



UNIVERSIDAD
DE SANTIAGO
DE CHILE

Fundamentos Ingeniería en Software

Manual de supervivencia, Tarea 4



Francisco Núñez D.
16 de febrero del 2021

Manual creado para S.O. Ubuntu 18.04/20.04 LTS

ver 1.0

Índice de contenidos

Índice de contenidos	i
Instalación de herramientas y componentes previos.....	1
Instalación de Docker.....	2
Desplegar MongoDB con Docker	3
Montando Backend con NodeJs + Express	4
Conectando MongoDB al servidor.....	9
Modelos y controladores en NodeJs.....	11
Postman y Robo3T/Compass	15
Montando Frontend con VueJs.....	22
Conexión Backend y Frontend.....	23
Referencias.....	31

Instalación de herramientas y componentes previos

1. NodeJs y npm desde NodeSource [\[1\]](#)

Para hacerlo tendremos que disponer de `curl` instalado. Si no cuentas con esta herramienta, puedes instalarla con el comando:

```
sudo apt install curl
```

Luego, para la instalación y desde la terminal tendremos que ejecutar el comando

```
sudo apt install nodejs
```

Una vez terminada la instalación, los módulos NodeJs y npm deben estar instalados y listos para usar. Verificar mediante

```
node -version
```

```
npm --version
```

2. Vue/cli [\[2\]](#)

Para instalar un nuevo paquete

```
npm install -g @vue/cli
```

ó

```
yarn global add @vue/cli
```

Verificar mediante

```
vue --version
```

3. Postman

Para instalar Postman simplemente ejecutar desde el terminal

```
sudo snap install postman
```

Instalación de Docker

Docker es una herramienta que permite crear contenedores ligeros y portables para las aplicaciones software que puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga por debajo, facilitando así también los despliegues.

¿Qué hacemos para instalar Docker?

- Actualizar lista de paquetes existentes
`sudo apt update`
- Instalar paquetes de requisitos previos
`sudo apt install apt-transport-https ca-certificates curl software-properties-common`
- Añadir clave de GPG (siglas de GNU Privacy Guard)
`curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -`
- Agregar repositorio de Docker a las fuentes de APT
`sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"`
- Actualizar paquete de base de datos con los paquetes de Docker del repositorio recién creado
`sudo apt update`
- Instalación de Docker
`sudo apt install docker-ce`
- Con el paso anterior, Docker quedará instalado. Comprobar mediante
`sudo systemctl status docker`

Desplegar MongoDB con Docker

- Traer la imagen de Mongo al repositorio
`docker pull mongo`
- Verificar el repositorio de la imagen
`docker images`
- Arrancar la imagen para inicializar un contenedor, donde <name_container> es el nombre del contenedor que servirá de referencia para utilizarlo
`docker run --name <name_container> -d -p 27017:27017 mongo`
- Verificar el contenedor en el apartado NAMES, que debería mencionar el nombre del contenedor indicado en el paso anterior
`docker ps`

iBien!: Ya tenemos MongoDB corriendo en el puerto 27017. Cabe destacar que es una base de datos "simple", ya que no posee credenciales de acceso, estas pueden ser configuradas más tarde.

Montando Backend con NodeJs + Express

Aviso: Las imágenes mostrarán el puerto 8080 por defecto, mientras que en el instructivo se hará mención del puerto 8081 debido a la integración con VueJs. Favor seguir las instrucciones, imágenes solo de referencia. [\[3\]](#)



- Verificar que NodeJs y NPM estén instalados

```
~ ➤ node -v
v14.15.0
~ ➤ npm -v
6.14.8
```

- Nombrar la carpeta de instalación en un directorio a elección (en este caso se utiliza Fingeso, ubicada en el directorio Desktop)

```
cd Fingeso
mkdir backend
```

- Inicializar proyecto NodeJs

```
~/Desktop/Fingeso/Backend ➤ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (backend) █
```

- Recordar cambiar la descripción, nombre de autor y aceptar la licencia por defecto para generar el archivo `package.json`. El resto se puede omitir presionando `ENTER`

```
package name: (backend)
version: (1.0.0)
description: This is a backend test
entry point: (index.js)
test command:
git repository:
keywords:
author: Francisco
license: (ISC)
```

- En la carpeta creada dentro del directorio a elección (`~/Desktop/Fingesio/backend`), instalar Express
`npm install express`
- Utilizando el IDE preferido, abrir el directorio del proyecto y crear el archivo `index.js`. Luego, ingresar los siguientes datos

```
// Insertar luego de la siguiente línea

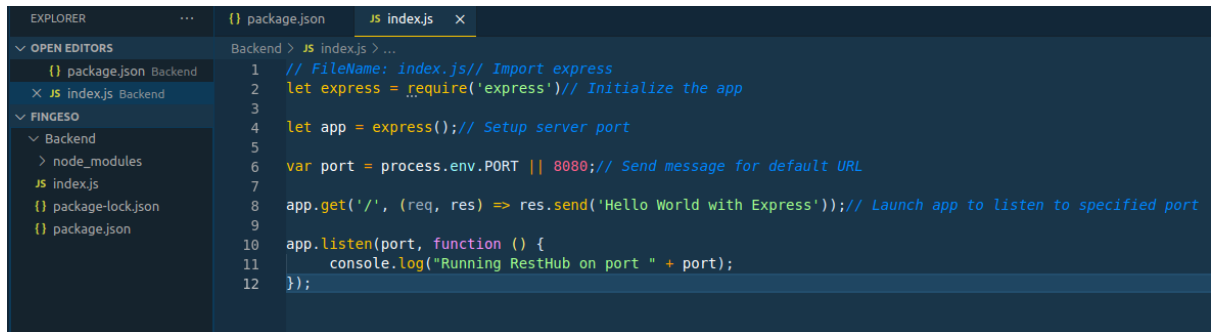
let express = require('express') // Initialize the app

let app = express(); // Setup server port

var port = process.env.PORT || 8081 // send message for default URL

app.get('/', (req, res) => res.send('Hello World with Express')); // Launch
app to listen to specified port

app.listen(port, function () {
  console.log("Running RestHub on port " + port);
});
```



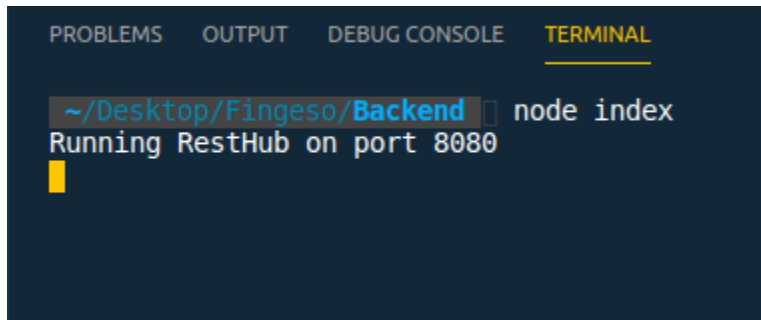
The screenshot shows the VS Code interface. On the left, the Explorer sidebar shows a project structure with folders 'Backend' and 'FINGESO'. The 'Backend' folder is expanded, showing files 'index.js', 'package-lock.json', and 'package.json'. The 'index.js' file is selected and its content is displayed in the editor. The code in 'index.js' is as follows:

```

1 // FileName: index.js // Import express
2 let express = require('express') // Initialize the app
3
4 let app = express() // Setup server port
5
6 var port = process.env.PORT || 8080 // Send message for default URL
7
8 app.get('/', (req, res) => res.send('Hello World with Express')); // Launch app to listen to specified port
9
10 app.listen(port, function () {
11   console.log("Running RestHub on port " + port);
12 });

```

- Guardar el archivo y ejecutar `node index` en el terminal



The screenshot shows the VS Code terminal with the 'TERMINAL' tab selected. The command prompt shows the current directory as `~/Desktop/Fingeso/Backend`. The command `node index` has been executed, and the output is `Running RestHub on port 8080`.

- Abrir el navegador web de preferencia e ingresar a `http://localhost:8081`



¡Excelente!: Ya inicializamos el servidor backend con NodeJs y Express. Ahora toca agregar un poco de MVC (modelo, vista, controlador) a la estructura de la aplicación, para esto indicaremos tres nuevos módulos de trabajo, api-routes, controller y model.

- Crear un archivo `api-routes.js` en el proyecto con el siguiente contenido

```

// Insertar luego de la siguiente línea
// Filename: api-routes.js
// Initialize express router
let router = require('express').Router();
// Set default API response

```



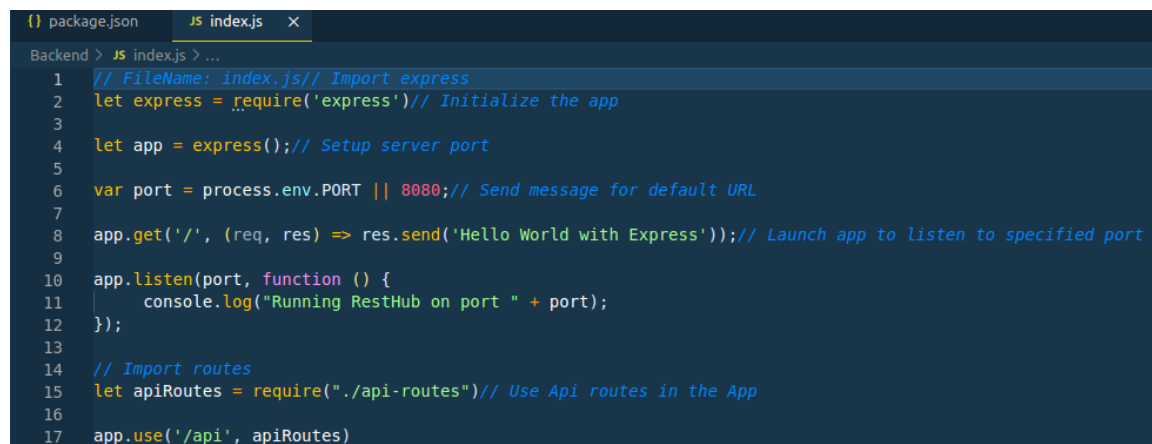
```
router.get('/', function (req, res) {
  res.json({
    status: 'API Its Working',
    message: 'Welcome to RESTHub crafted with love! '
  });
});

// Export API routes
module.exports = router;
```

- Luego, importar express router. Para disponibilizar esta ruta necesitamos modificar `index.js` añadiendo las siguientes líneas

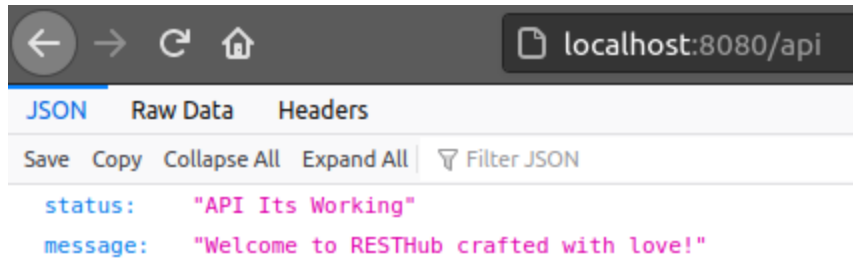
```
// Insertar luego de la siguiente línea
// Import routes
let apiRoutes = require("./api-routes")

//Use API routes in the App
app.use('/api', apiRoutes);
```



```
() package.json JS index.js x
Backend > JS index.js > ...
1 // FileName: index.js// Import express
2 let express = require('express')// Initialize the app
3
4 let app = express();// Setup server port
5
6 var port = process.env.PORT || 8080;// Send message for default URL
7
8 app.get('/', (req, res) => res.send('Hello World with Express'));// Launch app to listen to specified port
9
10 app.listen(port, function () {
11   console.log("Running RestHub on port " + port);
12 });
13
14 // Import routes
15 let apiRoutes = require("./api-routes");// Use Api routes in the App
16
17 app.use('/api', apiRoutes)
```

- Reiniciar el servidor terminando el proceso por la terminal y volviendo a iniciar con `node index`. Ingresar nuevamente a `http://localhost:8081` y luego a `http://localhost:8081/api`, debería arrojar el siguiente resultado



Consejo: Cada vez que se añade un archivo o editamos uno existente, para ver los resultados en el servidor hay que “reiniciarlo” cortando el servicio con `ctrl+c` y volviendo a ejecutar con `node index`. Para evitar este tipo de inconvenientes se recomienda instalar nodemon, un modulo de node que permite ver los cambios que se generan en los archivos del servidor al instante.

- Dentro del directorio de su proyecto
`npm install nodemon`

Conectando MongoDB al servidor

- Antes de configurar, instalar los paquetes correspondientes a mongoose y body-parser (respectivamente). Mongoose es un paquete de NodeJs para modelar MongoDB. Body-parser habilita la transformación de datos para la petición entrante.

```
npm install mongoose
```

```
npm install body-parser
```

- Modificar `index.js` con las siguientes líneas', donde `mongodbatabase` es el nombre del contenedor de Mongo en docker

```
// Insertar luego de la siguiente línea
// Import Body parser
let bodyParser = require('body-parser');

// Import Mongoose
let mongoose = require('mongoose');

// Configure bodyparser to handle post requests
app.use(bodyParser.urlencoded ({
  extended: true
}));

app.use(bodyParser.json());

// Connect to Mongoose and set connection variable
mongoose.connect('mongodb://localhost:27017/mongodatabase', { useNewUrlParser:
true});

var db = mongoose.connection;
```

```
// Import Body parser
let bodyParser = require('body-parser');

// Import Mongoose
let mongoose = require('mongoose');

// Configure bodyparser to handle post requests
app.use(bodyParser.urlencoded({
  extended: true
}));app.use(bodyParser.json());

// Connect to Mongoose and set connection variable
mongoose.connect('mongodb://localhost:27017/mongodatabase', { useNewUrlParser: true});

var db = mongoose.connection;
```

iPerfecto!: Nuestro servidor ya está conectado a la base de datos MongoDB, es hora de continuar con modelos y controladores, que permiten generar las entidades del servidor.

Modelos y controladores en NodeJs

Ahora vamos a configurar el modelo y controlador de una clase de prueba llamada `contact`. El controlador será el encargado de gestionar las peticiones y el modelo quien almacena/pide datos desde la base de datos. Para la implementación de la clase `contact`, se implementarán modelos de datos simples para almacenar información, estos son:

- Name
- Email
- Phone
- Gender

Además, se implementarán las siguientes peticiones:

- `GET /api/contacts` listar todos los contactos
- `POST /api/contacts` crear un nuevo contacto
- `GET /api/contacts/{id}` devolver un solo contacto
- `PUT /api/contacts/{id}` actualizar un contacto
- `DELETE /api/contacts/{id}` eliminar un contacto

Es momento de añadir dos archivos más, `contactController.js` y `contactModel.js`

```
// contactController.js
// Import contact model
Contact = require('./contactModel');

// Handle index actions
exports.index = function (req, res) {
  Contact.get(function (err, contacts) {
    if (err) {
      res.json({
        status: "error",
        message: err,
      });
    }
    res.json({
      status: "success",
      message: "Contacts retrieved successfully",
      data: contacts
    });
  });
};

// Handle create contact actions
exports.new = function (req, res) {
```

```
var contact = new Contact();
contact.name = req.body.name ? req.body.name : contact.name;
contact.gender = req.body.gender;
contact.email = req.body.email;
contact.phone = req.body.phone;
// save the contact and check for errors
contact.save(function (err) {
  if (err)
    res.json(err);
  res.json({
    message: 'New contact created!',
    data: contact
  });
});

// Handle view contact info
exports.view = function (req, res) {
  Contact.findById(req.params.contact_id, function (err, contact) {
    if (err)
      res.send(err);
    res.json({
      message: 'Contact details loading..',
      data: contact
    });
  });
};

// Handle update contact info
exports.update = function (req, res) {
  Contact.findById(req.params.contact_id, function (err, contact) {
    if (err)
      res.send(err);
    contact.name = req.body.name ? req.body.name : contact.name;
    contact.gender = req.body.gender;
    contact.email = req.body.email;
    contact.phone = req.body.phone;
    // save the contact and check for errors
    contact.save(function (err) {
      if (err)
        res.json(err);
      res.json({
        message: 'Contact Info updated',
        data: contact
      });
    });
  });
};

// Handle delete contact
exports.delete = function (req, res) {
  Contact.remove({
    _id: req.params.contact_id
  }, function (err, contact) {
```

```
        if (err)
            res.send(err);
        res.json({
            status: "success",
            message: 'Contact deleted'
        });
    });
};
```

Para habilitar el controlador y usar sus instancias para manejar el CRUD (Create, Retrieve, Update y Delete) se debe copiar y pegar el siguiente código correspondiente al modelo de `contact`

```
//contactModel.js
var mongoose = require('mongoose');

// Setup schema
var contactSchema = mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true
    },
    gender: String,
    phone: String,
    create_date: {
        type: Date,
        default: Date.now
    }
});

// Export Contact model
var Contact = module.exports = mongoose.model('contact',
contactSchema);module.exports.get = function (callback, limit) {
    Contact.find(callback).limit(limit);
}
```

En el modelo, importamos `mongoose` para crear el esquema de la base de datos para `contact` y finalmente se exporta el modulo para que sea accesible. Solo queda actualizar `api-routes.js` quedando de la siguiente forma

```
// api-routes.js

// Initialize express router
let router = require('express').Router();

// Set default API response
router.get('/', function (req, res) {
    res.json({
        status: 'API Its Working',
```

```
        message: 'Welcome to RESTHub crafted with love!',
      });
    });

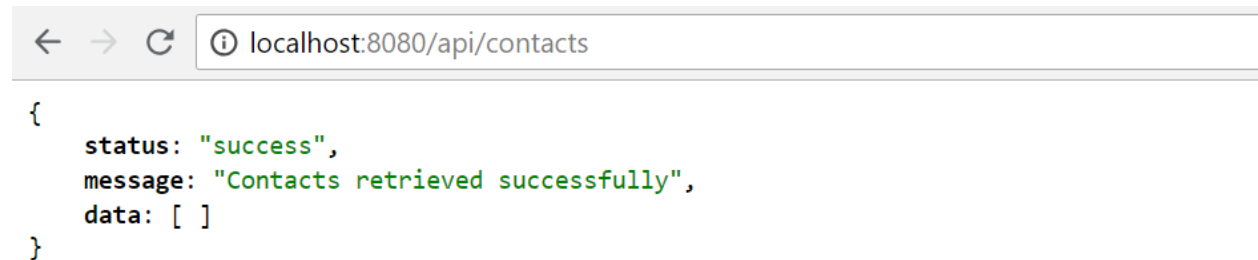
// Import contact controller
var contactController = require('./contactController');

// Contact routes
router.route('/contacts')
  .get(contactController.index)
  .post(contactController.new);

router.route('/contacts/:contact_id')
  .get(contactController.view)
  .patch(contactController.update)
  .put(contactController.update)
  .delete(contactController.delete);

// Export API routes
module.exports = router;
```

Revisemos con un buscador a elección, ingresa a `http://localhost:8080/api/contact`



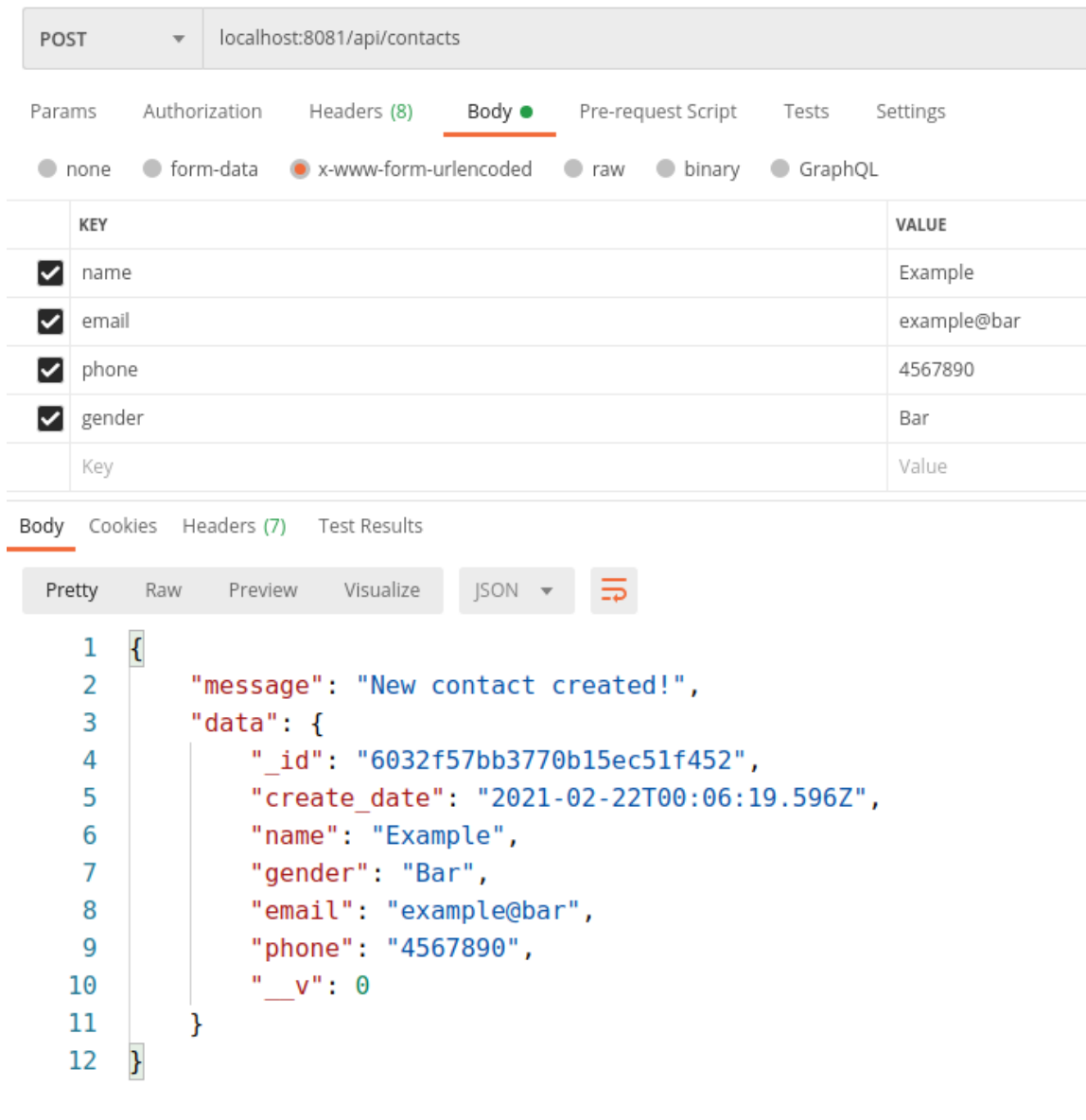
```
{
  status: "success",
  message: "Contacts retrieved successfully",
  data: [ ]
}
```

Como muestra la imagen, data está vacío puesto que no hay información en la base de datos. Es hora de rellenar con datos mediante **Postman**.

Postman y Robo3T/Compass

Para añadir contactos a la base de datos, podemos utilizar una buena herramienta para testeos y debugging de rutas API. Postman facilita el envío de peticiones al servidor backend por medio de rutas y métodos intuitivos, además de ofrecer particularidades como crear colecciones para un mismo tipo de clase o compartir datos para un equipo de trabajo.

- Vamos a probar creando un `contact`



The screenshot shows the Postman interface for a POST request to `localhost:8081/api/contacts`. The **Body** tab is selected, and the data is formatted as `x-www-form-urlencoded`. The request body is a JSON object with the following fields:

KEY	VALUE
<input checked="" type="checkbox"/> name	Example
<input checked="" type="checkbox"/> email	example@bar
<input checked="" type="checkbox"/> phone	4567890
<input checked="" type="checkbox"/> gender	Bar
Key	Value

The response is displayed in the **Body** tab, showing a JSON object with a success message and contact data:

```
1 {
2   "message": "New contact created!",
3   "data": {
4     "_id": "6032f57bb3770b15ec51f452",
5     "create_date": "2021-02-22T00:06:19.596Z",
6     "name": "Example",
7     "gender": "Bar",
8     "email": "example@bar",
9     "phone": "4567890",
10    "__v": 0
11  }
12 }
```

- Obtener todos los contactos

The screenshot shows a REST client interface with a GET request to `localhost:8081/api/contacts`. The 'Query Params' section is empty. The 'Body' tab is selected, showing a JSON response in 'Pretty' format. The response indicates a successful status and contains two contact objects in the 'data' array.

GET `localhost:8081/api/contacts`

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "status": "success",
3   "message": "Contacts retrieved successfully",
4   "data": [
5     {
6       "_id": "6032f4f0b3770b15ec51f451",
7       "create_date": "2021-02-22T00:04:00.038Z",
8       "name": "FooBar",
9       "gender": "Foo",
10      "email": "foo@bar",
11      "phone": "12345678",
12      "__v": 0
13    },
14    {
15      "_id": "6032f57bb3770b15ec51f452",
16      "create_date": "2021-02-22T00:06:19.596Z",
17      "name": "Example",
18      "gender": "Bar",
```

- Obtener un contacto en particular mediante el id

The screenshot shows a REST client interface with the following components:

- Request Bar:** Method: GET, URL: localhost:8081/api/contacts/6032f4f0b3770b15ec51f451
- Params Tab:** Query Params table with 2 columns: KEY, VALUE.
- Body Tab:** JSON response displayed in Pretty format.

KEY	VALUE
Key	Value

```
1  {
2    "message": "Contact details loading..",
3    "data": {
4      "_id": "6032f4f0b3770b15ec51f451",
5      "create_date": "2021-02-22T00:04:00.038Z",
6      "name": "Now is Bar!",
7      "gender": "Foo",
8      "email": "foo@bar",
9      "phone": "12345678",
10     "__v": 0
11   }
12 }
```

- Actualizar un `contact` mediante el id

The screenshot shows a REST client interface with a PUT request to `localhost:8081/api/contacts/6032f4f0b3770b15ec51f451`. The request body is a JSON object with the following fields:

```
1 {
2   "name": "Now is Bar!",
3   "gender": "Foo",
4   "email": "foo@bar",
5   "phone": "12345678"
6 }
```

Below the request, the response body is shown in JSON format:

```
1 {
2   "message": "Contact Info updated",
3   "data": {
4     "_id": "6032f4f0b3770b15ec51f451",
5     "create_date": "2021-02-22T00:04:00.038Z",
6     "name": "Now is Bar!",
7     "gender": "Foo",
8     "email": "foo@bar",
9     "phone": "12345678",
10    "__v": 0
11  }
12 }
```

Ya que tenemos nuestros datos creados, es momento de visualizarlos. Para esta instancia pueden utilizar tanto Robo3T como Compass, interfaces visuales de MongoDB que facilitan el manejo de datos y visualización de colecciones dentro de ésta.

- Instalación Robo3T

```
sudo snap install robo3t-snap
```

- Para la instalación de Compass, ingresar a

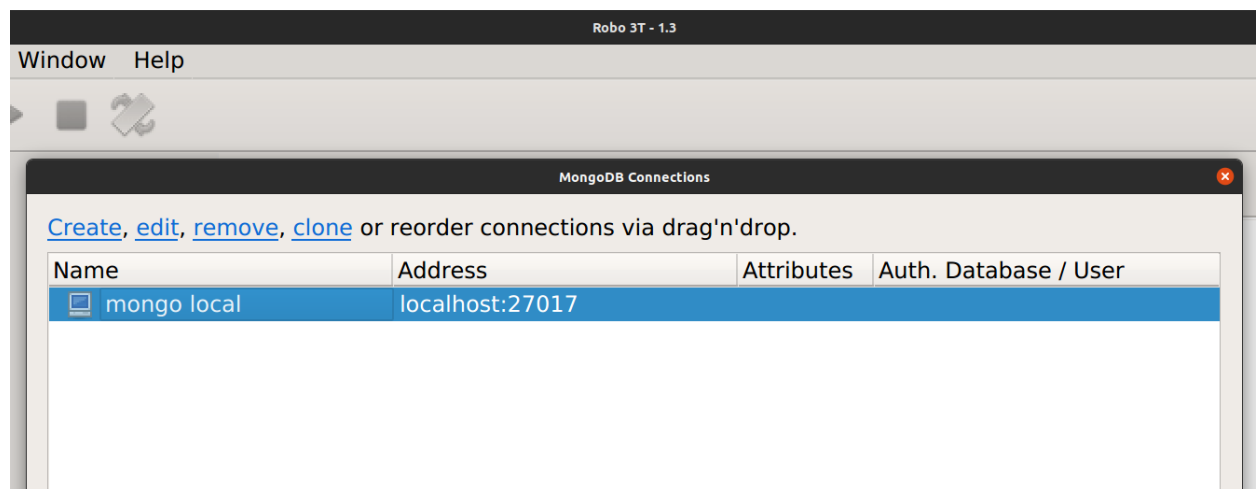
<https://www.mongodb.com/try/download/compass>

- Luego de seleccionar distribución y descargar el paquete .dev, instalar mediante el comando, donde EnterfileName.deb es el nombre del paquete descargado

```
sudo dpkg -i EnterfileName.deb
```

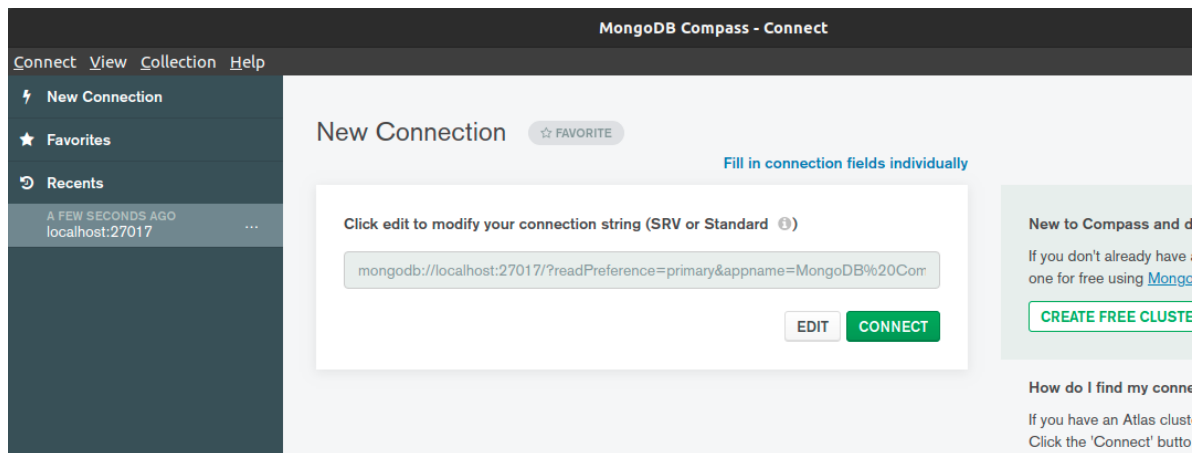
El funcionamiento de ambos gestores es similar, solo requieren de una conexión sin requisitos previos ya que es local.

- Para Robo3T



Ingresa directamente desde el localhost y Connect.

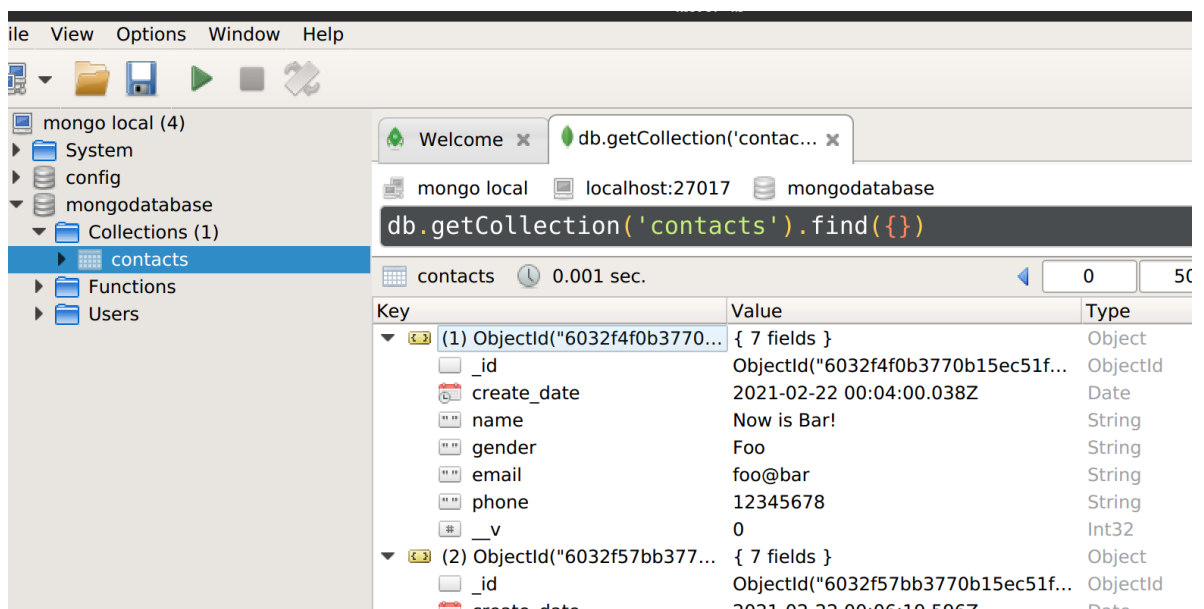
- Para Compass



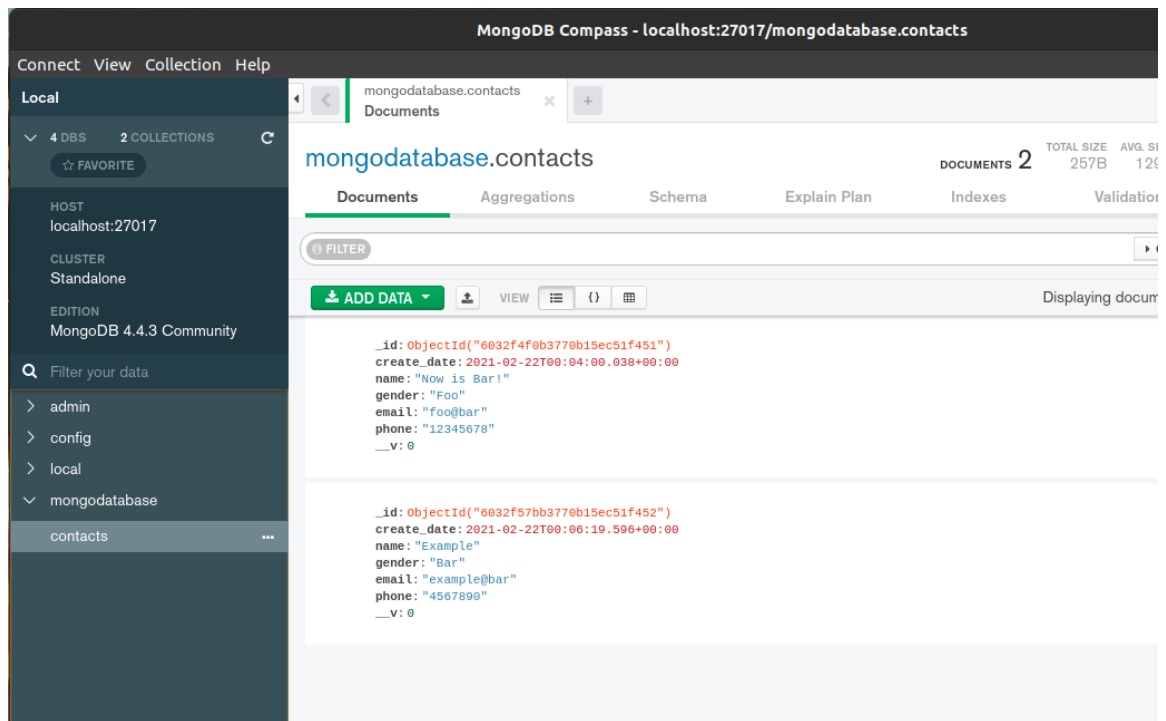
Dar click en Connect, no existe necesidad de rellenar datos.

Para visualizar datos, ingresar a los respectivos directorios que ofrecen las aplicaciones.

- Robo3T



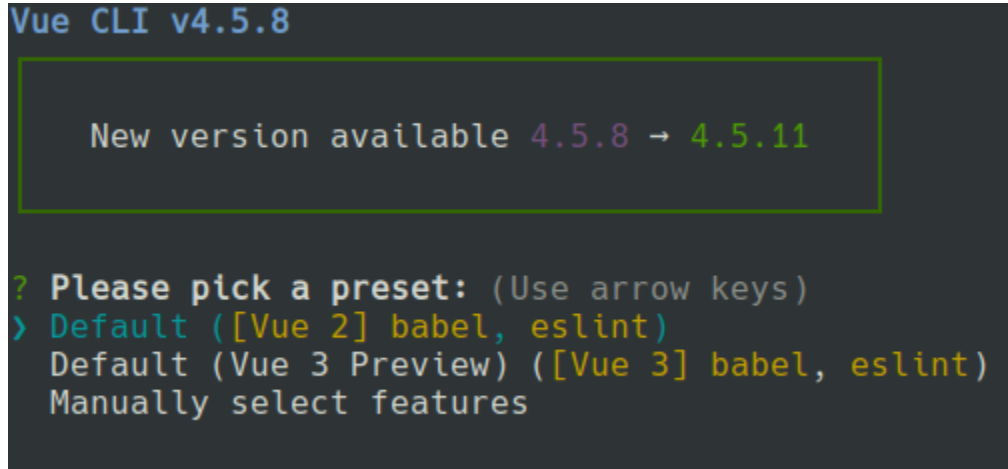
- Compass



Montando Frontend con VueJs

- Ubicarse en el directorio padre del proyecto (aquel que contiene al backend, en este caso Fingeso) desde la terminal, pegar lo siguiente considerando <nombre_proyecto_front> como la carpeta contendora del proyecto que generará VueJs (se utilizará el nombre frontend)

```
vue create <nombre_proyecto_front>
```



```
Vue CLI v4.5.8

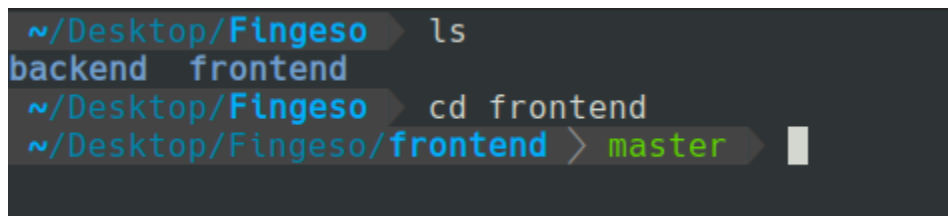
New version available 4.5.8 -> 4.5.11

? Please pick a preset: (Use arrow keys)
> Default ([Vue 2] babel, eslint)
  Default (Vue 3 Preview) ([Vue 3] babel, eslint)
  Manually select features
```

- Seleccionar
Default ([Vue 2] babel, eslint)

- Ingresar al directorio del proyecto

```
cd frontend
```



```
~/Desktop/Fingeso > ls
backend  frontend
~/Desktop/Fingeso > cd frontend
~/Desktop/Fingeso/frontend > master
```

- Ejecutar el servidor mediante el comando `npm run serve`, este quedará disponible en la siguiente URL

```
http://localhost:8080
```


Conexión Backend y Frontend

Antes de crear la conexión, es necesario agregar ciertos módulos que ayudarán en el proceso de envío de información desde el Frontend hasta el Backend.

- Ingresar al directorio que contenga el frontend

```
cd frontend
```

- Dentro del directorio ingresar el comando

```
vue add router
```

Esto generará un archivo llamado `index.js` dentro de la ruta `router`

- Dentro del mismo directorio del proyecto de frontend ingresar

```
npm install --save axios vue-axios
```

- Ingresar al archivo `main.js` e importar axios y router si estos no se han añadido

```
// Insertar luego de la siguiente línea
```

```
import router from './router'
```

```
import axios from 'axios'
```

- Luego de la línea `Vue.config.productionTip = false`, añadir

```
// Insertar luego de la siguiente línea
```

```
// Instancia de axios se configura con URL base del backend
```

```
const axiosInstance = axios.create({  
  baseURL: "http://localhost:8081/api"  
})
```

```
// Para acceder a axios desde this.$http
```

```
Vue.prototype.$http = axiosInstance
```

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

Vue.config.productionTip = false

// Instancia de axios se configura con URL base del backend
const axiosInstance = axios.create({
  baseURL: "http://localhost:8081"
})

// Para acceder a axios desde this.$http
Vue.prototype.$http = axiosInstance;

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

iBien!: Ya contamos con los módulos que permiten al frontend enviar peticiones mediante una URL. Nuestra baseURL nos permite ahorrar líneas de código e importaciones para llamar a axios y enviar peticiones al backend. ¿Qué sigue?

- Ingresar a `views` y crear un nuevo archivo llamado `Contact.vue`
- Anadir las siguientes líneas de código, nos permite generar un formulario simple

```
// Insertar luego de la siguiente línea
<template>
<div class="container">
  <h1>Agregar un contacto</h1>
  <form>
    <div>
      <label for="name">Name</label>
      <input type="text" id="name" v-model="newContact.name">
    </div>
    <div>
      <label for="name">Email</label>
      <input type="text" id="email" v-model="newContact.email">
    </div>
    <div>
      <label for="name">Gender</label>
```

```
        <input type="text" id="gender" v-model="newContact.gender">
      </div>
      <div>
        <label for="name">Phone</label>
        <input type="text" id="phone" v-model="newContact.phone">
      </div>
      <div>
        <button type="button" @click="send">Crear</button>
      </div>
      <div class="info">
        <h2>Objeto</h2>
        <code>{{newContact}}</code>
        <p class="message">
          {{message}}
        </p>
      </div>
    </form>
  </div>
</template>
```

- Ingresar a `router/index.js` y añadir la ruta de la nueva vista

```
// Insertar luego de la siguiente línea
{
  path: '/contact',
  name: 'Contact',
  component:() => import(/* webpackChunkName: "about" */ '../views/Contact.vue')
}
```

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import Home from '../views/Home.vue'

Vue.use(VueRouter)

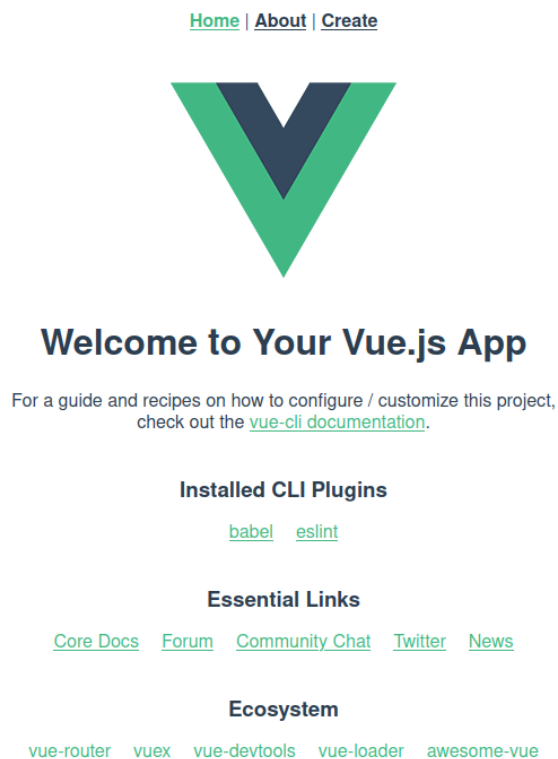
const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home
  },
  {
    path: '/about',
    name: 'About',
    // route level code-splitting
    // this generates a separate chunk (about.[hash].js) for this route
    // which is lazy-loaded when the route is visited.
    component: () => import(/* webpackChunkName: "about" */ '../views/About.vue')
  },
  {
    path: '/contact',
    name: 'Contact',
    component: () => import(/* webpackChunkName: "about" */ '../views/Contact.vue')
  }
]
```

- Luego, añadir a la barra de direcciones generada por Vue para acceder a esta vista. Esta barra de direcciones se encuentra en `App.vue`

```
// Insertar luego de la siguiente línea
<template>
  <div id="app">
    <div id="nav">
      <router-link to="/">Home</router-link> |
      <router-link to="/about">About</router-link> |
      <router-link to="/contact">Create</router-link>
    </div>
    <router-view/>
  </div>
</template>
```

```
<template>
  <div id="app">
    <div id="nav">
      <router-link to="/">Home</router-link> |
      <router-link to="/about">About</router-link> |
      <router-link to="/contact">Create</router-link>
    </div>
    <router-view/>
  </div>
</template>
```

- Iniciar el servidor
`npm run serve`
- Ingresar a la URL
`http://localhost:8080`



- Clickear en `Create`, para ingresar a la ventana que hemos creado

[Home](#) | [About](#) | [Create](#)

Agregar un contacto

Name

Email

Gender

Phone

Objeto

```
{}
```

¡Excelente!: Ya tenemos una ventana simple para crear un contacto, pero... no sirve 😞. Esto debido a que el botón Crear no está "gatillando" ningún evento. ¡Es momento de llamar al backend!

- Necesitamos añadir un modulo que permita recibir cualquier tipo de *request* en el servidor. Para esto necesitamos ingresar al directorio del backend
`cd backend`

- Ingresar el comando para la instalación de CORS (*Cross-origin resource sharing*)
`npm install cors`

- Añadir las siguientes líneas al archivo `index.js` dentro de los espacios señalados

```
// Insertar luego de la siguiente línea
var cors = require('cors') // Dentro de importaciones
app.use(cors())           // Dentro de ejecuciones
```

- Volver al directorio frontend e ingresar al archivo `Contact.vue`, crear una sección `<script></script>` bajo el `<template>` e ingresar las siguientes líneas

```
// Insertar luego de la siguiente línea
export default {
  data(){
    return{
      message:'',
      newContact:{
      }
    }
  },
  methods:{
    send:async function(){
      this.message = '';
      if (this.newContact.name == ''){
        this.message = 'Debes ingresar un nombre'
        return false
      }
      try {
        var result = await this.$http.post('/api/contacts',
this.newContact);
        let contact = result.data;
        this.message = `Se creó un nuevo contacto con id:
${contact.data._id}`;
        this.newContact = {};
      } catch (error) {
        console.log('error', error)
        this.message = 'Ocurrió un error'
      }
    }
  }
}
```

```
<script>
export default {
  data(){
    return{
      message:'',
      newContact:{
      }
    }
  },
  methods:{
    send:async function(){
      this.message = '';
      if (this.newContact.name == ''){
        this.message = 'Debes ingresar un nombre'
        return false
      }
      try {
        var result = await this.$http.post('/api/contacts', this.newContact);
        let contact = result.data;
        this.message = `Se creó un nuevo contacto con id: ${contact.data._id}`;
        this.newContact = {};
      } catch (error) {
        console.log('error', error)
        this.message = 'Ocurrió un error'
      }
    }
  }
}
</script>
```

iPerfecto!: Todo habilitado, es momento de crear un contacto y verificar los datos ingresados por medio de Robo3T/Compass.

Referencias

- [1] D.A. (s. f.). *NodeJS y npm, instalación en Ubuntu 20.04 / 18.04*. Ubunlog. Recuperado 16 de febrero de 2021, de <https://ubunlog.com/nodejs-npm-instalacion-ubuntu-20-04-18-04/>

- [2] VueJs. (2020, 23 septiembre). *Installation / Vue CLI*. cli.vuejs.org. <https://cli.vuejs.org/guide/installation.html>

- [3] Inyang-Etoh, D. (2018, 1 julio). *How To Build Simple RESTful API With NodeJs, ExpressJs And MongoDB*. Medium. <https://medium.com/@dinyangetoh/how-to-build-simple-restful-api-with-nodejs-expressjs-and-mongodb-99348012925d>