Universidad San Carlos de Guatemala

Manejo e Implementación de Archivos

Aux. Fernando Ocaña

David Enrique Lux Barrera 201931344

Manual Técnico

Tecnología Utilizada para la realización del programa:

- NodeJs
- Html
- Css
- JavaScript
- MongoDB

Requerimientos de Usuario

Dado que la aplicación fue desarrollada en el lenguaje de Java, utilizando varios elementos de análisis técnico y paradigmático, se necesita que el usuario cumpla con estos requerimientos:

1. Sistema Operativo Windows 7 o Superior de 32 bits o 64 bits o alguna distribución de Ubuntu o linux.
2. Mínimo 100 MB libres en el Disco Duro.
3. Mínimo 1 GB de Memoria RAM.
4. Instalar Visual Studio Code
5. Instalar Angular y Node.js

Solución Inicial

Para la solventar las peticiones de la base de datos:

- Diseñar la estructura de la base de datos: Es importante definir de antemano la estructura de la base de datos en MongoDB, incluyendo la creación de las colecciones y los campos que se utilizarán para almacenar la información de los productos, clientes, órdenes, etc.
- Crear la conexión con la base de datos: Para poder utilizar MongoDB en una aplicación de Ecommerce, es necesario establecer una conexión con la base de datos. Esto se puede hacer utilizando la biblioteca oficial de MongoDB para Node.js, Mongoose.
- Crear los modelos de datos: Para poder interactuar con la base de datos, es necesario crear los modelos de datos utilizando Mongoose. Estos modelos representan las colecciones en la base de datos y definen la estructura de los documentos que se almacenarán.
- Crear las rutas para las peticiones HTTP: Para permitir que los clientes de la aplicación realicen solicitudes a la base de datos, es necesario crear rutas HTTP para cada una de las operaciones CRUD (crear, leer, actualizar y eliminar). Por ejemplo, se podría crear una ruta para obtener la lista de productos, otra para agregar un nuevo producto al catálogo, y otra para actualizar la información de un producto existente.

- Implementar las funciones de controladores: Cada ruta HTTP debe tener una función de controlador asociada que se encargue de procesar la solicitud y devolver la respuesta adecuada. Estas funciones de controlador utilizan los modelos de datos de Mongoose para realizar operaciones CRUD en la base de datos.
- Probar y depurar la aplicación: Una vez que la aplicación esté implementada, es importante probarla exhaustivamente para asegurarse de que funciona correctamente y no presenta errores. Si se encuentra algún problema, se debe depurar el código para solucionar el problema.

Con estas soluciones iniciales, se puede comenzar a desarrollar una aplicación de Ecommerce que utilice MongoDB como base de datos. A partir de aquí, se pueden agregar más funcionalidades y mejorar el rendimiento según sea necesario.

Los pasos para utilizar un frontend y mostrar los datos obtenidos en el backend de MongoDB son los siguientes:

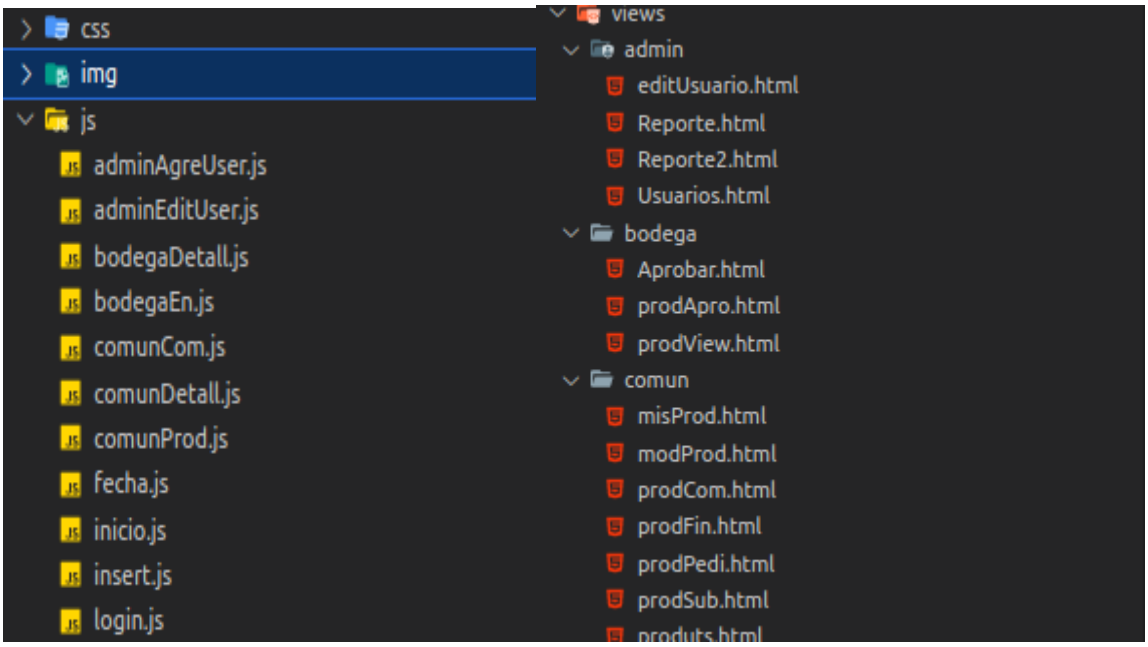
- Crear un proyecto de frontend utilizando la tecnología que prefieras, como React, Angular o Vue.js.
- Crear un archivo de configuración en tu proyecto para conectarse a la base de datos de MongoDB. Puedes utilizar el driver oficial de MongoDB para Node.js, Mongoose, o alguna otra librería.
- Escribir las consultas a la base de datos que necesitas para obtener los datos que quieres mostrar en el frontend. Por ejemplo, si quieres mostrar una lista de productos, necesitarás escribir una consulta que recupere los datos de la colección de productos.
- Una vez que tengas los datos, puedes utilizar la librería de frontend que estás utilizando para mostrar los datos en la página. Por ejemplo, si estás utilizando React, puedes crear un componente que muestre una lista de productos y pasarle los datos como propiedades.

Algunos consejos para mejorar el rendimiento de tu aplicación son:

- Utilizar la paginación para limitar la cantidad de datos que se muestran en una sola página.
- Utilizar caché para almacenar los datos que ya se han recuperado, de modo que no sea necesario volver a consultar la base de datos cada vez que se acceda a una página.
- Utilizar índices en la base de datos para mejorar la velocidad de las consultas.

Código utilizado

La distribución de los datos se debe establecer en carpetas para su mejor acceso, y mejor jerarquía entre las vistas.



Este código, es el método utilizado para poder conectarse al backend para hacer la consulta en el puerto establecido, de esta manera podemos ver los datos de los usuarios, por medio de la consulta FETCH que nos ayuda a encontrar la url de la información.

```
fetch(`http://localhost:3000/prod/detalles/${productId}`)
  .then(respuesta => respuesta.json())
  .then(datos => {
    const producto = datos;

    // Crear un contenedor para cada producto
    const productoContainer = document.createElement('div');
    productoContainer.classList.add('productosVer');

    // Crear un elemento HTML para mostrar la imagen del producto
    const imgElem = document.createElement('img');
    const rutaBase = 'http://localhost:3000';
    const rutaImagen = producto.img;
    imgElem.src = rutaBase + rutaImagen;

    // Agregar la imagen al contenedor de producto
    productoContainer.appendChild(imgElem);

    // Crear un elemento HTML para mostrar los datos del producto
    const productoElem = document.createElement('div');
    productoElem.innerHTML = `
      <br>
      <center><h3>${producto.nombre}</h3></center>
      <br>
      <p class="precio">Precio: Q${producto.precio}</p>
      <p>Descripcion: ${producto.descripcion}</p>
      <p>vendedor: ${producto.vendedor.nombre}</p>
      <p>Cantidad Existente: ${producto.cantidad_existente}</p>
      <p>Categorías: ${producto.categorias.join(', ')}</p>
      <p>Estado: ${producto.estado}</p>
    `;
```

Este es el formato para las consultas post del sistema en la que podemos obtener datos de la base de datos

```
const dpi = localStorage.getItem('dpi');

const dpiUsuarioElem = document.getElementById('dpiUsuar
//Para mostrarlo en el cuadro
dpiUsuarioElem.textContent = dpi;

// Hacer la petición HTTP POST al endpoint /buscar del b
fetch('http://localhost:3000/api/buscar', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({ dpi })
})
.then(respuesta => respuesta.json())
.then(datos => {
  // Mostrar el nombre, correo y cargo del usuario en la
  const nombreUsuario = datos.nombre_usuario;
  const correoUsuario = datos.correo_electronico;
  const cargoUsuario = datos.tipo_usuario;

  const nombreDiv = document.getElementById('nombreUsuar
  nombreDiv.innerText = `Nombre: ${nombreUsuario}`;

  const correoDiv = document.getElementById('correoUsuar
  correoDiv.innerText = `Correo: ${correoUsuario}`;

  const cargoDiv = document.getElementById('cargoUsuario
  cargoDiv.innerText = `Cargo: ${cargoUsuario}`;
})
.catch(error => console.log(error));
```

Aquí, mostramos la conexión utilizados para la conectarme a a base de datos no relacional de MongoDB:

```
//clase estatica para acceder a nuestras imagenes
app.use(express.static(path.join(__dirname, 'controllers')));
//app.use(multer({dest: path.join(__dirname,'controllers/img/imgproductos')}).single('image
//Connection to data base
async function start(){
    try{
        const db = await mongoose.connect('mongodb://localhost:27017/Ecomerce',{
            useNewUrlParser:true,
            useUnifiedTopology:true,
            family: 4
        });
        console.log('Connectado a la base de datos', db.connection.name);
    }catch (error){
        console.log('error');
    }
}
start();
```

Aquí, se establecen las rutas principales para poder acceder a la información de las rutas

```
//Routes
app.use('/api',usuarioRoutes);
app.use('/prod',productoRoutes);
app.use('/carr', carritoRoutes);
app.use('/pedido', pedRoutes);

app.listen(3000);
```

Estas son las librerías utilizadas para poder utilizar y llamar funciones que nos permite la mejor manipulación de datos

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const usuarioRoutes = require('./routes/usuario.routes')
const productoRoutes = require('./routes/producto.routes')
const carritoRoutes = require('./routes/carrito.routes')
const pedRoutes = require('./routes/pedido.routes')
const path = require('path');
const multer = require('multer');
```

Estas son las rutas utilizadas para acceder a consultas específicas según el controlador programado.

```
router.post('/carrito/:dpi', controller.carritoAgreg);
router.get('/carritoVer/:dpi', controller.obtenerCarrito);
router.get('/carritoId/:dpi/id', controller.obtenerIdCarritoPorDpi);
router.delete('/carritoEl/:carritoId/producto/:productoId', controller.eliminarProducto);
router.delete('/carritoTodo/:carritoId/productos', controller.eliminarProductosCarrito);
```

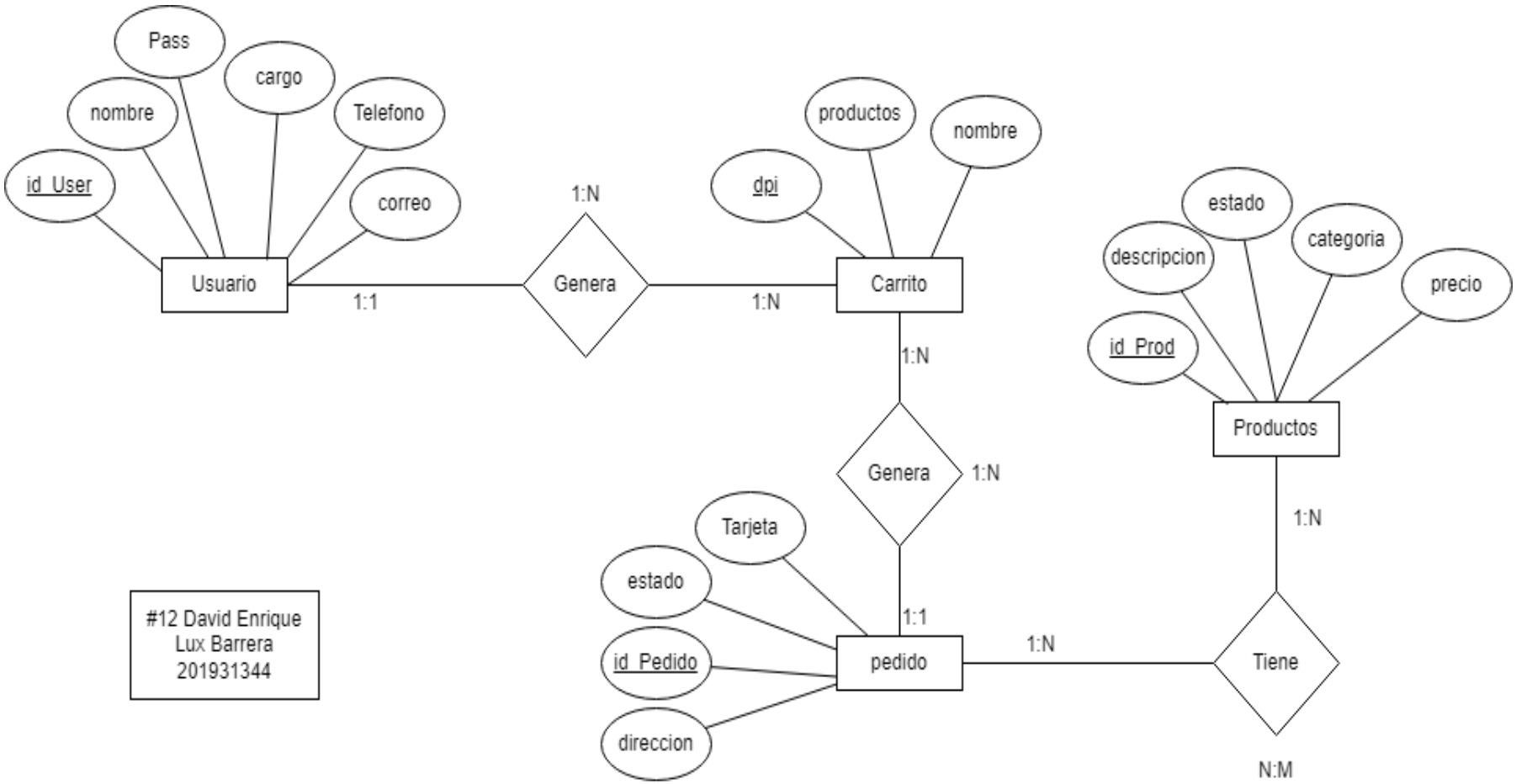
Este es el formato utilizado para poder establecer la forma de almacenamiento de los datos, llamado modelos

```
const mongosee = require('mongoose');
const Schema = mongosee.Schema;
const model = mongosee.model;

const productoSchema = new Schema({
  _id: Schema.Types.ObjectId,
  nombre: String,
  cantidad: Number,
  precio: Number
});

const carritoSchema = new Schema({
  _id: Schema.Types.ObjectId,
  nombre_usuario: String,
  dpi: Number,
  productos_agregados: [productoSchema]
});

module.exports = model('carrito', carritoSchema)
```

Y Aquí podemos observar el formato de los controladores utilizados para poder acceder a la información según sea nuestra necesidad

```
const multer = require('multer');
const mongoose = require('mongoose');
const Producto = require('../models/Productos');
const Carrito = require('../models/carrito');

const agregarProducto = async (req, res) => {
  const { dpi, nombre_producto, cantidad } = req.body;
  try {
    let carrito = await Carrito.findOne({ dpi });
    if (!carrito) {
      carrito = new Carrito({ dpi, productos_agregados: [] });
    }
    const producto = carrito.productos_agregados.find(
      (p) => p.nombre_producto === nombre_producto
    );
    if (producto) {
      producto.cantidad += cantidad;
    } else {
      carrito.productos_agregados.push({ nombre_producto, cantidad });
    }
    await carrito.save();
    res.json({ mensaje: 'Producto agregado al carrito correctamente' });
  } catch (error) {
    console.log(error);
    res.status(500).json({ mensaje: 'Error al agregar el producto al carrito' });
  }
};
```

Diagrama de Cheen:


Pass
cargo
nombre
Telefono
id_User
correo
Usuario
1:1
1:N
Genera
1:N
productos
nombre
dpi
Carrito
1:N
estado
categoria
descripcion
precio
id_Prod
Productos
Genera
1:N
1:N
Tarjeta
estado
1:1
id_Pedido
pedido
1:N
Tiene
direccion
N:M
#12 David Enrique
Lux Barrera
201931344