

31 de marzo: Diseño de vistas utilizando PowerPoint

LOGIN

Ingresar a su cuenta

Correo electrónico

Contraseña

Iniciar sesión

Crear una nueva cuenta

Nombre y apellido

Correo electrónico

Contraseña

Repetir contraseña

Registrar

INICIO

Fletes

Inicio

Nuevo flete

Reportes

{nombre de usuario}

Editar usuario

Cerrar sesión

Bienvenido, {nombre de usuario}

Agregar un nuevo flete

Ver reportes de fletes

REALIZAR FLETES

Fletes

Inicio

Nuevo flete

Reportes

{nombre de usuario}

Editar usuario

Cerrar sesión

Agregar un nuevo flete

Fecha del pedido

dd/mm/yyyy

Fecha de envío

dd/mm/yyyy

Costo

Ingresar valor numérico...

Cliente

Ingresar texto...

Descripción

Ingresar texto...

Guardar flete

The screenshot shows the 'Reportes' menu, which is highlighted in blue. The menu items are 'Fletes', 'Inicio', 'Nuevo flete', 'Reportes', and '{nombre de usuario}'. The '{nombre de usuario}' item is expanded, showing a dropdown menu with two options: 'Editar usuario' and 'Cerrar sesión'.

[illegible]

Editar usuario

Fletes
Inicio
Nuevo flete
Reportes
{nombre de usuario}

Datos actuales

Nombre: {nombre del usuario}

Correo electrónico: {correo electrónico del usuario}

Editar información (dejar en blanco si no desea modificar)

Nombre y apellido

Correo electrónico

Contraseña actual

Nueva contraseña

Repetir nueva contraseña

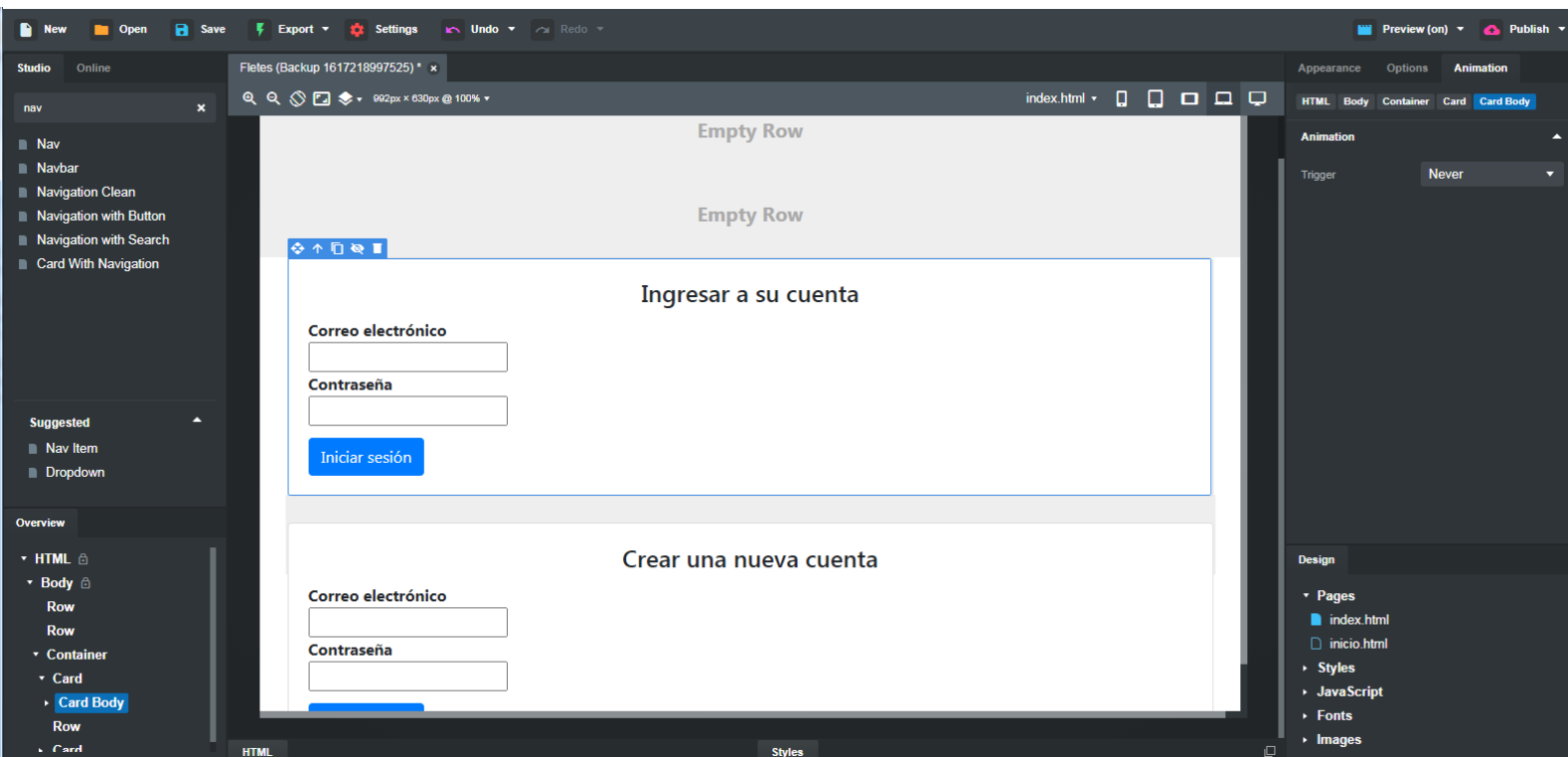
Editar usuario

Cerrar sesión

Actualizar

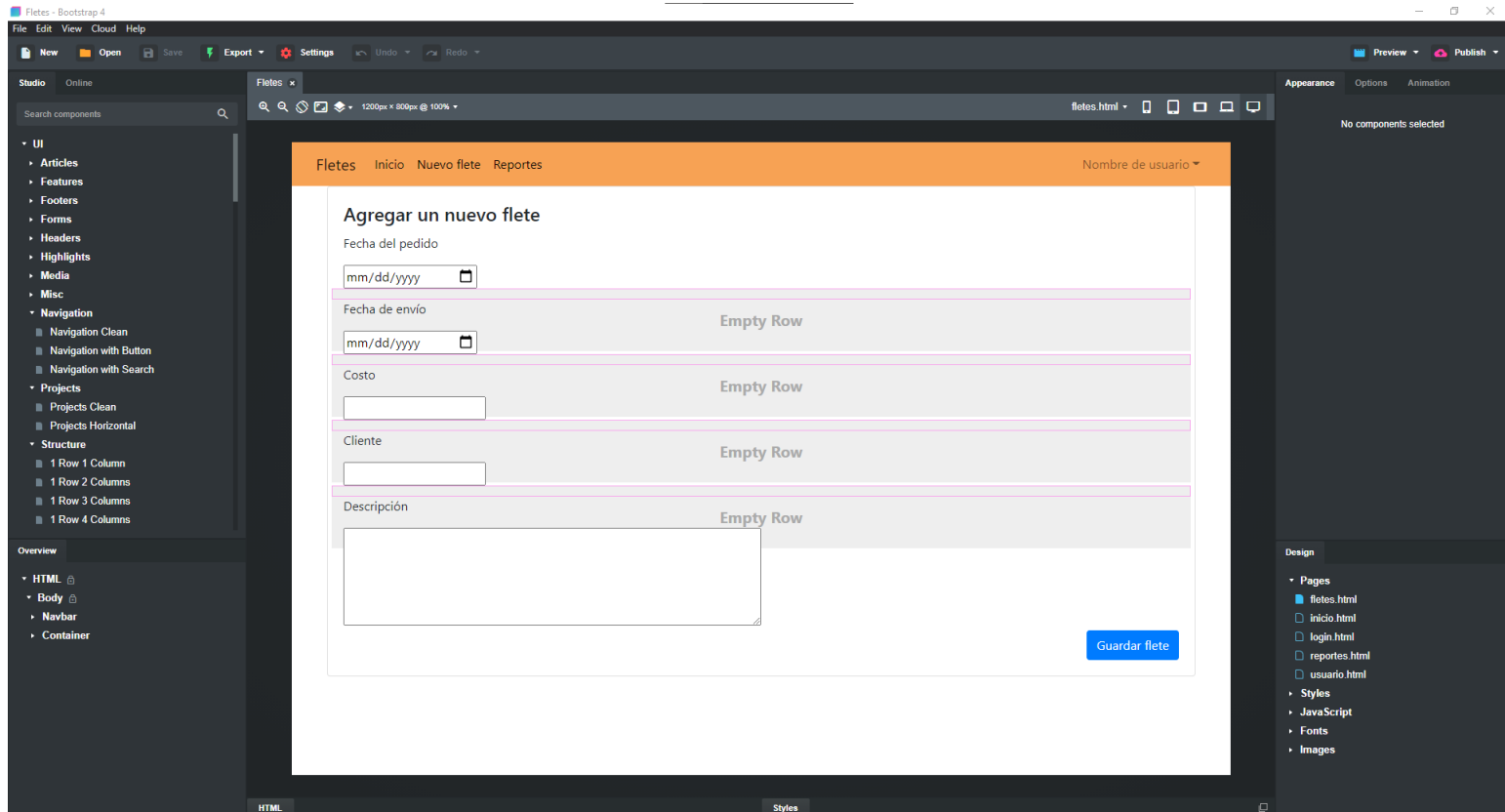
Generación de vistas utilizando Bootstrap Studio

login.html

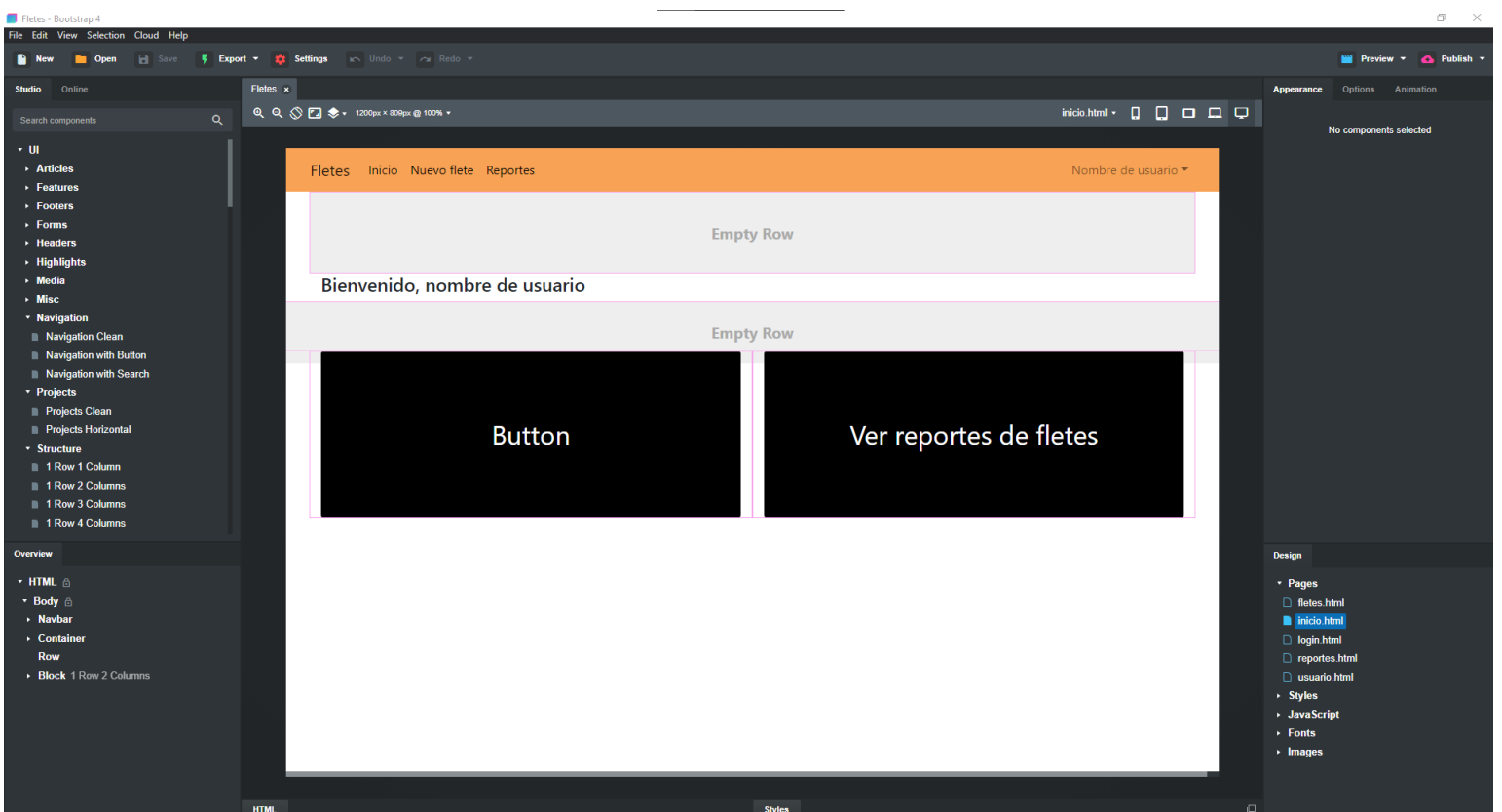


12 de abril: Generación de vistas usando Bootstrap Studio

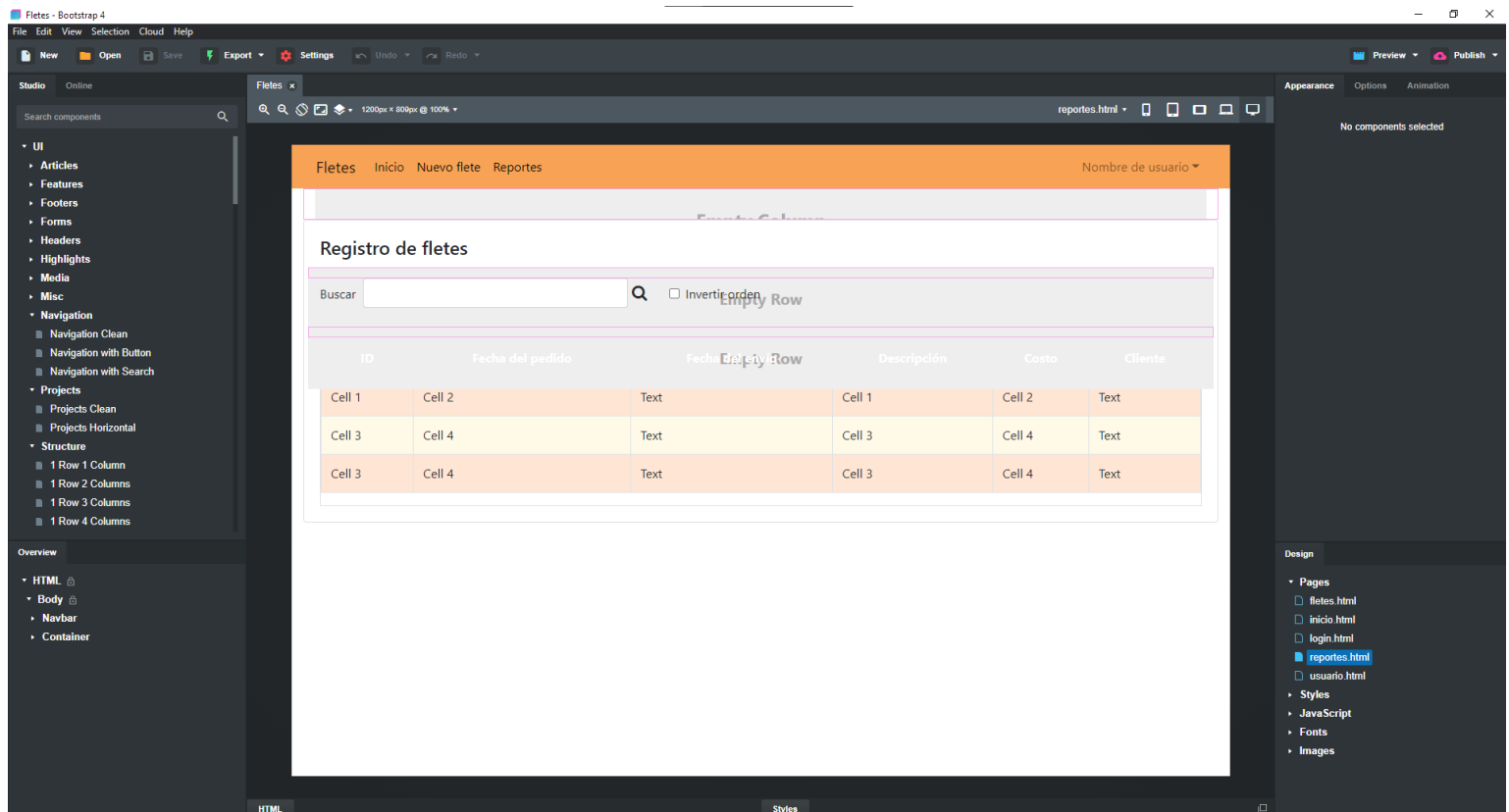
fletes.html



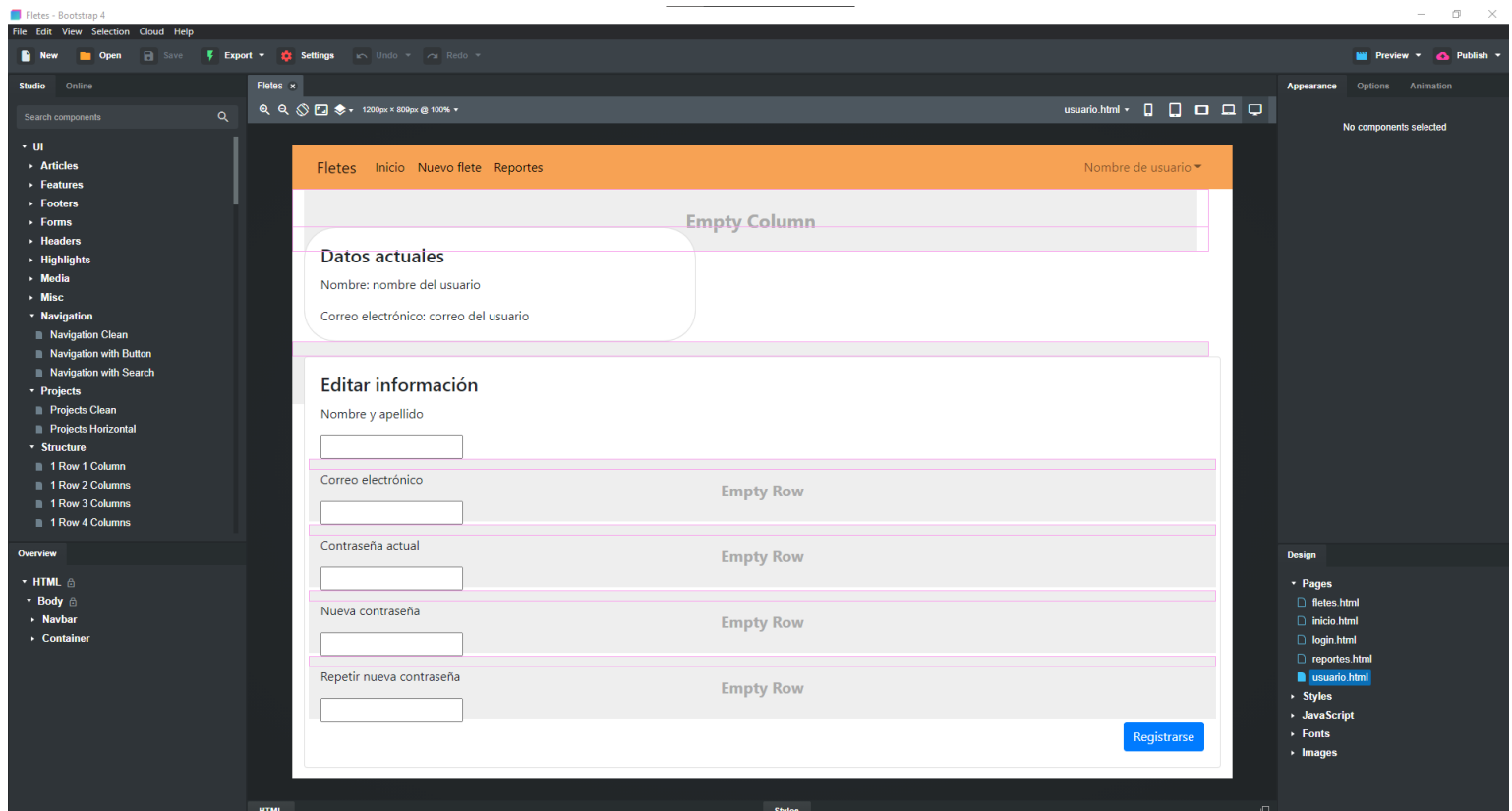
inicio.html



reportes.html



usuario.html



13 de abril: Terminé de crear las vistas usando Bootstrap Studio y las exporté a HTML para luego pasarlas a PHP. Creamos controladores básicos para empezar a usar Codeigniter.

localhost/codeigniter/index.php/Fletes

Fletes Toggle navigation

- Inicio
- Nuevo flete
- Reportes
- Nombre de usuario
- Editar usuario
- Cerrar sesión

Agregar un nuevo flete

Fecha del pedido

dd/mm/aaaa

Fecha de envío

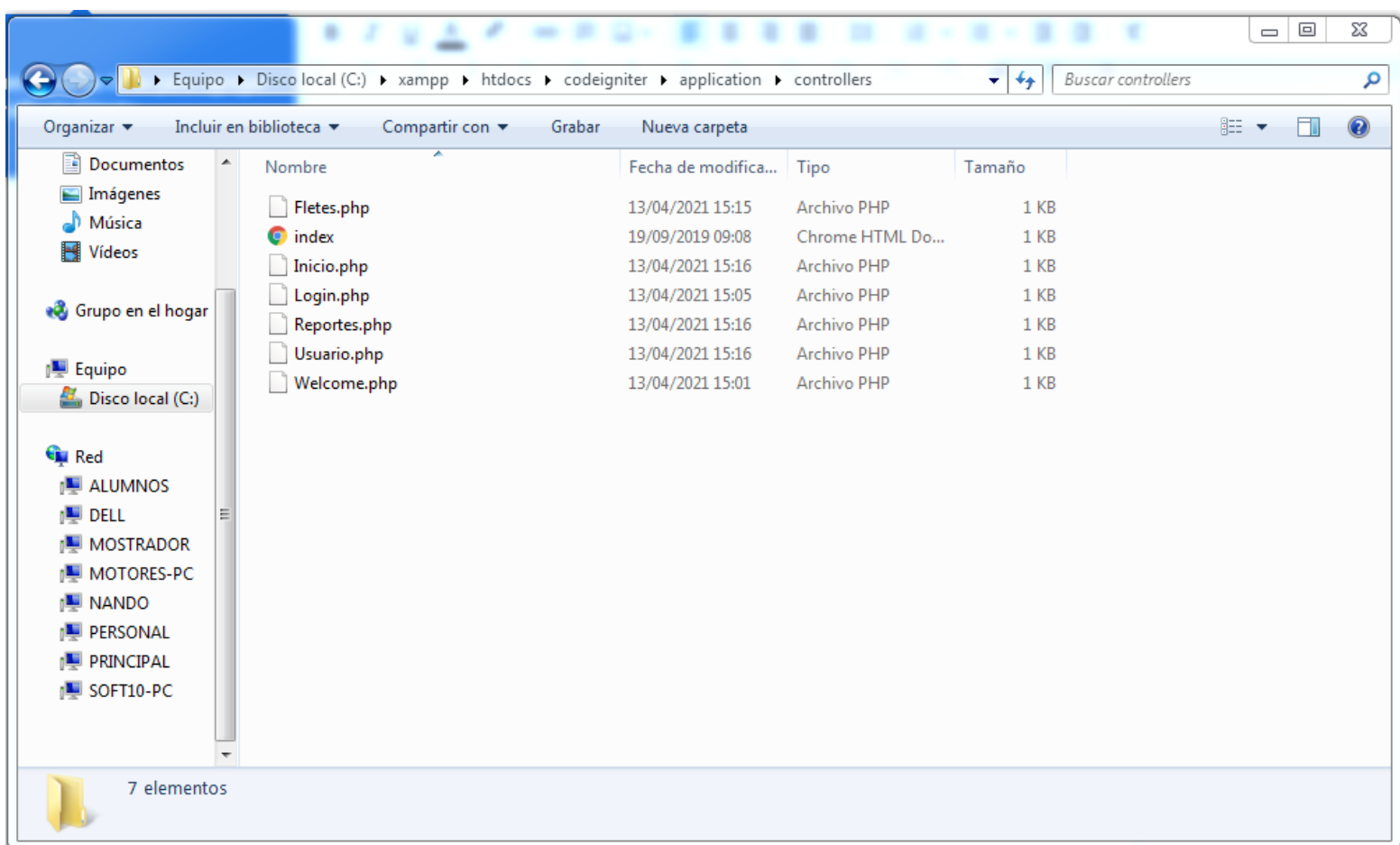
dd/mm/aaaa

Costo

Cliente

Descripción

Guardar flete



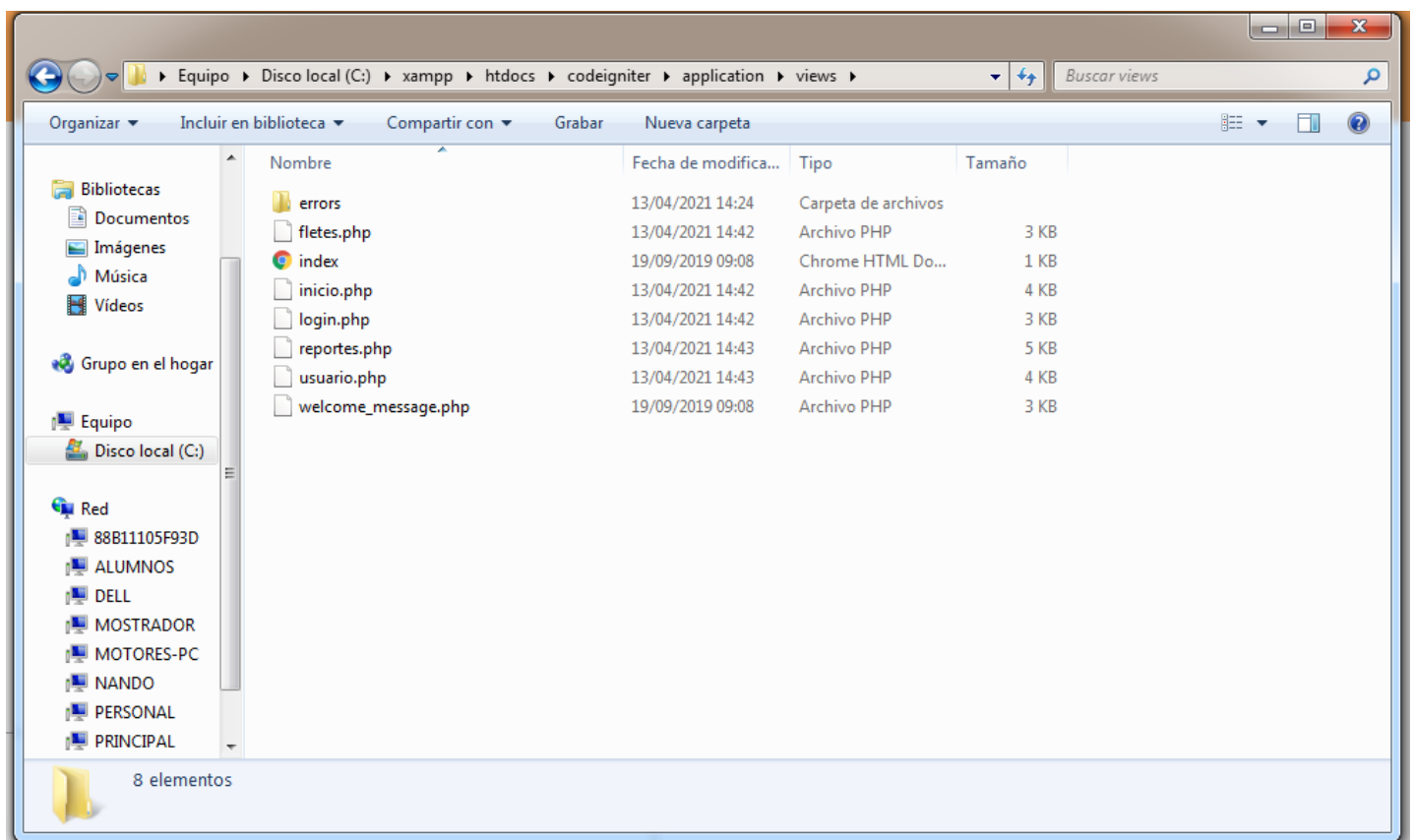
C:\xampp\htdocs\codeigniter\application\controllers\Usuario.php - Notepad++

Archivo Editar Buscar Vista Codificación Lenguaje Configuración Macro Ejecutar Plugins Ventana ?

Inicio.php welcome_message.php Login.php database.php Fletes.php Reportes.php Usuario.php

```
1 <?php
2 defined('BASEPATH') OR exit('No direct script access allowed');
3
4 class Usuario extends CI_Controller {
5
6     /**
7      * Index Page for this controller.
8      *
9      * Maps to the following URL
10      *      http://example.com/index.php/welcome
11      * - or -
12      *      http://example.com/index.php/welcome/index
13      * - or -
14      * Since this controller is set as the default controller in
15      * config/routes.php, it's displayed at http://example.com/
16      *
17      * So any other public methods not prefixed with an underscore will
18      * map to /index.php/welcome/<method_name>
19      * @see https://codeigniter.com/user_guide/general/urls.html
20      */
21     public function index()
22     {
23         $this->load->view('usuario');
24     }
25 }
26 ?>
```

PHP Hypertext Preprocessor file length : 673 lines : 26 Ln : 23 Col : 35 Sel : 0 UNIX ANSI INS



14 de abril: Diseñé la base de datos a usar, añadí los CSS a todas las vistas y seguí conectando los controladores a las views. Además añadí el primer modelo, que se encarga de subir nuevos fletes a la base de datos.

Base de datos: tabla fletes

Servidor: 127.0.0.1 » Base de datos: fletes » Tabla: fletes							
Examinar Estructura SQL Buscar Insertar Exportar Importar Privilegios Op							
Estructura de tabla Vista de relaciones							
	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado
<input type="checkbox"/>	1	id_flete	int(11)			No	Ninguna
<input type="checkbox"/>	2	direccion	varchar(50)	latin1_general_ci		No	Ninguna
<input type="checkbox"/>	3	costo	float			No	Ninguna
<input type="checkbox"/>	4	costo_adicional	float			No	Ninguna
<input type="checkbox"/>	5	fecha_pedido	date			No	Ninguna
<input type="checkbox"/>	6	fecha_entrega	date			No	Ninguna
<input type="checkbox"/>	7	tipo	int(10)			No	Ninguna

Base de datos: tabla usuario

Servidor: 127.0.0.1 » Base de datos: fletes » Tabla: usuario							
Examinar Estructura SQL Buscar Insertar Exportar Importar Privilegios Op							
Estructura de tabla Vista de relaciones							
	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado
<input type="checkbox"/>	1	id_usuario	int(11)			No	Ninguna
<input type="checkbox"/>	2	nombre	varchar(20)	latin1_general_ci		No	Ninguna
<input type="checkbox"/>	3	apellido	varchar(20)	latin1_general_ci		No	Ninguna
<input type="checkbox"/>	4	usuario	varchar(25)	latin1_general_ci		No	Ninguna
<input type="checkbox"/>	5	correo	varchar(40)	latin1_general_ci		No	Ninguna
<input type="checkbox"/>	6	password	varchar(50)	latin1_general_ci		No	Ninguna
<input type="checkbox"/>	7	rep_password	varchar(50)	latin1_general_ci		No	Ninguna

Modelo: crud_model (para cargado de fletes a la base de datos)

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');
class Crud_model extends CI_Model
{
    function saverecords($data)
    {
        $this->db->insert('fletes',$data);
        return true;
    }

    function saveuser($data)
    {
        $this->db->insert('usuario',$data);
        return true;
    }
}
?>
```


Controlador “fletes” para el cargado de datos a la DB

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');
class Crud extends CI_Controller
{
    public function __construct(){
        parent::__construct();
        $this->load->database();
        $this->load->model('Crud_model');
        $this->load->model('Crud_model_update');
        $this->load->model('Crud_model_select');
    }


    /*Insert*/
    public function savedata(){
        $this->load->view('fletes');
        if($this->input->post('save'))
        {
            $data['fecha_pedido']=$this->input->post('fechapedido');
            $data['fecha_entrega']=$this->input->post('fechaenvio');
            $data['costo']=$this->input->post('costo');
            $data['costo_adicional']=$this->input->post('costoadicional');
            $data['direccion']=$this->input->post('direccion');
            $data['descripcion']=$this->input->post('descripcion');
            $response=$this->Crud_model->saverecords($data);
            if($response==true){
                echo "Records Saved Successfully";
            }
            else{
                echo "Insert error !";
            }
        }
    }
}
```

Vista correspondiente


Fletes Inicio Nuevo flete Reportes

Agregar un nuevo flete

Fecha del pedido



Fecha de envío



Costo

Costo adicional

Dirección

Descripción

24 de abril: Seguí programando las funciones de la página. Se incluyen capturas de pantalla del código a continuación:

Función de login, registrado y cambiado de datos del usuario

```
<?php
class Crud_update extends CI_Controller
{
    public function __construct(){
        parent::__construct();
        $this->load->database();
        $this->load->model('Crud_model_update');
    }

    public function dispdata(){
        $mail=$this->input->post('correo');
        $password=$this->input->post('currentpassword');
        if($this->Crud_model_update->check_usuario($mail, $password) == true){
            $this->updatedata($mail, $password);
        }else{
            echo 'Asegúrese de haber ingresado datos correctos';
        }
        $this->load->view('usuario');
    }

    public function updatedata($mail, $password){
        if($this->input->post('update')){
            $nombre=$this->input->post('nombre');
            $newpassword=$this->input->post('newpassword');
            $repetirpassword=$this->input->post('repetirpassword');
            $result=$this->Crud_model_update->displayrecordsByMail($mail, $password);
            $id=$result[0]->id_usuario;
            $this->Crud_model_update->update_records($nombre, $mail, $newpassword, $repetirpassword, $id);
            $this->session->set_userdata('username', $nombre);
            echo 'Datos actualizados exitosamente';
        }
    }
}
```

Funciones SELECT
(mostrar nombre,
mostrar reporte de fletes,
buscar
reportes cuando
el usuario usa la
searchbar)

```
<?php
class Crud_select extends CI_Controller
{
    public function __construct(){
        parent::__construct();
        $this->load->database();
        $this->load->model('Crud_model_select');
    }

    public function getname(){
        $result['nombre']=$this->Crud_model_select->get_name();
        $this->load->view('reportes', $result);
    }

    public function displaydata(){
        $result['data']=$this->Crud_model_select->display_records();
        $this->load->view('reportes',$result);
    }

    public function displaydataordered(){
        $result['data']=$this->Crud_model_select->display_records_ordered();
        $this->load->view('reportes',$result);
    }

    public function search(){
        $search = $this->input->post('search');
        $result['data'] = $this->Crud_model_select->display_records_search($search);
        $this->load->view('reportes',$result);
    }
}
```

25 de abril: Se probaron todas las vistas.

Proyecto RASN

28 de abril

Definimos el proyecto en clase.

4 de mayo

Trabajamos la fundamentación y resultado esperado. Además empezamos a trabajar en el frontend (página principal) y en el backend (base de datos).

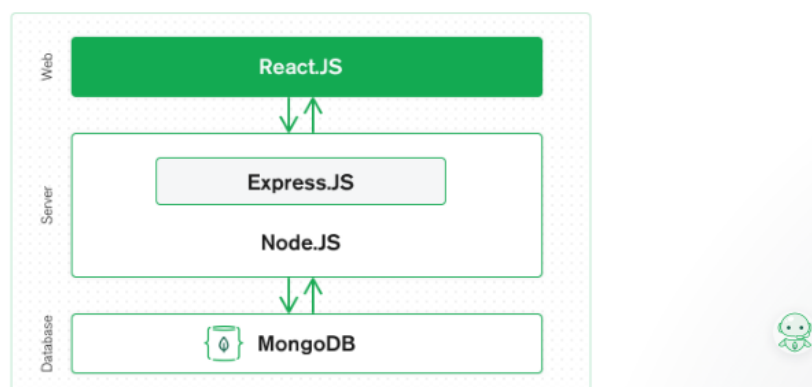
Base de datos: tabla animales

12 de mayo

Nos juntamos con los profesores para discutir las tecnologías a usar. Determinamos que Codeigniter no será nuestra solución de backend, además hablamos de otras posibilidades para el frontend y backend. Nos decidimos por usar el stack MERN, basado en JavaScript. Además acordamos utilizar GitHub como nuestra solución de control de versiones y de coordinación de tareas.

How does the MERN stack work?

The MERN architecture allows you to easily construct a 3-tier architecture (frontend, backend, database) entirely using JavaScript and JSON.



18 de mayo

Nos juntamos de nuevo con los profesores para determinar los requerimientos de entrega para el primer checkpoint (26 de mayo). Nos distribuimos las tareas restantes para poder entregar todo dentro de los plazos.

24 de mayo

Realicé el diagrama de Gantt con las tareas hechas y por hacer (puede verse en [este link](#))

25 de mayo

Organicé la carpeta de campo del proyecto en preparación para la entrega de mañana. Queda pendiente actualizar para incluir los avances que se realizaron hoy. La carpeta puede verse en [este link](#).

1 de junio

Actualicé la carpeta para la entrega de mañana (ya que se movió una semana).

2 de junio

Tuvimos la devolución, hablamos con los profesores. El principal feedback fue que empezáramos con el backend y también con las funcionalidades faltantes del frontend.

5-8 de junio

Estuve realizando un curso de MongoDB en la página [codigofacilito.com](#). Esto en preparación para poder armar las bases de datos que necesitará la página.

9 de junio

Inicié la base de datos en MongoDB Atlas. Además probamos el método de conexión desde diferentes computadoras y diseñé un schema preliminar.

The screenshot shows the MongoDB Atlas interface for a cluster named 'clusterRASN' in the 'Sandbox' environment. The cluster is a 'Shared Tier Cluster' (M0 Sandbox) with version 4.4.6. It is located in the 'Azure / Virginia-East2 (eastus2)' region and is configured as a 'Replica Set - 3 nodes'. The cluster tier is 'M0 Sandbox (General)'. The logical size is 0.0 B, and the logical size bar chart shows 0.0 B out of a 512.0 MB max. The operations bar chart shows 0 R, 0 W, and 0 operations over the last 6 hours. The connections bar chart shows 0 connections out of a 500 max over the last 6 hours. The page includes a sidebar with navigation links for Data Storage, Clusters, Triggers, Data Lake, Security, Database Access, Network Access, and Advanced. The top navigation bar includes links for Team RASN, Access Manager, Billing, All Clusters, Get Help, and Samuel. The bottom of the page features a 'Get Started' button, a 'System Status: All Good' message, and a footer with copyright information for MongoDB, Inc. and links to Status, Terms, Privacy, Atlas Blog, and Contact Sales.

16 de junio

Configuré una verificación para los atributos que se insertan en cada colección utilizando la opción de Schema Validation de MongoDB. Cada una de las colecciones tiene su propia validación para comprobar que los datos que se insertan sean apropiados.

El resto de los validators utilizados se pueden encontrar en [este link](#).

22 a 29 de junio

Empecé a ingresar datos de prueba en la base de datos, pero no pude acceder directamente a la DB ya que estaba en la escuela y no se puede conectar a la misma.

Por lo tanto, utilicé la interfaz web de Atlas para añadir datos a la DB.

```
db.createCollection( "animales", {
  validator: { $jsonSchema: {
    bsonType: "object",
    required: [ "nombre", "color", "sexo", "peso"], //atributo requerido
    properties: {
      nombre: {
        bsonType: "string", //tipo de cada atributo
        description: "string requerido"
      },
      color: {
        bsonType: "string",
        description: "string requerido"
      },
      sexo: {
        enum: [ "M", "F" ], //solo puede ser uno de los dos
        description: "Solo puede ser uno de los dos"
      },
      peso: {
        bsonType: "double",
        description: "número requerido (se guarda en kg)"
      },
      fechaNacimiento: {
        bsonType: "date",
        description: "fecha no requerida"
      },
      raza: {
        bsonType: "string",
        description: "string no requerido"
      },
      actitud: {
        bsonType: "string",
        description: "string no requerido"
      },
      tamano: {
        enum: ["Pequeño", "Mediano", "Grande"],
        description: "Solo puede ser una de las opciones."
      },
      imagen: {
        bsonType: "string",
        description: "link a la imagen (no requerido)"
      }
    }
  }
})
```

```
var animal1 = {
  nombre: "Pepi",
  color: "Marrón",
  sexo: "M",
  peso: 8
}

var animal2 = {
  nombre: "Husky",
  color: "Blanco",
  sexo: "F",
  peso: 10,
  fechaNacimiento: new Date(2018, 0, 13),
  raza: "Pastor blanco",
  tamano: "Mediano",
  imagen: "https://upload.wikimedia.org/wikipedia/commons/thumb/3/30/Schweizer_Sch%C3%A4ferhund%2C_9_Monate.JPG/220px-Schweizer_Sch%C3%A4ferhund%2C_9_Monate.JPG"
}
```

Los datos ingresados pueden encontrarse en [este link](#).

6 de julio

Con la base de datos terminada, empecé a escribir código en Node.js para poder establecer una conexión con la DB. Una vez se ha establecido la conexión, se puede trabajar el CRUD para integrar con el frontend. El trabajo realizado puede encontrarse en [este link](#).

12 y 13 de julio

Al volver a prácticas profesionalizantes presenciales, nos dimos cuenta de que no podríamos conectarnos a la DB desde el colegio, por lo cual iba a ser necesario hacer un dump de la DB para poder habilitarla en offline en la escuela.

Receso de invierno (19 a 30 de julio)

3 y 4 de agosto

Mientras se terminaban de habilitar los equipos de la escuela para poder trabajar con Node.js y la base de datos, estuve terminando de cargar datos de prueba a las colecciones faltantes.

```
_id: ObjectId("60d506aac0e9f26ba4e94eb8")
pregunta: "¿Dónde mantenemos a las mascotas?"
respuesta: "Las mascotas se encuentran resguardadas en un refugio alquilado. Nuest..."
```

En la imagen pueden verse los datos cargados a la colección de preguntas frecuentes (FAQ).

```
_id: ObjectId("60d50a39a3b4f743ac5559b9")
pregunta: "Quiero adoptar a una de las mascotas. ¿Qué debo hacer?"
respuesta: "Si quieres adoptar a una de las mascotas, puedes contactarte con nosot..."
```

```
_id: ObjectId("60d50a60a3b4f743ac5559ba")
pregunta: "Quiero ayudar como voluntario. ¿Qué debo hacer?"
respuesta: "Si quieres formar parte de nuestro equipo, contáctate con nosotros usa..."
```

```
_id: ObjectId("60d50ab9a3b4f743ac5559bb")
pregunta: "¿Cómo logramos financiar el refugio?"
respuesta: "El refugio se financia mediante donaciones de diversos tipos y eventos..."
```

10 y 11 de agosto

Dedicamos estos días a configurar el ambiente de trabajo en la escuela para poder avanzar. Instalamos los programas necesarios, generamos la base de datos a partir del dump que yo había creado el mes anterior y verificamos que nuestro trabajo hasta el momento funcione para poder avanzar.

18 de agosto

El día de hoy documentamos la estructura actual del proyecto, y a pedido del profesor hicimos un diagrama estructural de la base de datos.

Desarrollo de funciones y endpoints

24 de agosto

Trabajé en el envío de datos desde la base de datos MongoDB, hacia la interfaz gráfica compuesta por React. Diseñé las funciones de acuerdo a los requisitos del equipo de frontend, enviando los datos de una manera que resultara fácil de procesar

Determinamos que lo ideal sería exponer los datos mediante **endpoints** que permiten el acceso desde front. Configuramos el servidor para que ejecute una consulta de datos cuando se le solicita una determinada ruta, y

```
app.route('/Rasn/admin/animales').get(function (req, res) {  
  client.connect(function(err, db) {  
    if (err) throw err;  
    var dbo = db.db("proyectoRasn");  
    dbo.collection("animales").find({}).toArray(function(err, result) {  
      if (err) throw err;  
      console.log(result);  
      res.send(result);  
      db.close();  
    });  
  });  
});
```

exponga los datos devueltos en formato JSON.

En el caso que se muestra en la imagen, el servidor responde a un request de la ruta `"/Rasn/admin/animales"` realizando una consulta a la base de datos (en este caso, pidiendo todos los animales), y muestra los datos recibidos como respuesta en formato JSON. Con estos datos, el frontend puede operar.

De la misma manera se expusieron los datos de las colecciones de posts, integrantes y faq en sus respectivas rutas.

01 a 15 de septiembre - INSERT, UPDATE y DELETE

Una vez logramos comprobar que se podían extraer datos de todas las colecciones para mostrarlos en el frontend, pasamos a la adición, editado y borrado de entidades.

En primer lugar, implementamos un punto de acceso para borrar animales de la colección utilizando su ID. Utilizamos el método POST para no enviar el ID mediante la URL. La consulta quedó de la siguiente manera:

```
app.route("/Rasn/admin/animales/delete").post(async function (req, res) {
  try {
    await client.connect();
    const db = client.db("proyectoRasn");
    const collection = db.collection("animales");
    const data = req.body;
    const newData = {
      ...data
    }
    console.log(newData);
    const result = await collection.deleteOne({ _id: new mongodb.ObjectId(req.body.id)});
    //res.send(result);
    if (result.deletedCount === 1) {
      console.dir("Successfully deleted one document.");
      console.log({ _id: new mongodb.ObjectId(req.body.id)});
    } else {
      console.log("No documents matched the query. Deleted 0 documents.");
    }
  } catch (error) {
    console.log(error);
  } finally {
    await client.close();
  }
});
```

Puede observarse que se envía un objeto *ObjectId* para que la base de datos identifique a la entidad en cuestión.

En esta función para añadir un nuevo animal, vemos que la nueva entidad se inserta como un objeto JSON. La base de datos se encarga de traducir este JSON y volcarlo en la colección con sus atributos.

```
app.route("/Rasn/admin/animales/nuevo-animal").post(async function (req, res) {
  try {
    await client.connect();
    const db = client.db("proyectoRasn");
    const collection = db.collection("animales");
    const data = req.body;
    data.fechaNacimiento = new Date(data.fechaNacimiento);
    console.log(data);
    const result = await collection.insertOne(data);
    //res.send(result);
    if (result.insertedId !== undefined) {
      console.dir("Successfully added one document.");
      console.log(result.insertedId);
    } else {
      console.log("not inserted");
    }
  } catch (error) {
    console.log(error);
  } finally {
    await client.close();
  }
});
```

En esta función para actualizar los datos de un animal, vemos que se utiliza la función `replaceOne()`, que permite ubicar una entidad ya existente y reemplazarla por otra (pero manteniendo el mismo identificador). La base de datos recibe la estructura **data** y reemplaza los datos de la entidad que ya existía.

```
app.route("/Rasn/admin/animales/actualizar-animal").post(async function (req, res) {
  try {
    await client.connect();
    const db = client.db("proyectoRasn");
    const collection = db.collection("animales");
    const data = req.body;
    const id = data.id;
    data.fechaNacimiento = new Date(data.fechaNacimiento);
    console.log(data);
    const result = await collection.replaceOne({ _id: new mongodb.ObjectId(id)},
    data);
    //res.send(result);
    if (result.updatedCount === 1) {
      console.dir("Successfully updated one document.");
      console.log({ _id: new mongodb.ObjectId(data.id)});
    } else {
      console.log("No documents matched the query. Updated 0 documents.");
    }
  } catch (error) {
    console.log(error);
  } finally {
    await client.close();
  }
});
```