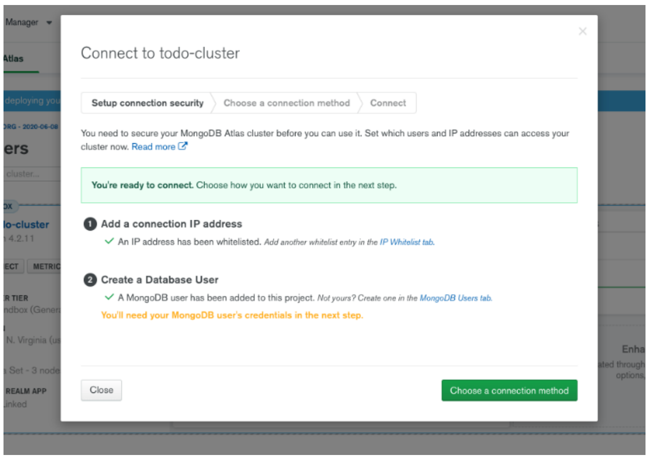
Username: admin

Password: ••••••

Create Database User

Close | Choose a connection method

Connecting to cluster



Connect to todo-cluster

Setup connection security › Choose a connection method › Connect

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. Read more

You're ready to connect. Choose how you want to connect in the next step.

1 Add a connection IP address

✓ An IP address has been whitelisted. *Add another whitelist entry in the IP Whitelist tab.*

2 Create a Database User

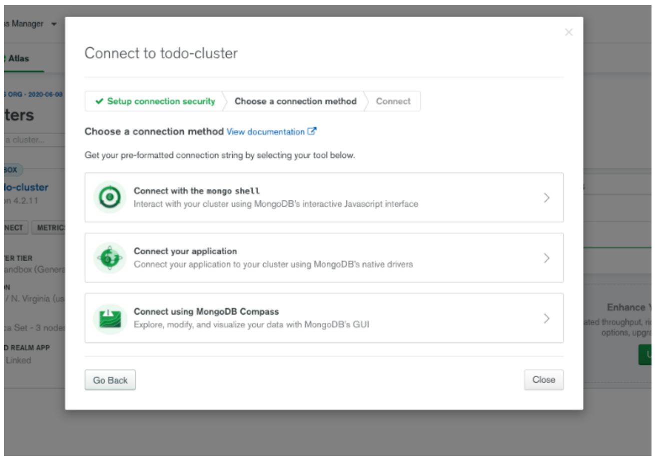
✓ A MongoDB user has been added to this project. *Not yours? Create one in the MongoDB Users tab.*

You'll need your MongoDB user's credentials in the next step.

Close | Choose a connection method

Connecting to cluster

You can see three ways of connecting to the cluster on the next screen.



Connect to todo-cluster

✓ Setup connection security › Choose a connection method › Connect

Choose a connection method View documentation

Get your pre-formatted connection string by selecting your tool below.

Connect with the `mongo shell` — Interact with your cluster using MongoDB's interactive Javascript interface

Connect your application — Connect your application to your cluster using MongoDB's native drivers

Connect using MongoDB Compass — Explore, modify, and visualize your data with MongoDB's GUI

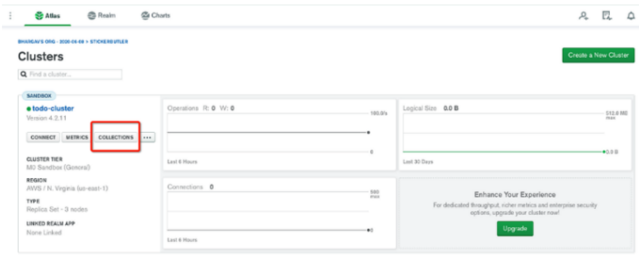
Go Back | Close

Ways of connecting

We will see all these three ways to connect to the cluster in the next sections.

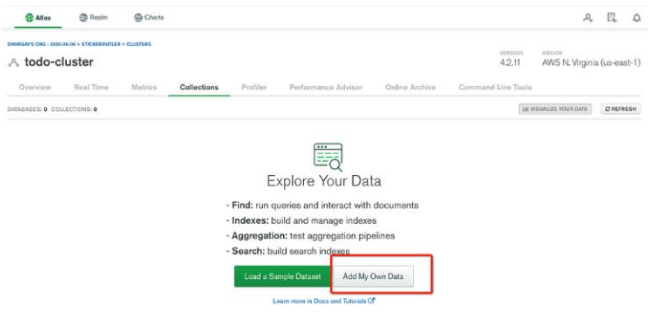
Create a Database

We have created a cluster and it's time to create a database. Click on the collections to create a new database as below.



Collections

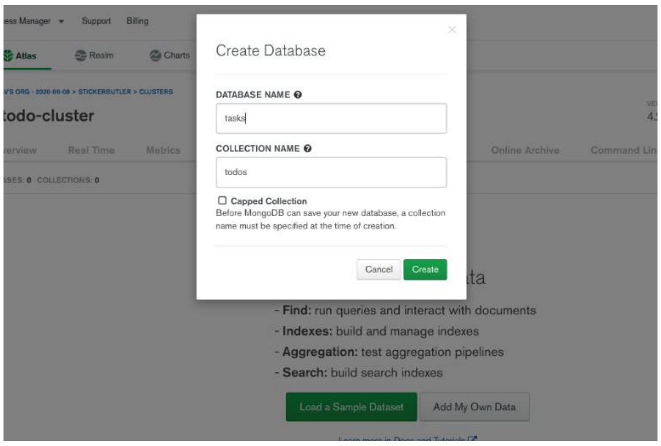
Click on the Add My Own Data Button to create a new database.



- Find: run queries and interact with documents
- Indexes: build and manage indexes
- Aggregation: test aggregation pipelines
- Search: build search indexes

Add My Own Data

I have given a database name as tasks and the collection name is todos.



Create Database

DATABASE NAME: tasks

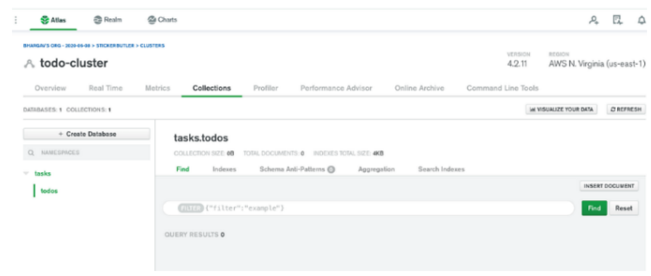
COLLECTION NAME: todos

☐ Capped Collection

Before MongoDB can save your new database, a collection name must be specified at the time of creation.

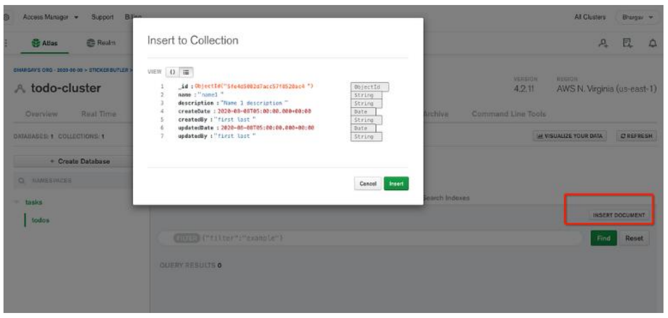
Creating a Database

You will see the below dashboard once the database is created. We have a database with empty collections.



Empty Collection

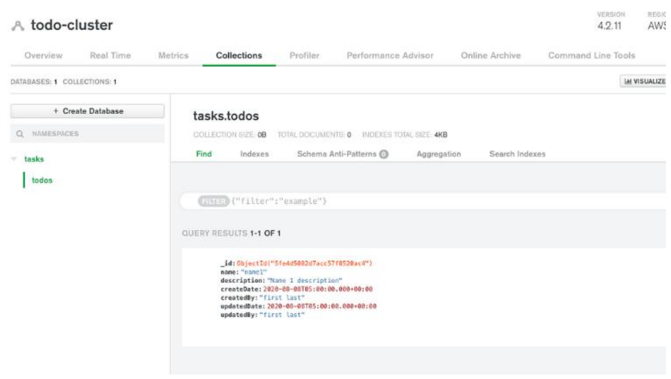
Let's insert the first document into the collection by clicking the button insert document



Insert to Collection

```
_id : ObjectId("5fe4d5082d7acc57f8520ac4 ")       ObjectId
name :"name1 "                                     String
description :"Name 1 description "                 String
createDate : 2020-08-08T05:00:00.000+00:00         Date
createdBy :"first last "                           String
updatedDate : 2020-08-08T05:00:00.000+00:00        Date
updatedBy :"first last "                           String
```

Inserting the Document



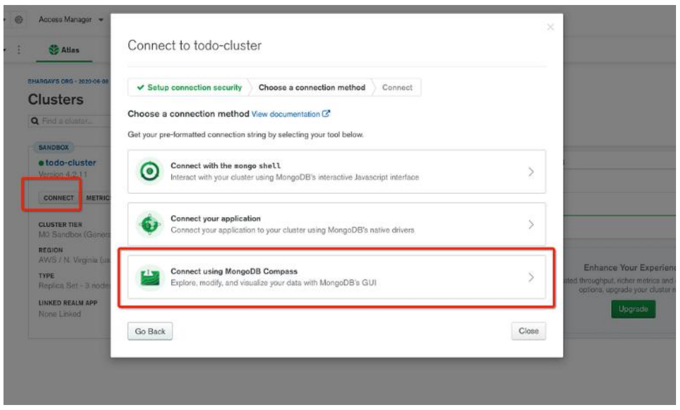
```
_id: ObjectId("5fe4d5082d7acc57f8520ac4")
name: "name1"
description: "Name 1 description"
createDate: 2020-08-08T05:00:00.000+00:00
createdBy: "first last"
updatedDate: 2020-08-08T05:00:00.000+00:00
updatedBy: "first last"
```

Document Inserted

1) Connect With Mongo Compass

We have seen three ways we can connect to this cluster and read the collections. Let's connect to the database with Mongo Compass. The first thing we need to do is to download and install Mongo Compass from this link.

Let's get a connection string from the Atlas dashboard as below.



Connect to todo-cluster

Setup connection security › Choose a connection method › Connect

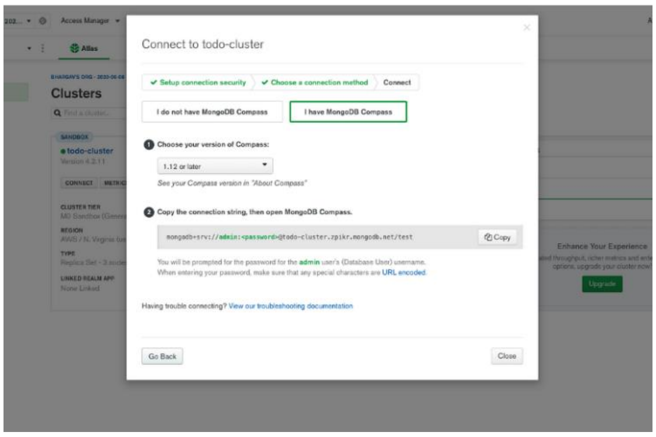
Choose a connection method View documentation

Get your pre-formatted connection string by selecting your tool below.

- Connect with the mongo shell — Interact with your cluster using MongoDB's interactive Javascript interface
- Connect your application — Connect your application to your cluster using MongoDB's native drivers
- Connect using MongoDB Compass — Explore, modify, and visualize your data with MongoDB's GUI

Connect with MongoDB Compass

Replace the password with the password that you created above.



Connect to todo-cluster

✓ Setup connection security ✓ Choose a connection method Connect

I do not have MongoDB Compass | I have MongoDB Compass

1 Choose your version of Compass:

1.12 or later

See your Compass version in "About Compass"

2 Copy the connection string, then open MongoDB Compass.

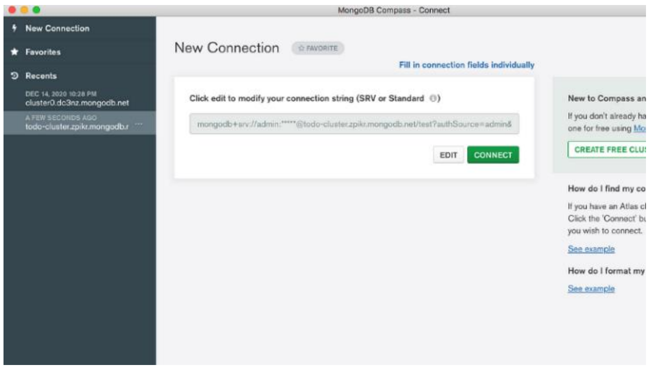
```
mongodb+srv://admin:<password>@todo-cluster.zpikr.mongodb.net/test
```

You will be prompted for the password for the admin user's (Database User) username.
When entering your password, make sure that any special characters are URL encoded.

Having trouble connecting? View our troubleshooting documentation

Connection String

Let’s connect to the database with the connection string



MongoDB Compass - Connect

New Connection

Click edit to modify your connection string (SRV or Standard)

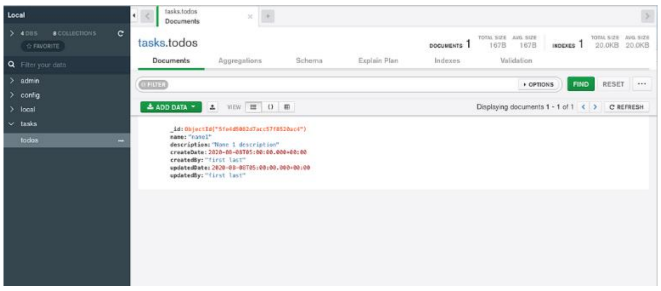
```
mongodb+srv://admin:*****@todo-cluster.zpikr.mongodb.net/test?authSource=admin&
```

DEC 14, 2020 10:28 PM
cluster0.dc3nz.mongodb.net

A FEW SECONDS AGO
todo-cluster.zpikr.mongodb.r

Connect with Connection String

You can actually see the same collection in the MongoDB Compass as well.



tasks.todos

```
_id: ObjectId("5fe4d5082d7acc57f8520ac4")
name: "name1"
description: "Name 1 description"
createDate: 2020-08-08T05:00:00.000+00:00
createdBy: "first last"
updatedDate: 2020-08-08T05:00:00.000+00:00
updatedBy: "first last"
```

MongoDB Compass

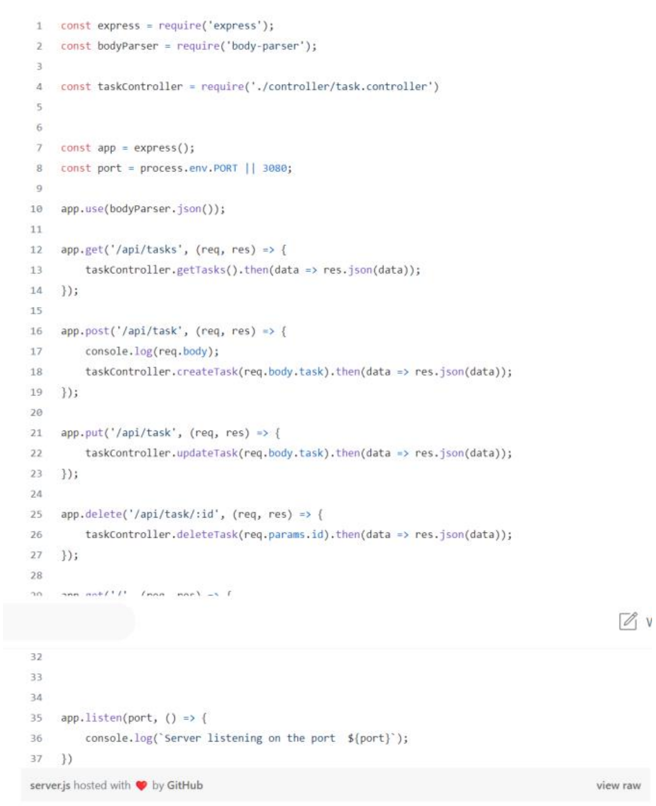
2) Building API

We have configured MongoDB in the previous section, it’s time to build the API. I would recommend you go through two articles posted in the prerequisites section. Let me put those here as well.

How to Build NodeJS REST API with Express and MongoDB

How to write production - ready Node. js Rest API — Javascript version

The starting point of the API is the ***server. js*** file in which we define all the routes and import the express. Here is the file where the nodejs server runs on port 3080 and starts listening for the incoming requests.



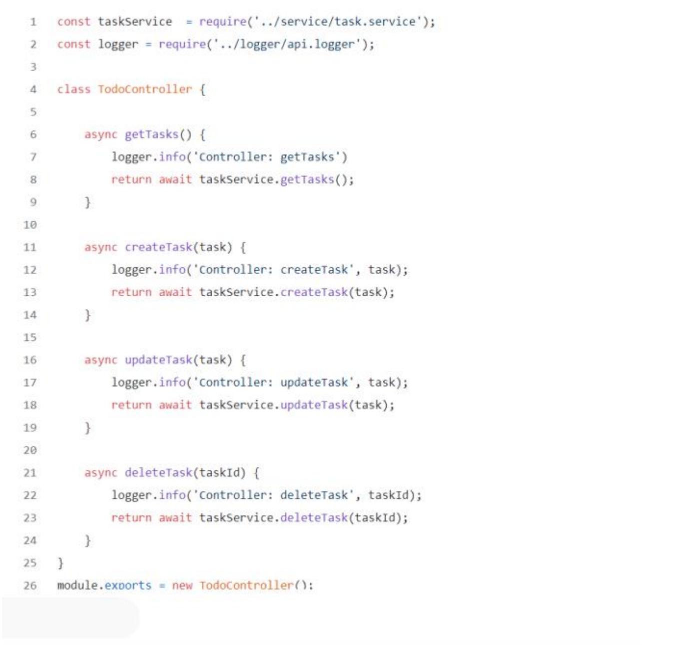
```
const express = require('express');
const bodyParser = require('body-parser');

const taskController = require('./controller/task.controller')


const app = express();
const port = process.env.PORT || 3080;

app.use(bodyParser.json());

app.get('/api/tasks', (req, res) => {
    taskController.getTasks().then(data => res.json(data));
});

app.post('/api/task', (req, res) => {
    console.log(req.body);
    taskController.createTask(req.body.task).then(data => res.json(data));
});

app.put('/api/task', (req, res) => {
    taskController.updateTask(req.body.task).then(data => res.json(data));
});

app.delete('/api/task/:id', (req, res) => {
    taskController.deleteTask(req.params.id).then(data => res.json(data));
});

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



app.listen(port, () => {
    console.log(`Server listening on the port  ${port}`);
})
```

server.js hosted with ❤ by GitHub view raw

server.js

We have defined 4 routes for CRUD operations. Notice that we are using four different HTTP methods for creating, updating, reading, and deleting operations. The request comes to these routes and each route calls the respective method in the controller class. You can read the body of the incoming requests in the req object defined in each route. The result of these methods is promise based so you need to use ***then*** method to read and send back to the client with the method res. json ().

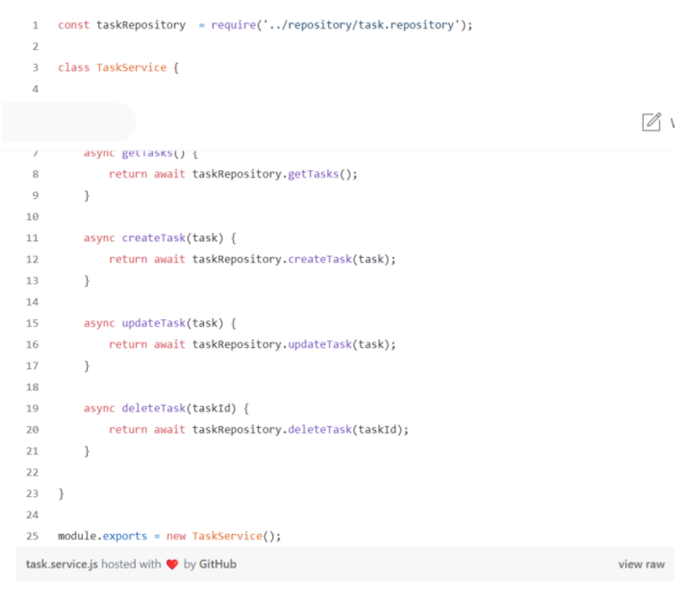
Here is the controller class in which we are calling the service class with async/await. The async/await is the cleaner way of reading promises. You don’t need async/await here since we are directly returning the result of the service class.



```
const taskService  = require('../service/task.service');
const logger = require('../logger/api.logger');

class TodoController {

    async getTasks() {
        logger.info('Controller: getTasks')
        return await taskService.getTasks();
    }

    async createTask(task) {
        logger.info('Controller: createTask', task);
        return await taskService.createTask(task);
    }

    async updateTask(task) {
        logger.info('Controller: updateTask', task);
        return await taskService.updateTask(task);
    }

    async deleteTask(taskId) {
        logger.info('Controller: deleteTask', taskId);
        return await taskService.deleteTask(taskId);
    }
}
module.exports = new TodoController();
```

Task Controller

Let’s look at the service class in which we call the repository to interact with the MongoDB data.



```
const taskRepository  = require('../repository/task.repository');

class TaskService {

    async getTasks() {
        return await taskRepository.getTasks();
    }

    async createTask(task) {
        return await taskRepository.createTask(task);
    }

    async updateTask(task) {
        return await taskRepository.updateTask(task);
    }

    async deleteTask(taskId) {
        return await taskRepository.deleteTask(taskId);
    }

}

module.exports = new TaskService();
```

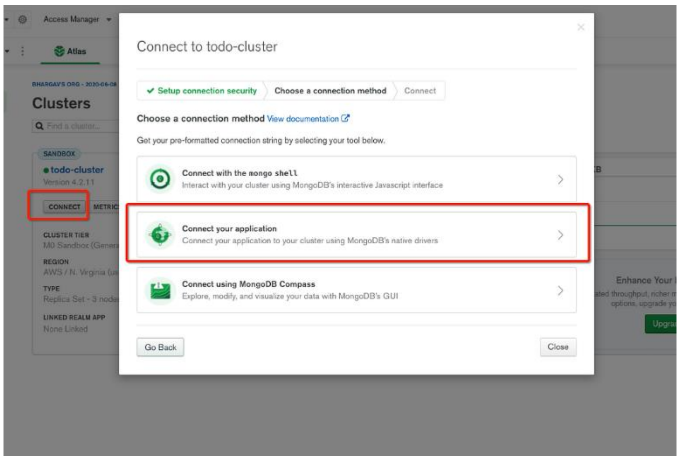
task.service.js hosted with ❤ by GitHub view raw

Task Service

You need to know how to configure Mongo Connection in the NodeJS before looking at the repository so that you can read the data from MongoDB. Let’s find that out in the following section.

3) Configure MongoDB In API

Let’s configure the Mongo Client from our application. The first thing we need to do is to get the connection string.



Connect to todo-cluster

✓ Setup connection security | Choose a connection method | Connect

Choose a connection method View documentation

Get your pre-formatted connection string by selecting your tool below.

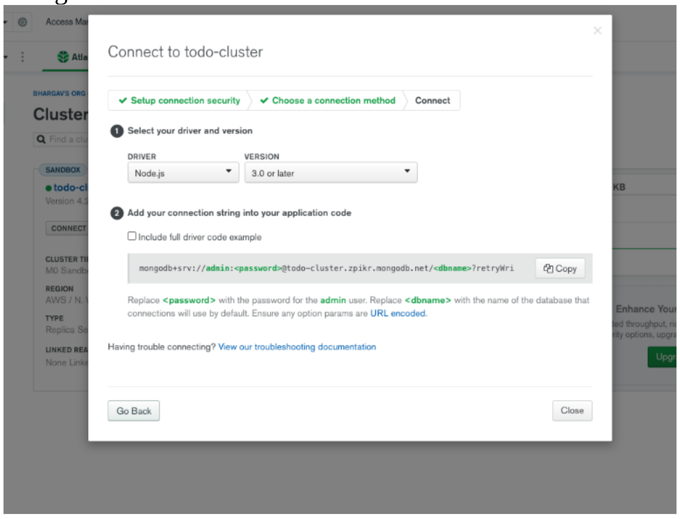
Connect with the mongo shell
Interact with your cluster using MongoDB's interactive Javascript interface

Connect your application
Connect your application to your cluster using MongoDB's native drivers

Connect using MongoDB Compass
Explore, modify, and visualize your data with MongoDB's GUI

Connect your application

Make sure you select the right language and the right MongoDB driver version.



Connect to todo-cluster

✓ Setup connection security | ✓ Choose a connection method | Connect

1 Select your driver and version

DRIVER: Node.js
VERSION: 3.0 or later

2 Add your connection string into your application code

☐ Include full driver code example

```
mongodb+srv://admin:<password>@todo-cluster.zpikr.mongodb.net/<dbname>?retryWri
```

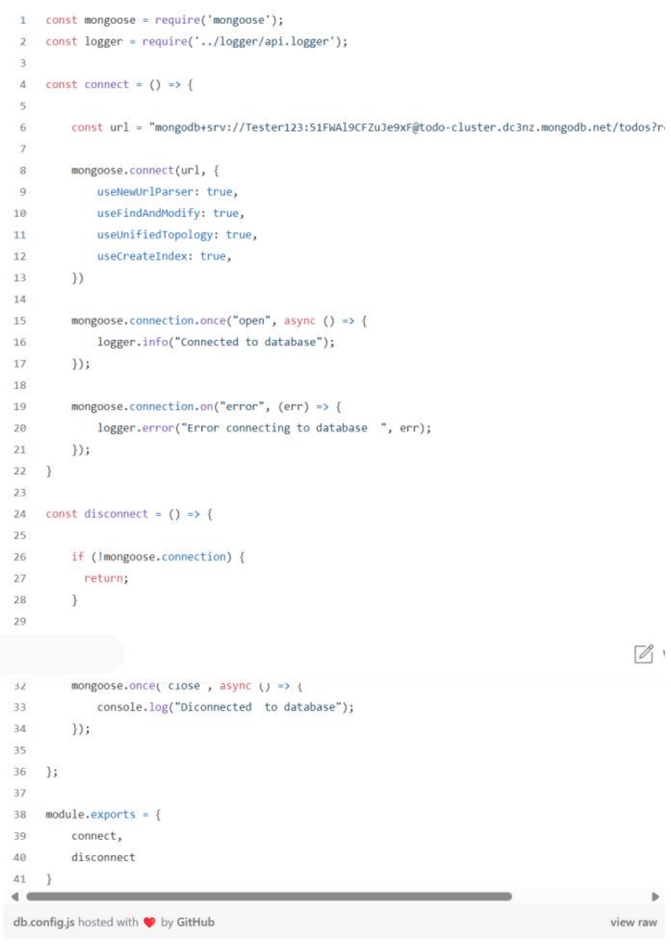
Replace <password> with the password for the admin user. Replace <dbname> with the name of the database that connections will use by default. Ensure any option params are URL encoded.

Having trouble connecting? View our troubleshooting documentation

Connection String

Let’s place the connection string and database name in the application properties file as below. You have to URL encode the password if you have any special characters in the password.

Here is the configuration file in which you connect to MongoDB with the help of the connection string. We are using Mongoose to connect with MongoDB for all the queries. Mongoose makes it easy for you to interact with MongoDB.

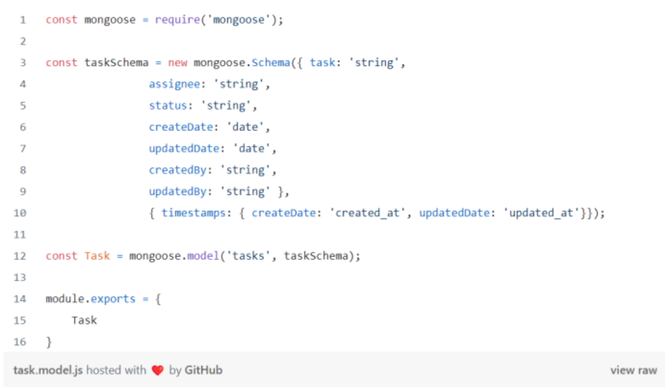


```
const mongoose = require('mongoose');
const logger = require('../logger/api.logger');

const connect = () => {

    const url = "mongodb+srv://Tester123:51FWAl9CFZuJe9xF@todo-cluster.dc3nz.mongodb.net/todos?r

    mongoose.connect(url, {
        useNewUrlParser: true,
        useFindAndModify: true,
        useUnifiedTopology: true,
        useCreateIndex: true,
    })

    mongoose.connection.once("open", async () => {
        logger.info("Connected to database");
    });

    mongoose.connection.on("error", (err) => {
        logger.error("Error connecting to database  ", err);
    });
}

const disconnect = () => {

    if (!mongoose.connection) {
      return;
    }

    mongoose.once("close", async () => {
        console.log("Diconnected  to database");
    });

};

module.exports = {
    connect,
    disconnect
}
```

db.config.js hosted with ❤ by GitHub view raw

DB Configuration File

The next thing we should define is the schema for the database model as below.

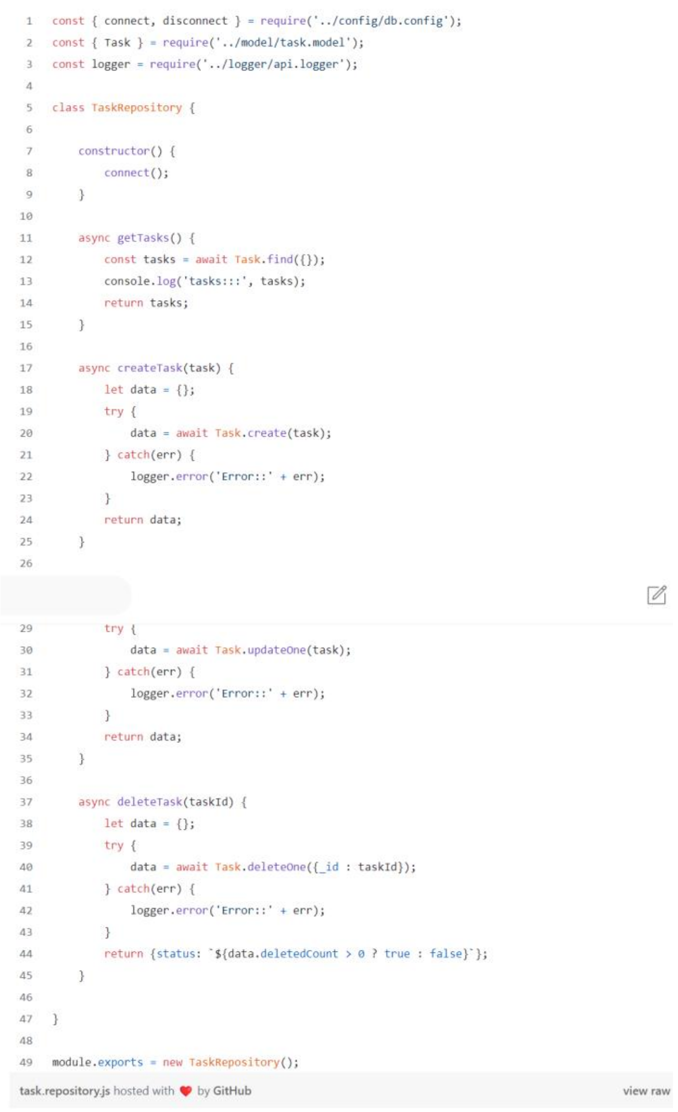


```
const mongoose = require('mongoose');

const taskSchema = new mongoose.Schema({ task: 'string',
                assignee: 'string',
                status: 'string',
                createDate: 'date',
                updatedDate: 'date',
                createdBy: 'string',
                updatedBy: 'string' },
                { timestamps: { createDate: 'created_at', updatedDate: 'updated_at'}});

const Task = mongoose.model('tasks', taskSchema);

module.exports = {
    Task
}
```

task.model.js hosted with ❤ by GitHub view raw

Task Model

Finally, we have a repository class as below using the above model for the CRUD operations.



```
const { connect, disconnect } = require('../config/db.config');
const { Task } = require('../model/task.model');
const logger = require('../logger/api.logger');

class TaskRepository {

    constructor() {
        connect();
    }

    async getTasks() {
        const tasks = await Task.find({});
        console.log('tasks:::', tasks);
        return tasks;
    }

    async createTask(task) {
        let data = {};
        try {
            data = await Task.create(task);
        } catch(err) {
            logger.error('Error::' + err);
        }
        return data;
    }

        try {
            data = await Task.updateOne(task);
        } catch(err) {
            logger.error('Error::' + err);
        }
        return data;
    }

    async deleteTask(taskId) {
        let data = {};
        try {
            data = await Task.deleteOne({_id : taskId});
        } catch(err) {
            logger.error('Error::' + err);
        }
        return {status: `${data.deletedCount > 0 ? true : false}`};
    }

}

module.exports = new TaskRepository();
```

task.repository.js hosted with ❤ by GitHub view raw

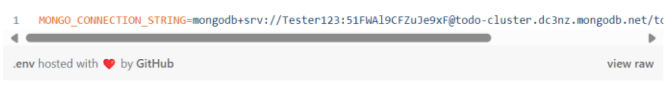
Task Repository

4) Externalize the Environment Variables

We have seen how to configure your MongoDB connection in the API. We need to store this kind of configuration outside of your app so that you can build once and deploy it in multiple environments with ease.

We need to use the **dotenv** library for environment - specific things. Dotenv is a zero - dependency module that loads environment variables from a. env file into process. env. Storing configuration in the environment separate from code is based on The Twelve - Factor App methodology.

The first step is to install this library npm install dotenv and put the. env file at the root location of the project.

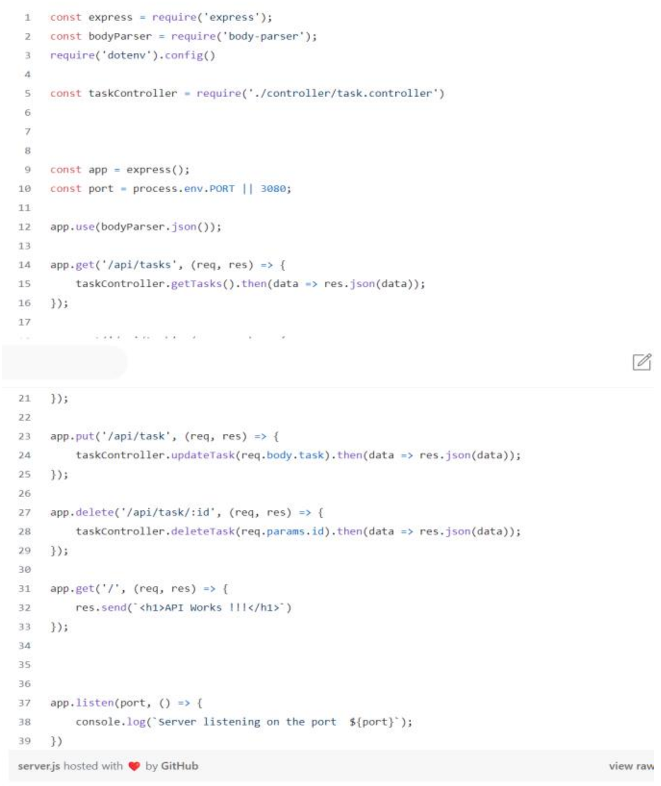


```
MONGO_CONNECTION_STRING=mongodb+srv://Tester123:51FWAl9CFZuJe9xF@todo-cluster.dc3nz.mongodb.net/tc
```

.env hosted with ❤ by GitHub view raw

.env file

We just need to put this line require ('dotenv'). config () as early as possible in the application code as in the server. js file.



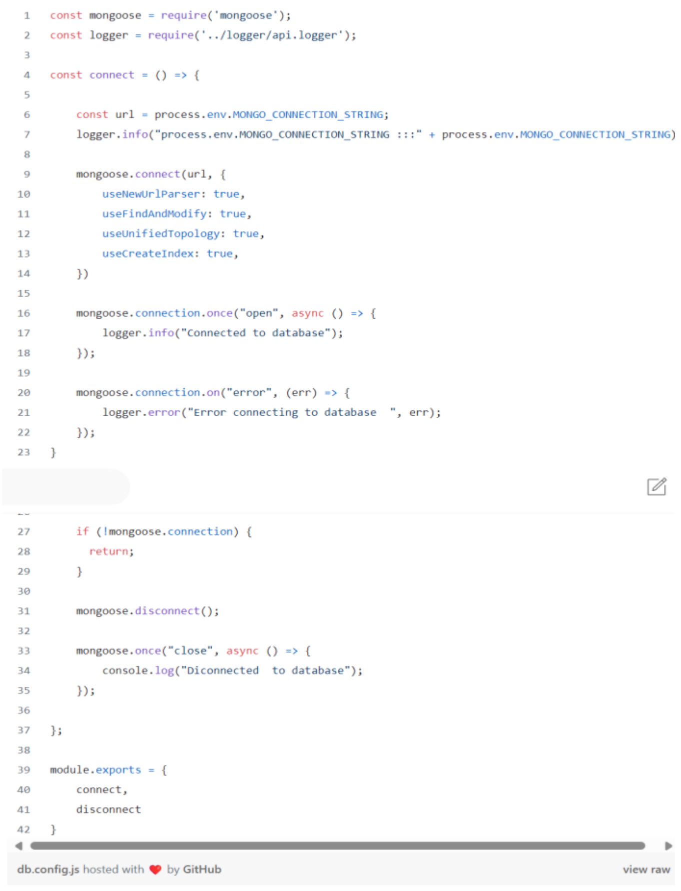
```
const express = require('express');
const bodyParser = require('body-parser');
require('dotenv').config()

const taskController = require('./controller/task.controller')



const app = express();
const port = process.env.PORT || 3080;

app.use(bodyParser.json());

app.get('/api/tasks', (req, res) => {
    taskController.getTasks().then(data => res.json(data));
});

```

```
});

app.put('/api/task', (req, res) => {
    taskController.updateTask(req.body.task).then(data => res.json(data));
});

app.delete('/api/task/:id', (req, res) => {
    taskController.deleteTask(req.params.id).then(data => res.json(data));
});

app.get('/', (req, res) => {
    res.send(`<h1>API Works !!!</h1>`)
});



app.listen(port, () => {
    console.log(`Server listening on the port  ${port}`);
})
```

server.js hosted with ❤ by GitHub view raw

server.js

Let's define the configuration class where it creates a connection with the connection string we just copied from the Atlas Dashboard. We are fetching the Mongo connection string with the dotenv library and connecting it to MongoDB with Mongoose. We are exposing two functions from this file connect and disconnect.



```
const mongoose = require('mongoose');
const logger = require('../logger/api.logger');

const connect = () => {

    const url = process.env.MONGO_CONNECTION_STRING;
    logger.info("process.env.MONGO_CONNECTION_STRING :::" + process.env.MONGO_CONNECTION_STRING)

    mongoose.connect(url, {
        useNewUrlParser: true,
        useFindAndModify: true,
        useUnifiedTopology: true,
        useCreateIndex: true,
    })

    mongoose.connection.once("open", async () => {
        logger.info("Connected to database");
    });

    mongoose.connection.on("error", (err) => {
        logger.error("Error connecting to database  ", err);
    });
}
```

```
    if (!mongoose.connection) {
      return;
    }

    mongoose.disconnect();

    mongoose.once("close", async () => {
        console.log("Diconnected  to database");
    });

};

module.exports = {
    connect,
    disconnect
}
```

db.config.js hosted with ❤ by GitHub view raw

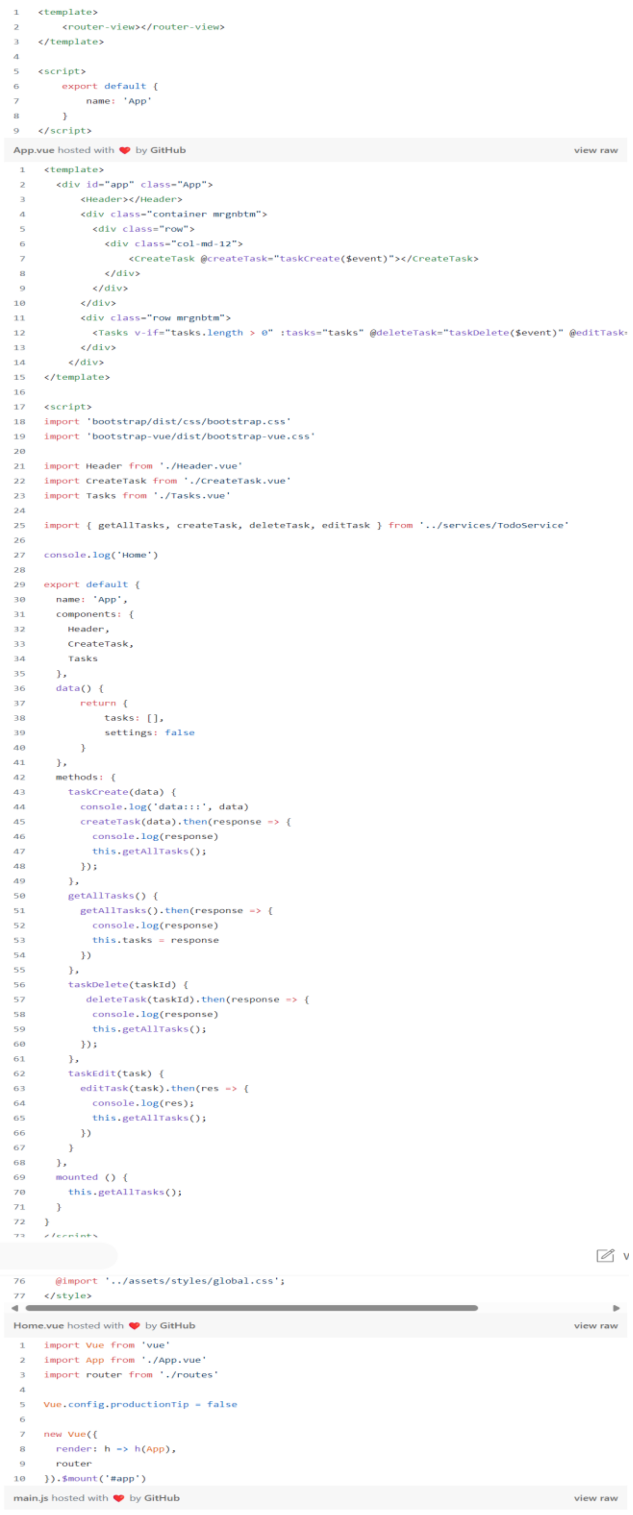
Configuration with dotenv

5) Building UI

Once you create the separate folder for the UI code you need to start with the following command to scaffold the Vue structure with the help of the Vue CLI Service. We will not build the entire app here instead we will go through important points here. You can clone the entire GitHub Repo and check the whole app.

```
vue create ui
```

Here are the **main. js, App. vue,** and **Home. vue** files for the app as the bootstrap components which means this is the first component that loads in the browser. You can import all the CSS - related files in the **Home. vue** file.



```
<template>
    <router-view></router-view>
</template>

<script>
    export default {
        name: 'App'
    }
</script>
```

App.vue hosted with ❤ by GitHub view raw

```
<template>
  <div id="app" class="App">
      <Header></Header>
      <div class="container mrgnbtm">
        <div class="row">
          <div class="col-md-12">
              <CreateTask @createTask="taskCreate($event)"></CreateTask>
          </div>
        </div>
      </div>
      <div class="row mrgnbtm">
        <Tasks v-if="tasks.length > 0" :tasks="tasks" @deleteTask="taskDelete($event)" @editTask
      </div>
    </div>
</template>

<script>
import 'bootstrap/dist/css/bootstrap.css'
import 'bootstrap-vue/dist/bootstrap-vue.css'

import Header from './Header.vue'
import CreateTask from './CreateTask.vue'
import Tasks from './Tasks.vue'

import { getAllTasks, createTask, deleteTask, editTask } from '../services/TodoService'

console.log('Home')

export default {
  name: 'App',
  components: {
    Header,
    CreateTask,
    Tasks
  },
  data() {
      return {
          tasks: [],
          settings: false
      }
  },
  methods: {
    taskCreate(data) {
      console.log('data:::', data)
      createTask(data).then(response => {
        console.log(response)
        this.getAllTasks();
      });
    },
    getAllTasks() {
      getAllTasks().then(response => {
        console.log(response)
        this.tasks = response
      })
    },
    taskDelete(taskId) {
       deleteTask(taskId).then(response => {
        console.log(response)
        this.getAllTasks();
      });
    },
    taskEdit(task) {
      editTask(task).then(res => {
        console.log(res);
        this.getAllTasks();
      })
    }
  },
  mounted () {
    this.getAllTasks();
  }
}
```

```
  @import '../assets/styles/global.css';
</style>
```

Home.vue hosted with ❤ by GitHub view raw

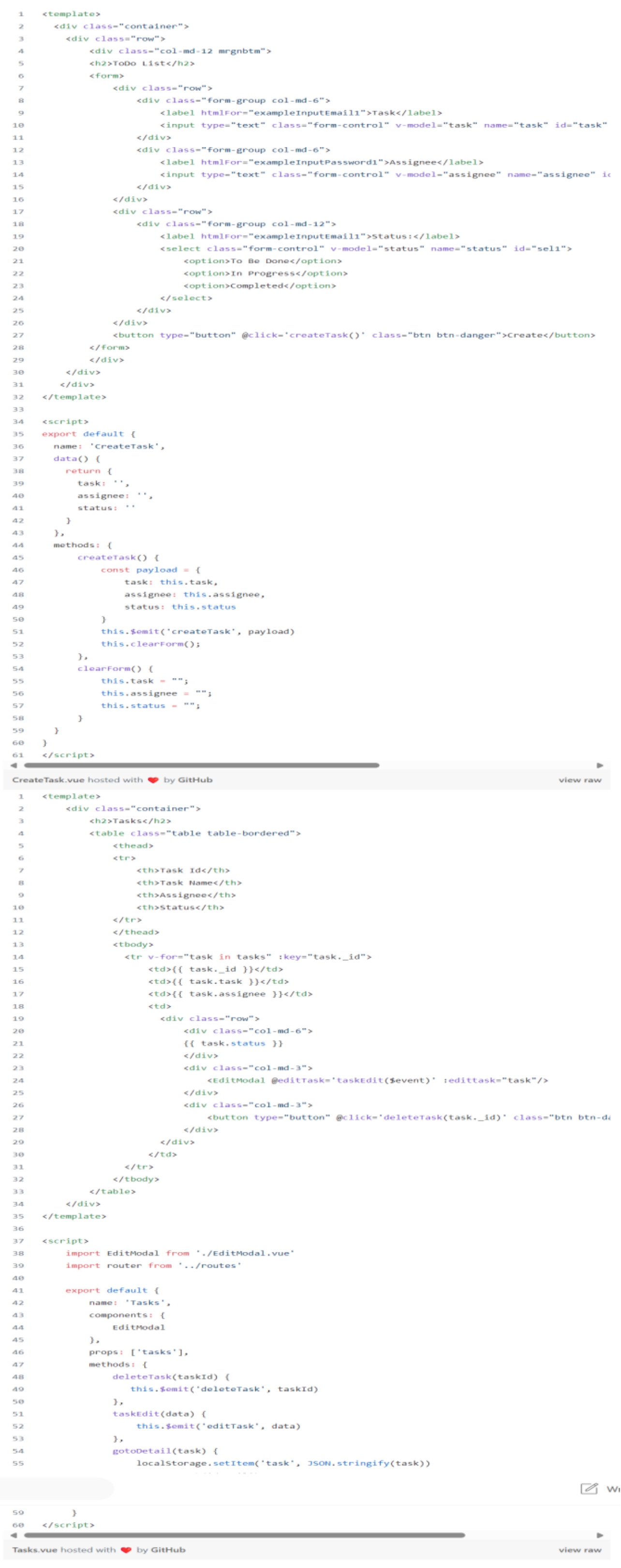
```
import Vue from 'vue'
import App from './App.vue'
import router from './routes'

Vue.config.productionTip = false

new Vue({
  render: h => h(App),
  router
}).$mount('#app')
```

main.js hosted with ❤ by GitHub view raw

App and Home Components

The App. vue component is the first component that loads since it is defined in the main. js file. We have a router defined in the App component that loads the Home component. This is a simple application where you add, update, delete tasks. You can go through the GitHub repo to check the rest of the files.

We have another two important components here one is for the createTask Form component and another is for the Tasks table.



```
<template>
  <div class="container">
    <div class="row">
        <div class="col-md-12 mrgnbtm">
        <h2>ToDo List</h2>
        <form>
            <div class="row">
                <div class="form-group col-md-6">
                    <label htmlFor="exampleInputEmail1">Task</label>
                    <input type="text" class="form-control" v-model="task" name="task" id="task"
                </div>
                <div class="form-group col-md-6">
                    <label htmlFor="exampleInputPassword1">Assignee</label>
                    <input type="text" class="form-control" v-model="assignee" name="assignee" ic
                </div>
            </div>
            <div class="row">
                <div class="form-group col-md-12">
                    <label htmlFor="exampleInputEmail1">Status:</label>
                    <select class="form-control" v-model="status" name="status" id="sel1">
                        <option>To Be Done</option>
                        <option>In Progress</option>
                        <option>Completed</option>
                    </select>
                </div>
            </div>
            <button type="button" @click='createTask()' class="btn btn-danger">Create</button>
        </form>
        </div>
    </div>
   </div>
</template>

<script>
export default {
  name: 'CreateTask',
  data() {
    return {
      task: '',
      assignee: '',
      status: ''
    }
  },
  methods: {
      createTask() {
          const payload = {
              task: this.task,
              assignee: this.assignee,
              status: this.status
          }
          this.$emit('createTask', payload)
          this.clearForm();
      },
      clearForm() {
          this.task = "";
          this.assignee = "";
          this.status = "";
      }
  }
}
</script>
```

CreateTask.vue hosted with ❤ by GitHub view raw

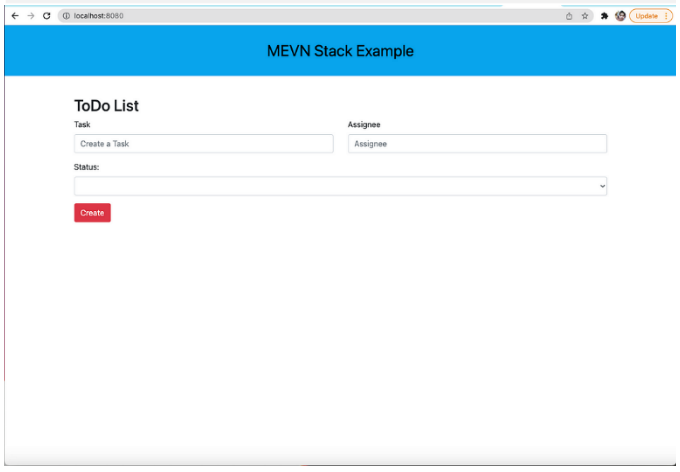
```
<template>
    <div class="container">
        <h2>Tasks</h2>
        <table class="table table-bordered">
            <thead>
            <tr>
                <th>Task Id</th>
                <th>Task Name</th>
                <th>Assignee</th>
                <th>Status</th>
            </tr>
            </thead>
            <tbody>
              <tr v-for="task in tasks" :key="task._id">
                  <td>{{ task._id }}</td>
                  <td>{{ task.task }}</td>
                  <td>{{ task.assignee }}</td>
                  <td>
                    <div class="row">
                        <div class="col-md-6">
                        {{ task.status }}
                        </div>
                        <div class="col-md-3">
                            <EditModal @editTask='taskEdit($event)' :edittask="task"/>
                        </div>
                        <div class="col-md-3">
                            <button type="button" @click='deleteTask(task._id)' class="btn btn-d
                        </div>
                    </div>
                  </td>
              </tr>
            </tbody>
        </table>
    </div>
</template>

<script>
    import EditModal from './EditModal.vue'
    import router from '../routes'

    export default {
        name: 'Tasks',
        components: {
            EditModal
        },
        props: ['tasks'],
        methods: {
            deleteTask(taskId) {
               this.$emit('deleteTask', taskId)
            },
            taskEdit(data) {
                this.$emit('editTask', data)
            },
            gotoDetail(task) {
                localStorage.setItem('task', JSON.stringify(task))

        }
</script>
```

Tasks.vue hosted with ❤ by GitHub view raw

Create Task and Tasks Components

Run the Vue code in local with the following command which runs on port **8080** on localhost. Make sure you are in the root folder of Vue code which is todo - app here.

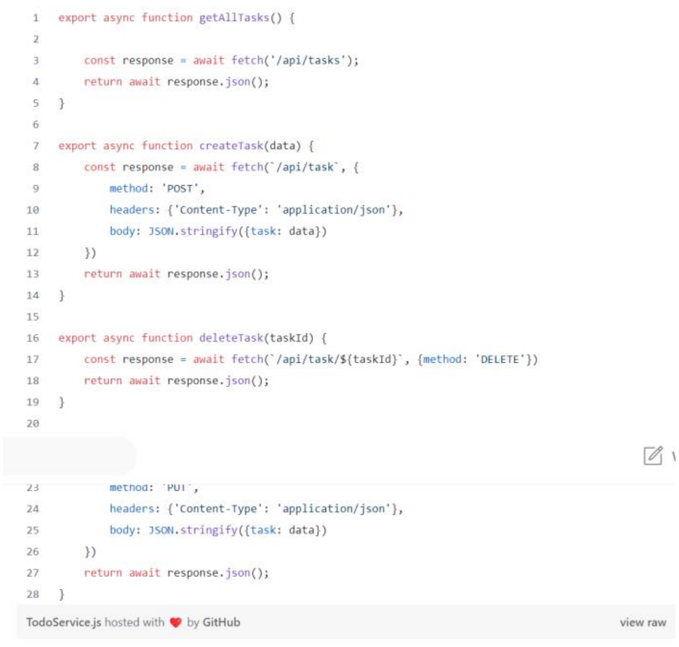
```
cd ui
npm run serve
```



Vue Code running on port 8080

6) Make API Calls From UI

Here is the service file which calls the API, in this case. We have four API operations to get, add, edit, and delete tasks with root path **/api.**

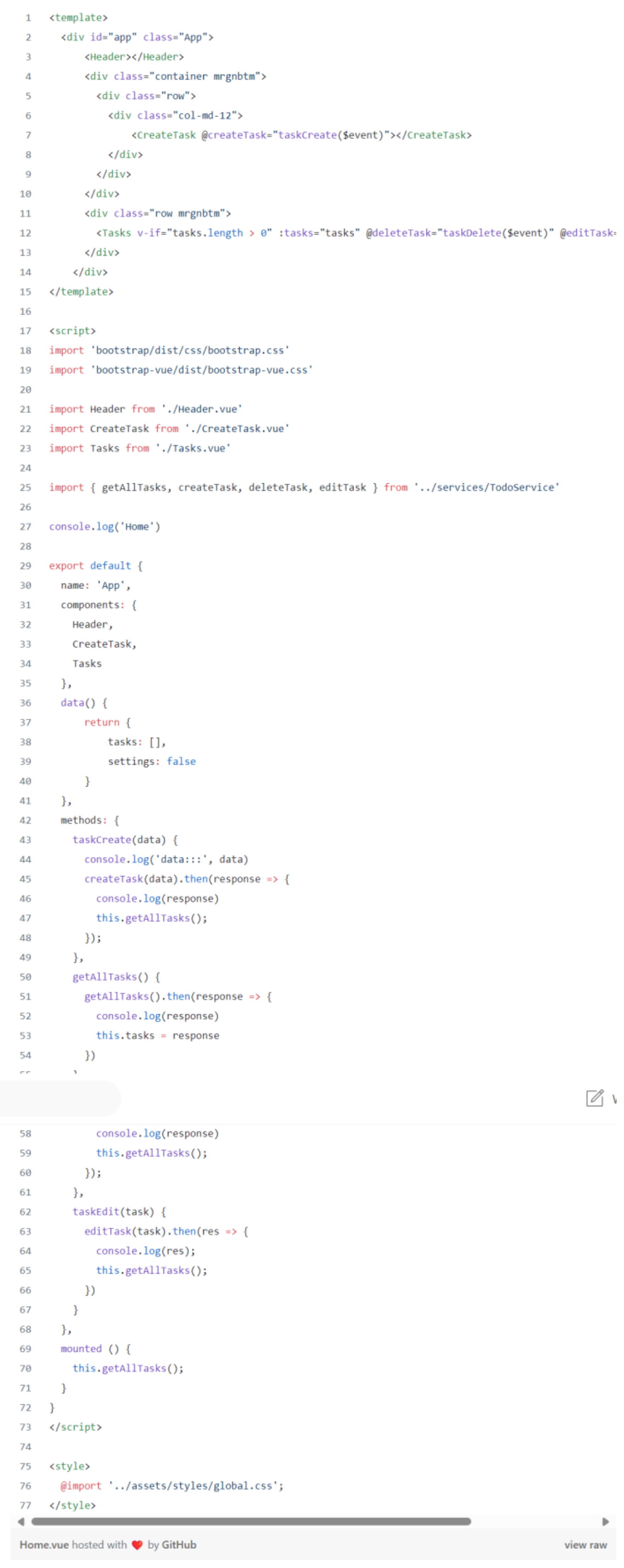


```
export async function getAllTasks() {

    const response = await fetch('/api/tasks');
    return await response.json();
}

export async function createTask(data) {
    const response = await fetch(`/api/task`, {
        method: 'POST',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify({task: data})
      })
    return await response.json();
}

export async function deleteTask(taskId) {
    const response = await fetch(`/api/task/${taskId}`, {method: 'DELETE'})
    return await response.json();
}

        method: 'PUT',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify({task: data})
      })
    return await response.json();
}
```

TodoService.js hosted with ❤ by GitHub view raw

TodoService.js

From the Vue components, you can call this service to get the data as below. Here is an example.



```
<template>
  <div id="app" class="App">
      <Header></Header>
      <div class="container mrgnbtm">
        <div class="row">
          <div class="col-md-12">
              <CreateTask @createTask="taskCreate($event)"></CreateTask>
          </div>
        </div>
      </div>
      <div class="row mrgnbtm">
        <Tasks v-if="tasks.length > 0" :tasks="tasks" @deleteTask="taskDelete($event)" @editTask=
      </div>
    </div>
</template>

<script>
import 'bootstrap/dist/css/bootstrap.css'
import 'bootstrap-vue/dist/bootstrap-vue.css'

import Header from './Header.vue'
import CreateTask from './CreateTask.vue'
import Tasks from './Tasks.vue'

import { getAllTasks, createTask, deleteTask, editTask } from '../services/TodoService'

console.log('Home')

export default {
  name: 'App',
  components: {
    Header,
    CreateTask,
    Tasks
  },
  data() {
      return {
          tasks: [],
          settings: false
      }
  },
  methods: {
    taskCreate(data) {
      console.log('data:::', data)
      createTask(data).then(response => {
        console.log(response)
        this.getAllTasks();
      });
    },
    getAllTasks() {
      getAllTasks().then(response => {
        console.log(response)
        this.tasks = response
      })

        console.log(response)
        this.getAllTasks();
      });
    },
    taskEdit(task) {
      editTask(task).then(res => {
        console.log(res);
        this.getAllTasks();
      })
    }
  },
  mounted () {
    this.getAllTasks();
  }
}
</script>

<style>
  @import '../assets/styles/global.css';
</style>
```

Home.vue hosted with ❤ by GitHub view raw

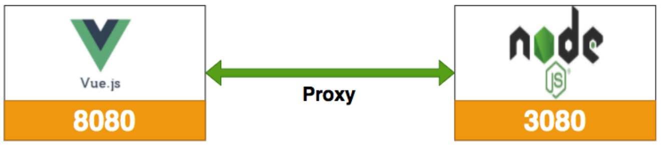
Home Component

You can look at the below article for a detailed post.
How To Make API calls in Vue. JS Applications

7) Development Environment Setup

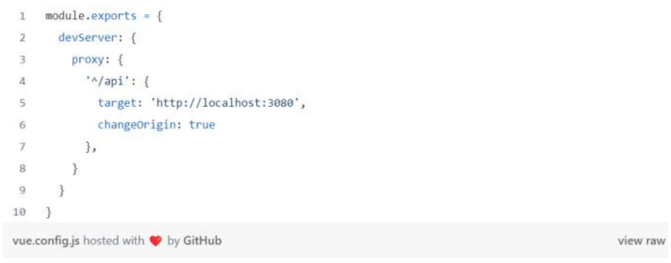
Usually, the way you develop and the way you build and run in production are completely different.

In the development phase, we run the nodejs server and the Vue app on completely **different ports.** It's easier and faster to develop that way. If you look at the following diagram the Vue app is running on port **8080** with the help of a webpack dev server and the nodejs server is running on port **3080**.



Development Environment

There should be some interaction between these two. We can proxy all the API calls to nodejs API. Vue - cli - service provides some inbuilt functionality and tells the development server to proxy any unknown requests to your API server in development, you need to add a **vue. proxy. js** file at the root of the location where package. json resides. We need to add the following file.



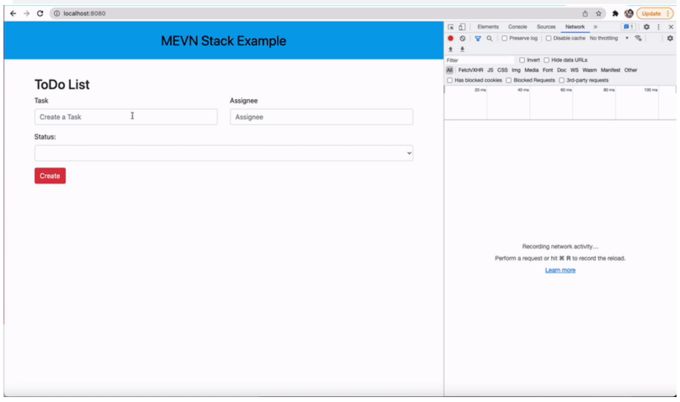
```
module.exports = {
  devServer: {
    proxy: {
      '^/api': {
        target: 'http://localhost:3080',
        changeOrigin: true
      },
    }
  }
}
```

vue.config.js hosted with ❤ by GitHub view raw

vue.config.js

Now you can run both Vue UI and NodeJS API on different ports and the Vue Code interacts with the API.

```
// Vue Code
cd ui
npm install
npm run serve

// API code
cd api
npm install
npm run dev
```



Network Calls

8) Running on Docker Compose

Docker Compose is really useful when we don't have the development environment setup on our local machine to run all parts of the application to test or we want to run all parts of the application with one command. For example, if you want to run NodeJS REST API and MongoDB database on different ports and need a single command to set up and run the whole thing. You can accomplish that with Docker Compose.

In the below post, we will see what is Docker Compose and how we can do the local development of MEVN Stack with Docker Compose, and its advantages as well.
Coming Soon!!
You can check out other stacks here
How To Run MERN Stack on Docker Compose
How To Run MEAN Stack on Docker Compose

9) Dockerize MEVN Stack

Docker is an enterprise - ready container platform that enables organizations to seamlessly build, share, and run any application, anywhere. Almost every company is containerizing its applications for faster production workloads so that they can deploy anytime and sometimes several times a day. There are so many ways we can build a MEVN Stack. One way is to dockerize it and create a docker image so that we can deploy that image any time or sometimes several times a day.

In the below post, we look at the example project and see the step - by - step guide on how we can dockerize the MEVN Stack.

Coming Soon !!
You can check out other stacks here
How To Dockerize MEAN Stack
How To Dockerize MERN Stack

a) Linting

We need to lint our project in that way it's easier to follow some standards in your project. We will see this in a separate article.
Coming Soon!!

b) Unit Testing API

There are so many tools out there to unit test your application such as Mocha, Chai, etc. We need a separate article for that to cover different libraries.
Coming Soon!!

c) Unit Testing UI

We will see how to unit test with UI with jest library.
Coming Soon!

d) Integration Tests

We will use cypress for the integration tests.
Coming Soon!

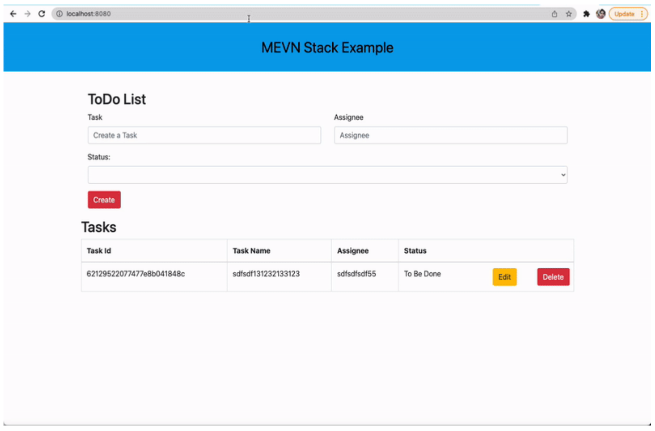
e) Build for production

We have to build the project for production in a different way. We can't use the proxy object. Here is the detailed article on how to package your app for production.
Packaging Your Vue. js App With NodeJS Backend For Production

How to Build MEVN Stack for Production

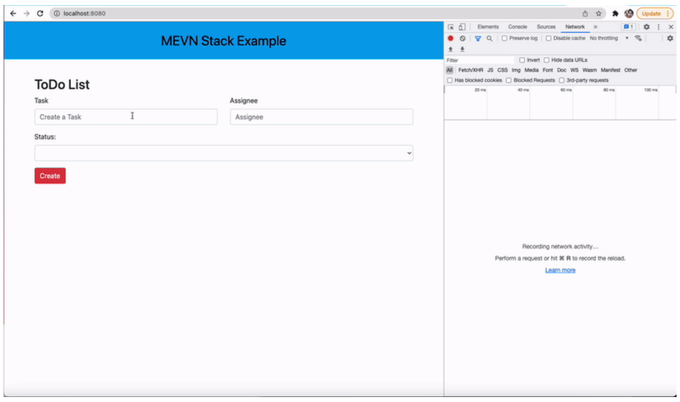
f) Demo

Here is an example of a simple tasks application that creates, retrieves, edits, and deletes tasks. We actually run the API on the NodeJS server and you can use MongoDB to save all these tasks.



Example Project

As you add users we are making an API call to the nodejs server to store them and get the same data from the server when we retrieve them. You can see network calls in the following video.



Network Calls

5. Summary

- There are so many ways we can build Vue apps and ship them for production.
- One way is to build the Vue app with NodeJS and MongoDB as a database. There are four things that make this stack popular and you can write everything in Javascript.
- The four things are MongoDB, VueJS, Express, and NodeJS. This stack can be used for a lot of uses cases in web development.
- We will have two package. json: one for the **VueJS** and another for **nodejs API**. It's always best practice to have completely different node_modules for each one.
- The core of MongoDB Cloud is MongoDB Atlas, a fully managed cloud database for modern applications. Atlas is the best way to run MongoDB, the leading modern database.
- We need to use the **dotenv** library for environment - specific things. Dotenv is a zero - dependency module that loads environment variables from a. env file into process. env. Storing configuration in the environment separate from code is based on The Twelve - Factor App methodology.
- In the development phase, we run the nodejs server and the **Vue** app on completely **different ports.** It's easier and faster to develop that way.
- We need to lint our project in that way it's easier to follow some standards in your project.

- There are so many tools out there to unit test the API such as Mocha, Chai, etc.
- We can unit test with UI with jest library.
- We will use cypress for the integration tests.
- We have to build the project for production in a different way. We can’t use the proxy object.

6. Conclusion

The MEVN stack provides a powerful framework for building modern web applications, offering flexibility and efficiency through JavaScript. This article has demonstrated the setup, development, and deployment of a MEVN stack application, highlighting various automation techniques. Future studies should explore further optimizations and deployment strategies across different cloud platforms to enhance scalability and performance.

7. References

[1] JavaScript Documentation https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics
[2] VueJS Documentation https://vuejs.org/guide/introduction.html
[3] Bachina, Bachina. (2024). *Ultimate Full Stack Web Development with MEVN*. OrangeAVA. Available at https://orangeava.com/collections/all-ebooks/products/ultimate-full-stack-web-development-with-mevn?variant=45552578560218
[4] Bhargav Bachina. (2021). Optimizing Deployment: React with NodeJS Backend on Azure App Services. Journal of Scientific and Engineering Research, 8(4), 218–227. https://doi.org/10.5281/zenodo.10902911