Andrés Atehortúa Gutiérrez

Mariana Sinisterra

Julián Andrés restrepo

Yancarlos Cartagena Zuleta

Sebastián Ángel Marín

**Proyecto de taller 2**

Manual de programador para proyecto final de taller 2, evaluador Manolo Pájaro Borras

**Manual de programador**

Aplicativo web de VetLife

# TABLA DE CONTENIDO

[TABLA DE CONTENIDO 1](#_Toc152615256)

[INTRODUCCION 2](#_Toc152615257)

[CONCEPTOS BASICOS 3](#_Toc152615258)

[BACKEND 3](#_Toc152615259)

[FRONTEND 3](#_Toc152615260)

[BASE DE DATOS 3](#_Toc152615261)

[OTROS 3](#_Toc152615262)

[ENTORNO DE DESARROLLO Y HERRAMIENTAS 4](#_Toc152615263)

[VERSIONA MIENTO Y GESTIÓN DE TRABAJO 5](#_Toc152615264)

[COMANDOS BÁSICOS NPM 6](#_Toc152615265)

[RUTAS 8](#_Toc152615266)

# INTRODUCCION

Bienvenido al manual de programador del aplicativo web de la veterinaria vetlife. En este documento encontrara una guía detallada de la estructura funcionalidad y practicas usadas en la creación del aplicativo web asociadas al desarrollo de este, demostrando lo aprendido en diseño de aplicaciones webs.

El aplicativo web bridara facilidad en la gestión de citas de los usuarios vinculados a la empresa dándole información de sus mascotas en una interfaz intuitiva y cómoda. Este manual está diseñado para proporcionar a los desarrolladores una comprensión del código fuente y de su estructura para futuras mejoras y mantenimientos de esta plataforma.

# CONCEPTOS BASICOS

Para un correcto manejo de los conceptos de que utilizan se hablara de que se utiliza para frontend y backend, así como la base de datos que se implementa.

## BACKEND

Es la parte no visible para los usuarios del aplicativo web, este maneja la lógica, la gestión de datos y la comunicación con otras partes del sistema. Se ocupa del procesamiento de datos, la comunicación con el frontend, asegurando que las acciones del usuario se ejecuten correctamente en el aplicativo están conformados por node js, express, npm.

## FRONTEND

Es la parte del aplicativo web que interactúa directamente con el usuario. Se encarga de la presentación y la interfaz de usuario es todo lo que el usuario ve y con lo que interactúa en el sitio web se conformaría por HTML, CSS y JavaScript.

## BASE DE DATOS

Es un sistema organizado para almacenar y gestionar datos de manera estructurada en el caso del aplicativo web se utiliza una NoSQL que es mongodb.

## OTROS

En la creación de la aplicación se usó a manera de agilizar tiempo frameworks como los son bootstrap para el frontend.

# ENTORNO DE DESARROLLO Y HERRAMIENTAS

Para el desarrollo del aplicativo web se necesitaron varios programas y herramientas que son:

**Visual studio code:** Es un editor de texto altamente personalizable creado por Microsoft la integración de extensiones para agilizar métodos de trabajo y su alta variedad de lenguajes de programación siendo la principal herramienta usada para el desarrollo del aplicativo.

**Node js:** Es un entorno de ejecución multiplataforma, de código abierto, para la capa del servidor basado en lenguajes de programación JavaScript asíncrono.

**Express**:

**Npm:** Es el sistema de gestión de paquetes de node js es un entorno de ejecución para JavaScript, Nos facilita la instalación y la implementación de frameworks que se utilizan.

**HTML:** Es un lenguaje de marcado utilizado para estructurar y desplegar una página web y sus contenidos siendo como el esqueleto de un aplicativo web.

**CSS:** Es el lenguaje con el que damos estilos a todo el contenido que se etiqueta en un HTML se encarga de todo lo visual para dar un aspecto amigable al usuario, vendría a ser la piel del aplicativo.

**JavaScript:** Es un lenguaje de programación que los desarrolladores utilizan para hacer páginas web brindando la capacidad de interactuar con el usuario y los recursos asociados puede ser con la base de datos, servidores e información derivada, esto sería como los músculos del aplicativo.

**MongoDB:** Es el sistema de base de datos NoSQL que se utiliza en el aplicativo al ser no relacional brinda gran versatilidad en el manejo de grandes cantidades de datos.

**Postman:** Se utilizo para las pruebas de API de las funcionalidades del aplicativo para agilizar el tiempo de pruebas.

**Bootstrap:** Es que se utilizó en el aplicativo framework de diseño de frontend que facilita la creación de interfaces web atractivas.

**Git:** es un software de control de versiones pensando en la eficiencia la confiabilidad y la compatibilidad del mantenimiento de versiones de aplicaciones cuando se tiene un gran número de archivos de código fuente.

**Json:** Es un formato de texto sencillo para el intercambio de datos, se trata de un subconjunto de notación literal de objetos de JavaScript.

# VERSIONA MIENTO Y GESTIÓN DE TRABAJO

Para mejorar la eficacia del trabajo y el versiona miento de este en el equipo se usó Github ya que tiene la fácil de crear espacios de trabajo para los integrantes del equipo así todos pueden tener su espacio propio para pruebas y más sin afectar a otros dándole el control de que se integra al proyecto principal al administrador de esta integridad con visual studio code, además visual studio code trae la facilidad de dichas integraciones con la instalación de la herramienta de git, el repositorio del proyecto es el siguiente <https://github.com/andrwe455/veterinaria.git>

## COMANDOS BÁSICOS NPM

Para el correcto manejo y además para pruebas a futuro se mostrarán los comandos básicos para el uso de estos se debe usar la terminal que brinde el entorno de desarrollo en este caso visual studio code:

* npm init: Inicia un nuevo proyecto Node.js.
* npm install [Dependencia que se necesita]: Instala dependencias del proyecto.
* npm start: inicia la aplicación.
* npm test: Ejecuta pruebas.
* npm -run dev: inicia el servidor.
* Ctrl + c: termina procesos que se estén ejecutando.

# VISTA DE CARPETAS.

# CODIGO

En la raíz se verán varios archivos donde el más relevante es el de index.HTML, es un archivo HTML que sirve como punto de entrada principal para una aplicación web. Es la página principal que se carga automáticamente cuando un usuario visita un sitio web.

## CARPETA JS

En esta seccione se explicará la estructura y funcionamiento de los códigos que se implementan en el aplicativo web, ubicados en la carpeta js.

## APP.JS

Aquí se establece un servidor web utilizando Express configurando la conexión a MongoDB, primero se importan los módulos necesarios para la creación del servidor.

const express = require('express');

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

const cors = require('cors');

Se crea una instancia de la aplicación Express que se utiliza para configurar el servidor.

const app = express();

Se configura la conexión a MongoDB Atlas desde el archivo conexión.js.

const conexion = require('./database/conexion.js');

conexion.connect();

Se configura el arranque del servidor.

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Servidor escuchando en el puerto ${PORT}`);

});

Configuración de express.

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

app.use(bodyParser.json());

app.use(cors(({ origin: '\*' })));

app.use(express.json());

app.use(express.static('public'));

Importa las rutas desde el archivo routes.js y las utiliza en la raíz de la aplicación.

const router =require('./rutas/routes.js');

app.use('/', router);

## LOGIN.JS

Se ubica en la raíz siendo un archivo js es un evento de escucha de envío de formulario en JavaScript, se realiza una solicitud POST al servidor con los datos del formulario (correo electrónico y contraseña). Si la respuesta del servidor tiene un estado 200, redirige al usuario a la página indexAdmin.html y muestra un mensaje de alerta. En caso de error, muestra el error en la consola.

const lform =document.getElementById("loginForm");

lform.addEventListener("submit", async function (event) {

event.preventDefault(); // Evitar que el formulario se envíe automáticamente

var email = document.getElementById("email").value;

var password = document.getElementById("pswd").value;

var formData = {

Email: email,

password: password

};

let resp;

try {

resp=await fetch('http://localhost:3000/login', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify(formData)

});

if (resp.status==200) {

window.location.href = "../Front/indexAdmin.html";

}

alert("Usuario loggeado correctamente");

} catch (error) {

console.error('Error:', error);

}

console.log(resp);

return resp;

});

## CONTROLLER

Se ubica en la carpeta llamada controller y el archivo tiene el nombre de controlador.js básicamente son las implementaciones de las operaciones CRUD.

**Importación de Modelos y Esquemas:** Se importan modelos y esquemas que representan entidades en la base de datos, como usuarios, mascotas, historias y citas.

const usuario = require('../schemas/usuario.js');

const mascota = require('../schemas/mascotas.js');

const historias = require('../schemas/historias.js');

const cita = require('../schemas/citas.js');

**Función de signup:** Registra de manera asíncrona nuevos usuarios en la base de datos realizando validaciones para evitar duplicar usuarios.

async function signup(req, res) {

console.log('Solicitud POST recibida en /nuevo');

try {

const nuevousuario = new usuario(req.body);

const Email = req.body.Email;

const user = await usuario.findOne({Email});

try {

if (user!==null) {

if (Email === user.Email) {

res.status(500).json({error: 'El usuario ya existe'});

console.log("El usuario ya existe");

} else {

await nuevousuario.save();

return res.status(200).json(nuevousuario);

}

}

else {

await nuevousuario.save();

console.log(nuevousuario);

return res.status(200).json(nuevousuario);

}

} catch (error) {

console.log(error);

res.status(500).json({error: 'Error al crear el servicio'});

}

} catch (error) {

res.status(500).json({error: 'Error al crear el servicio'});

}

}

**Función login**: Autentifica a los usuarios verificando el correo electrónico y la contraseña según el resultado de autentificación da respuesta.

async function login(req, res) {

const { Email,password }=req.body;

try {

const user = await usuario.findOne({Email});

if(!user){

return res.status(404).json({msg:"El usuario no existe"});

}

if(user.password !== password){

return res.status(401).json({msg:"Password incorrecto"});

}

if (user.password === password )

{

if (user.role === "admin") {

console.log("Usuario logeado correctamente");

return res.status(200).json({msg:"Usuario logeado correctamente"});

}

else

{

return res.status(403).json({msg:"Usuario logeado correctamente"});

}

}

}catch (error) {

res.status(500).json({error: 'Error al crear el servicio'});

}

}

**Función Crearmascotas**: Crear nuevas entradas de mascotas en la base de datos.

async function Crearmascota (req, res) {

console.log('Solicitud POST recibida en /crearmascota');

try {

const nuevaMascota = new mascota(req.body);

await nuevaMascota.save();

res.status(201).json(nuevaMascota);

}

catch (error) {

res.status(500).json({error: 'Error al crear el servicio'});

}

}

**Funciones Crearmascota y Crearhistoria**: Crea nuevas entradas de mascotas e historias de la base de datos respectivamente su estructura es similares difiriendo en la solicitud POST.

async function Crearmascota (req, res) {

console.log('Solicitud POST recibida en /crearmascota');

try {

const nuevaMascota = new mascota(req.body);

await nuevaMascota.save();

res.status(201).json(nuevaMascota);

}

catch (error) {

res.status(500).json({error: 'Error al crear el servicio'});

}

}

**Función getmascotas**: Recupera las mascotas almacenadas en la base de datos.

async function getmascotas (req,res) {

console.log('Solicitud GET recibida en /mascotas');

try {

const mascotas = await mascota.find();

return res.json(mascotas);

}

catch (error) {

res.status(500).json({error: 'Error al crear el servicio'});

}

}

**Función eliminar**: Elimina una mascota especifica de la base de datos.

async function eliminar (req,res) {

console.log('Solicitud DELETE recibida en /eliminarMascota');

try {

const id = req.body;

console.log(id);

await mascota.deleteOne({\_id:id});

res.status(200).json({msg:"Mascota eliminada correctamente"});

} catch (error) {

res.status(500).json({error: 'Error al crear el servicio'});

}

}

**Función Asignarcita**: Crea y guarda nuevas citas en la base de datos.

async function Asignarcita (req,res) {

console.log('Solicitud POST recibida en /asignarcita');

try {

const nuevaCita = new cita(req.body);

await nuevaCita.save();

res.status(201).json(nuevaCita);

}

catch (error) {

res.status(500).json({error: 'Error al crear el servicio'});

}

}

**Exportación de funcionalidades.**

module.exports = {

signup,

login,

Crearmascota,

Crearhistoria,

getmascotas,

eliminar,

Asignarcita

}

**Manejo de errores**: se incluye bloques para manejar posibles errores durante las operaciones de la base de datos arrojando mensajes de error.

## CONEXIÓN

Se ubica en la carpeta database y el nombre el archivo es conexión.js manejando la conexión y desconexión a la base de datos MongoDB utilizando el paquete mongoose que es una biblioteca de modelado de objetos de MongoDB para Node.js que facilita la interacción con la base de datos asi como la función connect que establece la conexión con la base de datos usando una URL de conexión; la función disconnect que desconecta la aplicación a la base de datos con mensaje de confirmación y por último la exportación de las funciones ya mencionadas.

const mongoose = require('mongoose');

async function connect () {

try { mongoose.connect(`mongodb+srv://andresgutierrez83111:fO1RwkAfBKaGKFA8@cluster0.labzm9a.mongodb.net/taller2?retryWrites=true&w=majority`);

console.log('DB ONLINE');

} catch (error) {

console.log(error);

throw new Error('Error a la hora de inicializar base de datos');

}

}

async function disconnect (req, res) {

try {

mongoose.disconnect();

res.status(200).json({msg:"Desconexion exitosa"});

console.log('DB OFFLINE');

} catch (error) {

console.log(error);

throw new Error('Error a la hora de desconectar base de datos');

}

}

module.exports = {

connect,

disconnect

}

## RUTAS

Se sitúa en la carpeta rutas con nombre de archivo router.js, este código configura el enrutamiento del aplicativo web utilizando express.

**Importación de módulos:** Aquí se importan varias funcionalidades de express facilitando la conexión con el servidor.

const { express } = require('express');

const {Router} = require('express');

const {check, validationResult} = require('express-validator');

**Importación del controlador y conexión a base de datos**: Importa la ruta del módulo que contiene las funciones del controlador y el otro la ruta donde se realiza la conexión a la base de datos.

const controlador = require('../controller/controlador');

const conexion = require('../database/conexion.js');

**Solicitudes al servido**r: Definimos varias rutas usando nuestro enrutador. Cada ruta tiene asociada una operación específica (POST, GET, UPDATE y DELETE) y está vinculada a una función controladora en el módulo controlador donde:

* Las solicitudes POST se utilizan para agregar información a la base de datos.
* Las solicitudes UPDATE se utilizan para actualizar información en la base de datos
* Las solicitudes GET se utilizan para obtener información de la base de datos.
* Las solicitudes DELETE se utilizan para eliminar un recurso especifico en la base de datos.

router.post('/nuevo', controlador.signup);

router.post('/login', controlador.login);

router.post('/crearmascota', controlador.Crearmascota);

router.post('/crearhistoria',controlador.Crearhistoria);

router.post('/asignarcita',controlador.Asignarcita);

router.get("/mascotas",controlador.getmascotas);

router.delete("/eliminarMascota",controlador.eliminar);

**Exportación de enrutador:** Controla el enrutamiento del backend.

module.exports = router;

ESQUEMAS

Se ubica en la carpeta se schemas que contiene varios archivos que son citas.js, historias.js, mascotas.js y usuario.js el código que contiene la definición de las entidades en la base de datos se explicara a continuación la estructura con citas.js.

**Importa el módulo de mongoose.**

const mongose = require('mongoose');

**Definición del esquema**: Se define el esquema con sus campos correspondientes en este caso serian \_id, Nombre\_Mascota, Nombre\_Duenno, Fecha, Hora. Este citasSchema sirve como un molde para los dicuemntos que se guardaran en la colección Citas.

const citasSchema = new mongose.Schema({

\_id: String,

Nombre\_Mascota: String,

Nombre\_Duenno: String,

Fecha: Date,

Hora: String,

Motivo: String

});

**Creación del modelo**: se utiliza para realizar operaciones CRUD en la colección "Citas".

const citas = mongose.model('citas', citasSchema, 'Citas');

**Exportación del modelo**: para poder ser utilizado en otras partes de la aplicación.

module.exports = citas;

La estructura de los demás es similar variando en los campos.

## CARPETA FRONT

Aquí se ubica todo lo que es la parte de frontend que es o visual de la pagina en esta carpeta de conforma por la carpeta de admin y la carpera página.

En la raíz de la carpeta Front están los diferentes index.HTML de las vistas como lo son el de admin, Cliente, User, Login y Service. Cada uno de estas siendo una pantalla del aplicativo web.

## Carpeta Pagina

Se divide en cuatro carpetas una de css y otra de js que es donde se sitúa los archivos de .css y .j que usa Bootstrap que es el framework Las otras dos son la de fonts que contiene los tipos de letras que utiliza el aplicativo web y la de images donde están las imágenes que se ven en la página.