Facultad de Ingeniería
Arturo Narro Siller

UAT

VERDAD, BELLEZA, PROBIDAD

ADMINISTRACION DE BASES DE DATOS CORPORATIVAS

Profesor: Olga Leticia Mares Lopez

Examen Teórico

MySQL API-Rest CRUD

ALUMNO:

Zubiri Valdez Hedson Leonardo

MATRÍCULA:

2183330170

*Semestre **7°*** *Grupo: "**G**"*

26 / 10 / 2021

Desarrollo WEB de una API-Rest para realizar operaciones CRUD con MySQL

Introducción

El desarrollo de este proyecto se compone de tres tecnologías principales, VueJS para el desarrollo de la interfaz (Frontend), MySQL como base de datos de nuestra aplicación y por último un servidor en lenguaje Golang para servir de intermediario entre las dos tecnologías antes mencionadas.

Instalaciones necesarias

Lenguaje Golang.

Después de instalar Go, desde una terminal debemos descargar los siguientes paquetes para trabajar con MySQL:

```
go get -u github.com/go-sql-driver/mysql
```

```
go get -u "github.com/joho/godotenv"
```

Node js

Una vez instalado correctamente NodeJS abrir una terminal de comandos para instalar las siguientes tecnologías.

- **Axios.** Nos ayudará a hacer peticiones a nuestro servidor golang. Se instala mediante el comando:

```
npm i axios
```

- **VueJS**. Descargar los paquetes necesarios para trabajar con este Framework:

```
npm install -g vue
```

- **Vue UI.** Posteriormente instalar una interfaz para la gestión de proyectos con Vue:

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

Visual Studio Code

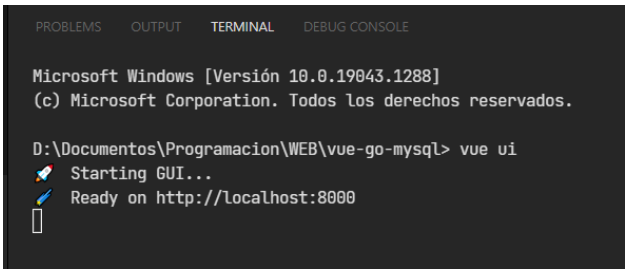
VsCode Será nuestro editor de código por lo que se recomienda buscar e instalar las siguientes extensiones:

- Go
- HTML CSS Support- ecmel

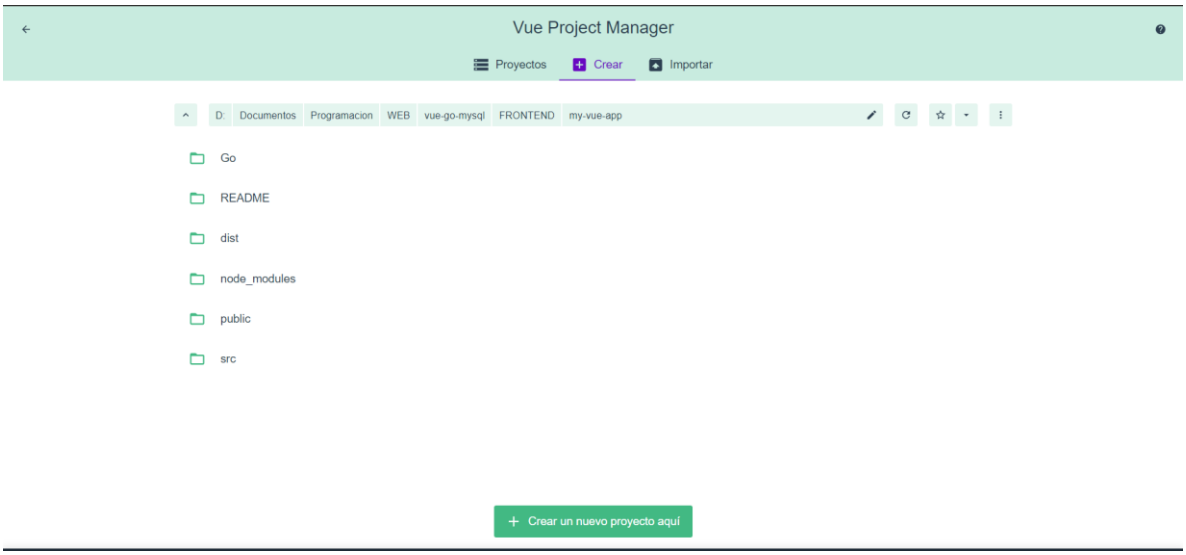
- JavaScript(ES6) code snippets - Charalampos
- Vetur
- Material Icon Theme

SET UP de la estructura del proyecto

El primer paso es abrir una terminal, esta puede ser la terminal integrada al vscode. Ejecutar el comando vue ui

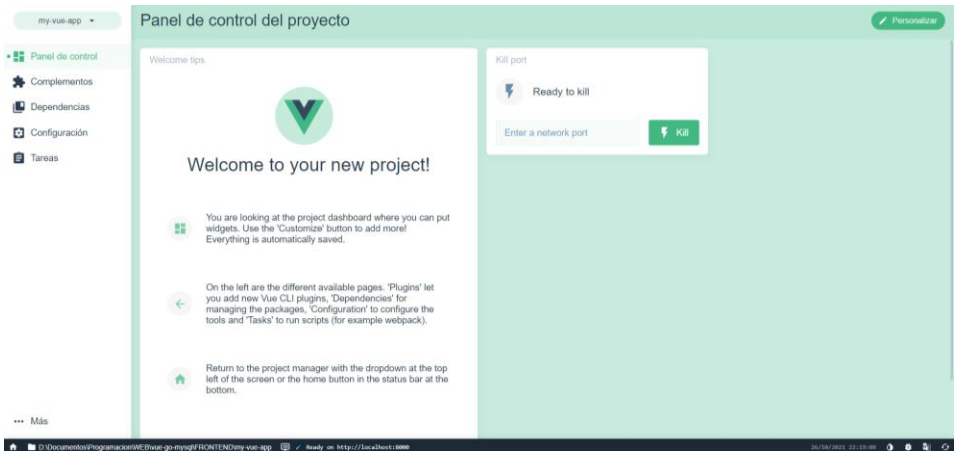


Se abre en nuestro navegador la interfaz para crear los proyectos:

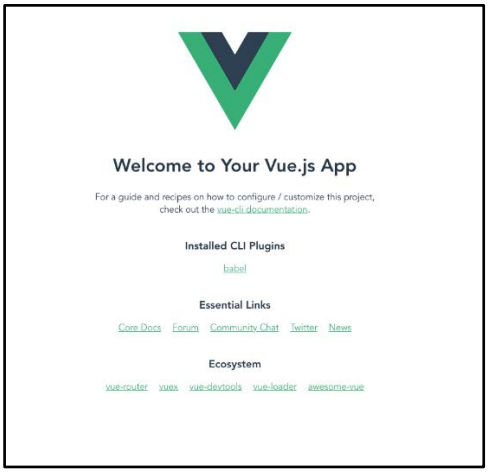


Seleccionamos la ruta y creamos el proyecto, seleccionamos que el gestor de paquetes sea NPM y que sea basado en VUE 3.0.

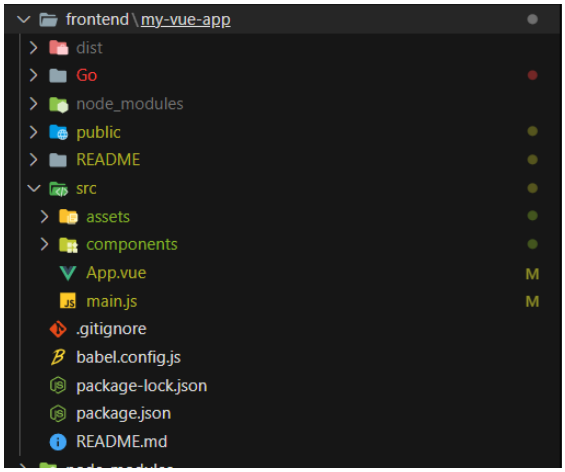
Una vez que se accede al proyecto vamos al apartado de Tareas, y Ejectuar Tarea para empezar con la ejecución de la interfaz en VUE



Una vez terminado el proceso tendremos la opción de Abrir aplicación habilitada. Esto redirecciona hacia la interfaz de nuestro proyecto en Vue.



Al acceder desde VsCode a nuestro proyecto creado se encuentra la siguiente estructura, donde crearemos una carpeta llamada GO y dentro crearemos un archivo llamado `main.go`



Siempre que terminemos funcionalidades en nuestro FrontEnd debemos generar un distribuible, que podría entenderse como una versión funcional. Ejecutamos el siguiente código desde la ruta raíz del proyecto en una terminal: `npm run build`

Lo anterior crea una carpeta llamada **`dist`** , ruta donde se encuentra lo generado.

Desde nuestro **`main.go`** de manera general tenemos lo siguiente:

Paquetes necesarios:

```go
package main

import (
    "database/sql"
    "encoding/json"
    "fmt"
    "log"
    "net/http"
    "os"

    _ "github.com/go-sql-driver/mysql"
    "github.com/joho/godotenv"
)
```

Una función main que recibe el acceso a direcciones y apunta hacia métodos. Ejemplo:

Localhost:3000/delete apunta al método deleteData( ) dentro del main.go

Además se indica que desde la direccion raiz “/”, cargue la interfaz en la carpeta dist antes generada. Se inica el servidor en el puerto 3000.

```go
func main() {
    // Use the thumbnailHandler function
    http.HandleFunc("/data", setData)
    http.HandleFunc("/get", getData)
    http.HandleFunc("/delete", deleteData)
    http.HandleFunc("/editar", editarUsuario)

    // Serve static files from the frontend/dist directory.
    fs := http.FileServer(http.Dir("../dist"))
    http.Handle("/", fs)

    // Start the server.
    fmt.Println("Server listening on port 3000")
    log.Panic(
        http.ListenAndServe(":3000", nil),
    )

}
```

Se cargan las credenciales para acceder a la base de datos de un archivo .env

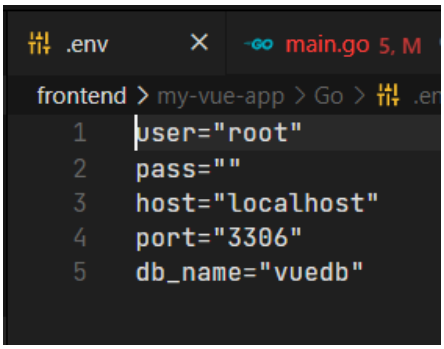
```go
var _ = godotenv.Load(".env") // Cargar del archivo llamado ".env"
var (
    ConnectionString = fmt.Sprintf("%s:%s@tcp(%s:%s)/%s",
        os.Getenv("user"),
        os.Getenv("pass"),
        os.Getenv("host"),
        os.Getenv("port"),
        os.Getenv("db_name"))
)

const AllowedCORSDomain = "http://localhost"

// Credenciales de conexion y dominios permitidos para hacer peticiones

func getDB() (*sql.DB, error) {
    return sql.Open("mysql", ConnectionString)
}
```

El archivo .env se crea en la misa carpeta Go y debe tener lo siguiente.



Iniciar nuestro servidor

Si es la primera vez debemos inicializar el modulo go a través de la terminal:

```
go mod init goservername
go mod tidy
```

Después siempre que se hagan cambios debemos hacer una construcción:

```
go build
```

Dentro de la carpeta de Go se crea un ejecutable, a través de la terminal lo ejecutamos:

```
goservername.exe
```

Y el servidor go estará ejecutándose.

Comunicación entre VUE JS y GOLANG

Desde el main.go también tenemos lo siguiente:

```go
type person struct {
    ID       int    `json:"id"`
    Nombre   string `json:"nombre"`
    Apellido string `json:"apellido"`
    Email    string `json:"email"`
}

//Recibe datos en formato JSON y los transforma a variables go
func setData(w http.ResponseWriter, request *http.Request) {
    decoder := json.NewDecoder(request.Body)

    var data person
    decoder.Decode(&data)

    w.Header().Set("Content-Type", "application/json; charset=UTF-8")
    w.Header().Set("Access-Control-Allow-Origin", "*")
    w.WriteHeader(http.StatusOK)

    insertPerson(data)

}
```

Es importante generar una estructura, para definir como tratar la información JSON que reciba o retorne, en este caso la estructura person.

El método **setData** recibe una petición, lo que llega es un JSON con datos, y a través de la estructura es como sabe con que variables nativas en go relacionarlas. Despues se le retorna una respuesta http.StatusOK para indicar que llegó la info.

Posteriormente ya un método que conecta a la base de datos. InsertPerson(data).

Desde el lado de VUE

Un componente en vue tiene la siguiente estructura, entonces en el proyecto hay varios componentes con formularios para ingresar informacion o tablas para mostrar:

```
<template>
  <div>
      <!— Estructura HTML que retorna —>
  </div>
</template>

<script>
//Codigo JS
export default {

}
</script>

<!— Estilos css del componente —>
<style>

</style>
```

Por ejemplo el siguiente botón de un formulario hace referencia al método request( ) que se encuentra en el apartado de Script del componente

```
<button class="btn btn-primary" v-on:click="request()">
  SEND!!
</button>
```

Se utiliza Axios para realizar la petición al servidor, se obtienen los datos del formulario y se agregan a una lista tipo JSON para enviarla. Se le indica que queremos acceder a /data del servidor. Después se espera un resultado.

```
<script>
import axios from "axios";
export default {
  name: "Formulario",
  methods: {
    request() {
      //var nombreDato = { nombre: nombreInput };
      var nombreDato = document.getElementById("nombreInput").value;
      var apellidoDato = document.getElementById("apeInput").value;
      var email = document.getElementById("emailInput").value;

      var data2 = { nombre: nombreDato, apellido: apellidoDato, email: email };

      console.log(nombreDato);

      console.log(apellidoDato);

      axios({
        method: "POST",
        url: "http://127.0.0.1:3000/data",
        data: data2,
        headers: { "content-type": "text/plain" },
      }).then((result) => {
        console.log(result.data);
        if (result.statusText == "OK") alert("Usuario agregado");
      });
      /*
        .catch((error) => {
          console.error(error);
        });
        */
    },
  },
};
</script>
```

CREATE

```
//Mandas un tipo de estructura en particular
func insertPerson(persona person) {

    db, err := getDB()
    if err != nil {
        fmt.Println(err)
    }
    _, err = db.Exec("INSERT INTO Personas (Persona, Apellido, Email) VALUES (?, ?, ?)", persona.Nombre, persona.Apellido, persona.Email)
    if err != nil {
        fmt.Println(err)
    }
    defer db.Close()
}
```

Esta es la manera con la que se pueden ejecutar Querys a MySQL. Se crea un nuevo usuario en la base de datos.

READ.

Obtiene todos los datos de la tabla personas y retorna un archivo JSON con los datos al servidor.

```go
func getInfoTableNamed(nombre string, w http.ResponseWriter) {

    db, err := getDB()
    if err ≠ nil {
        fmt.Println(err)
    }
    results, err := db.Query("select * from " + nombre)
    if err ≠ nil {
        fmt.Println("Error when fetching product table rows:", err)
    }
    defer results.Close()

    fmt.Println(reflect.TypeOf(results))

    auxPers := person{}
    arrPers := []person{}

    for results.Next() {
        var (
            id     int
            name   string
            ape    string
            correo string
        )
        err = results.Scan(&id, &name, &ape, &correo)
        if err ≠ nil {
            panic(err.Error())
        }
        auxPers.ID = id
        auxPers.Nombre = name
        auxPers.Apellido = ape
        auxPers.Email = correo

        arrPers = append(arrPers, auxPers)
    }
    defer db.Close()

    //Mandar en JSON los resultados al navegador
    if err := json.NewEncoder(w).Encode(arrPers); err ≠ nil {
        panic(err)
    }
}
```

UPDATE

Actualiza los campos de un registro en base al ID

```
func updateByID(persona person){
    db, err := getDB()
if err != nil {
    fmt.Println(err)
}
_, err = db.Exec("UPDATE Personas SET Persona=?, Apellido=?, Email=? WHERE CodPersona=?"
, persona.Nombre, persona.Apellido, persona.Email,persona.ID)
if err != nil {
    fmt.Println(err)
}
defer db.Close()
}
}
```

DELETE

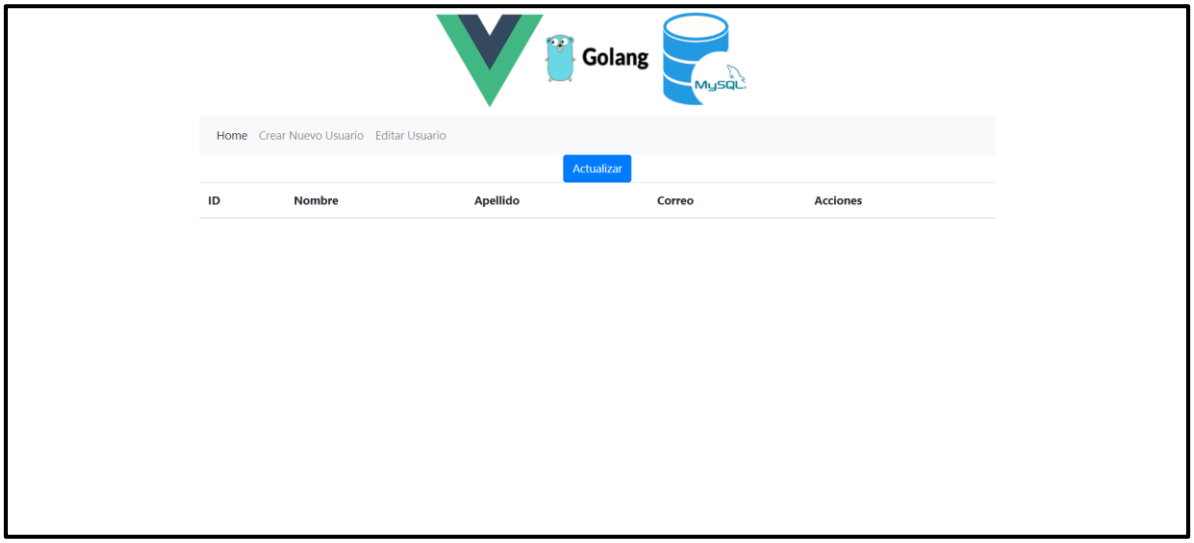
Elimina una persona de la tabla personas en base a su ID.

```
func deleteByID(id string) {
    db, err := getDB()
    if err != nil {
        fmt.Println(err)
    }
    _, err = db.Query("DELETE FROM personas WHERE codPersona=? ", id)
    if err != nil {
        fmt.Println("Error:", err)
    }
}
```

Resultados

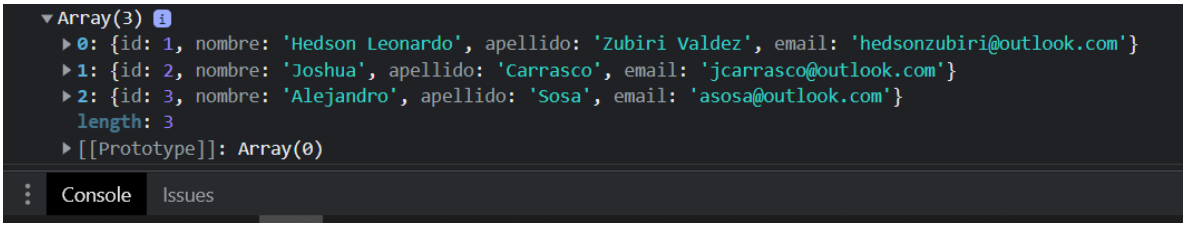
```
D:\Documentos\Programacion\WEB\vue-go-mysql\frontend\my-vue-app\Go>go build

D:\Documentos\Programacion\WEB\vue-go-mysql\frontend\my-vue-app\Go>mysqlgit.exe
Server listening on port 3000
```



Al presionar el boton de eliminar se elimina el registro

Se puede observar cómo llega la información al navegador



Para consultar el proyecto completo:

https://github.com/Leo-Zubiri/vue-go-mysql