Manual técnico del Sistema de Ventas Online

Estructura General del Proyecto

Este proyecto se divide en Backend (Java con Spring Boot) y Frontend (React), que se comunicarán mediante una API REST. A continuación, se detalla cada parte y algunos puntos claves del código.

Backend (Java 17 y Spring Boot)

1. Configuración de Proyecto con Spring Boot

Utiliza Spring Initializr para crear el proyecto. Incluye las dependencias: Spring Web, Spring Data MongoDB y Spring Boot DevTools.

2. Estructura de Carpetas:

- **com.example.ventas**

- **controller** - Controladores de cada entidad.

- **service** - Lógica de negocio y servicios.

- **repository** - Repositorios para interactuar con MongoDB.

- **model** - Clases que representan las entidades (Usuario, Artículo, Pedido, etc.).

3. Implementación de Entidades

Define cada entidad como una clase en el paquete model.

Ejemplo de Entidades

Usuario.java

```java
@Data
@Document(collection = "usuarios")
public class Usuario {

    @Id
    private String id;

    @NotNull(message = "El nombre no puede ser nulo")
    private String nombre;

    @Email(message = "El correo debe ser válido")
    private String email;

    @NotNull(message = "El rol no puede ser nulo")
    private String rol;

    @NotNull(message = "La contraseña no puede ser nula")
    private String password;
}
```

Artículo.java

```java
@Data
@Document(collection = "articulos")
public class Articulo {

    @Id
    private String id;

    @NotNull(message = "El nombre no puede ser nulo")
    private String nombre;

    private String descripcion;

    private String categoria;

    @NotNull(message = "El precio no puede ser nulo")
    private Double precio;

    @NotNull(message = "El stock no puede ser nulo")
    private Integer stock;

    private String urlFoto;

    private String proveedorNombre;
}
```

Implementación de Repositorios y Controladores

- Crea interfaces en el paquete repository extendiendo MongoRepository para cada entidad.

- **UsuarioController**

```java
@RestController
@RequestMapping("/api/usuarios")
@CrossOrigin(origins = "http://localhost:3000")
public class UsuarioController {

    @Autowired
    private UsuarioService usuarioService;

    Tabnine | Edit | Test | Explain | Document | Ask
    @GetMapping
    public List<Usuario> obtenerUsuarios() {
        return usuarioService.obtenerTodosLosUsuarios();
    }

    Tabnine | Edit | Test | Explain | Document | Ask
    @GetMapping("/{id}")
    public ResponseEntity<Usuario> obtenerUsuarioPorId(@PathVariable String id) {
        Optional<Usuario> usuario = usuarioService.obtenerUsuarioPorId(id);
        return usuario.map(ResponseEntity::ok)
                .orElseGet(() -> ResponseEntity.status(HttpStatus.NOT_FOUND).build());
    }

    Tabnine | Edit | Test | Explain | Document | Ask
    @PostMapping
    public ResponseEntity<Usuario> crearUsuario(@Validated @RequestBody Usuario usuario) {
        Usuario nuevoUsuario = usuarioService.crearUsuario(usuario);
        return ResponseEntity.status(HttpStatus.CREATED).body(nuevoUsuario);
    }
```

ArticuloController:

```java
@RestController
@RequestMapping("/api/articulos")
@CrossOrigin(origins = "http://localhost:3000")
public class ArticuloController {

    @Autowired
    private ArticuloService articuloService;

    @Autowired
    private ProveedorService proveedorService;
    Tabnine | Edit | Test | Explain | Document | Ask
    @GetMapping("/producto/{id}")
    public ResponseEntity<Producto> obtenerProductoPorId(@PathVariable String id) {
        return articuloService.obtenerProductoPorId(id)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }

    Tabnine | Edit | Test | Explain | Document | Ask
    @GetMapping
    public List<Articulo> obtenerArticulos() {
        return articuloService.obtenerTodosLosArticulos();
    }

    Tabnine | Edit | Test | Explain | Document | Ask
    @GetMapping("/{id}")
    public ResponseEntity<Articulo> obtenerArticuloPorId(@PathVariable String id) {
        return articuloService.obtenerArticuloPorId(id)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }
```

Ejemplo de Service - Lógica de negocio y servicios

UsuarioService

```java
@Service
public class UsuarioService {

    @Autowired
    private UsuarioRepository usuarioRepository;

    Tabnine | Edit | Test | Explain | Document | Ask
    public List<Usuario> obtenerTodosLosUsuarios() {
        return usuarioRepository.findAll();
    }

    Tabnine | Edit | Test | Explain | Document | Ask
    public Optional<Usuario> obtenerUsuarioPorId(String id) {
        return usuarioRepository.findById(id);
    }

    Tabnine | Edit | Test | Explain | Document | Ask
    public Usuario crearUsuario(Usuario usuario) {
        return usuarioRepository.save(usuario);
    }

    Tabnine | Edit | Test | Explain | Document | Ask
    public void eliminarUsuario(String id) {
        usuarioRepository.deleteById(id);
    }

    Tabnine | Edit | Test | Explain | Document | Ask
    public Optional<Usuario> obtenerUsuarioPorEmail(String email) {
        List<Usuario> usuarios = usuarioRepository.findByEmail(email);
        return usuarios.isEmpty() ? Optional.empty() : Optional.of(usuarios.get(index:0));
    }
}
```

ArticuloService

```java
@Service
public class ArticuloService {

    @Autowired
    private ArticuloRepository articuloRepository;

    @Autowired
    private ProveedorRepository proveedorRepository;

    Tabnine | Edit | Test | Explain | Document | Ask
    public Optional<Producto> obtenerProductoPorId(String productoId) {

        for (Proveedor proveedor : proveedorRepository.findAll()) {

            for (Producto producto : proveedor.getProductos()) {
                if (producto.getId().equals(productoId)) {
                    return Optional.of(producto);
                }
            }
        }
        return Optional.empty();

    Tabnine | Edit | Test | Explain | Document | Ask
    public List<Articulo> obtenerTodosLosArticulos() {
        return articuloRepository.findAll();
    }

    Tabnine | Edit | Test | Explain | Document | Ask
    public Optional<Articulo> obtenerArticuloPorId(String id) {
        return articuloRepository.findById(id);
    }
```

Frontend (React)

1. Configuración del Proyecto

Inicialización del proyecto con React, instalación de dependencias como axios y react-router-dom.

2. Estructura del Proyecto React

- src/

- components/ - Componentes de la aplicación (Articulo, Empresa, Proveedores).

- App.js - Configuración de rutas de la aplicación.

3. Ejemplo de Componentes

ArticuloList.js:

```
const ArticuloList = () => {
    const [articulos, setArticulos] = useState([]);
    const [articuloEditado, setArticuloEditado] = useState(null);
    const [idBusqueda, setIdBusqueda] = useState('');
    const [mensaje, setMensaje] = useState('');
    const [articuloSeleccionado, setArticuloSeleccionado] = useState(null);

    useEffect(() => {
        obtenerArticulos();
    }, []);

    const obtenerArticulos = async () => {
        const response = await axios.get('http://localhost:8080/api/articulos');
        setArticulos(response.data);
    };

    const handleEdit = (articulo) => {
        setArticuloEditado(articulo);
    };

    const handleDelete = async (id) => {
        await axios.delete(`http://localhost:8080/api/articulos/${id}`);
        obtenerArticulos();
    };

    const handleArticuloGuardado = () => {
        setArticuloEditado(null);
        obtenerArticulos();
    };

    const buscarArticuloPorId = async () => {
        if (!idBusqueda) {
            setMensaje("Por favor, ingresa un ID para buscar.");
```

Login,js

```
const Login = ({ onLogin }) => {
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [error, setError] = useState('');
    const [isRegistering, setIsRegistering] = useState(false);
    const [role, setRole] = useState('cliente');
    const [adminPassword, setAdminPassword] = useState('');
    const [nombre, setNombre] = useState('');

    const handleSubmit = async (e) => {
        e.preventDefault();
        setError('');

        if (isRegistering) {
            await handleRegister();
        } else {
            await handleLogin();
        }
    };

    const handleLogin = async () => {
        try {
            const response = await fetch('http://localhost:8080/api/login', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify({ email, password }),
            });
```

Proveedores.js

```jsx
import React from 'react';

const Proveedor = ({ proveedor, onEdit, onDelete, onDeleteProducto }) => {
    return (
        <div className="proveedor">
            <h3>{proveedor.nombre}</h3>
            <h4>Productos:</h4>
            <table className="tabla-productos">
                <thead>
                    <tr>
                        <th>Producto</th>
                        <th>Categoría</th>
                        <th>Imagen</th>
                        <th>Precio (Q)</th>
                        <th>Acciones</th> {/* Nueva columna para acciones */}
                    </tr>
                </thead>
                <tbody>
                    {proveedor.productos && proveedor.productos.length > 0 ? (
                        proveedor.productos.map((producto, index) => (
                            <tr key={index}>
                                <td>{producto.nombre}</td>
                                <td>{producto.categoria}</td>
                                <td><img src={producto.urlFoto} alt={producto.nombre} /></td>
                                <td>Q {parseFloat(producto.precio).toFixed(2)}</td>
                                <td>
                                    <button onClick={() => onDeleteProducto(proveedor.id, producto.id)}>Eliminar Producto</
                                </td>
                            </tr>
                        ))
                    ) : (
```

Package.js

```
import Carrito from './components/Carrito/Carrito';
import ProveedorList from './components/Proveedores/ProveedorList';
import EmpresaForm from './components/Empresa/EmpresaForm';
import ArticuloList from './components/Articulos/ArticuloList';
import './App.css';

const App = () => {
    const [isLoggedIn, setIsLoggedIn] = useState(false);
    const [nombreUsuario, setNombreUsuario] = useState('');
    const [showProveedores, setShowProveedores] = useState(false);
    const [showEmpresas, setShowEmpresas] = useState(false);
    const [showCarrito, setShowCarrito] = useState(false);
    const [showArticulos, setShowArticulos] = useState(false);
    const [carrito, setCarrito] = useState([]);

    const handleLogin = (nombre) => {
        setIsLoggedIn(true);
        setNombreUsuario(nombre);
    };

    const handleLogout = () => {
        setIsLoggedIn(false);
        setNombreUsuario('');
        // Reiniciar todas las vistas
        setShowProveedores(false);
        setShowEmpresas(false);
        setShowCarrito(false);
        setShowArticulos(false);
        setCarrito([]);
    };

    const toggleView = (view) => {
        setShowProveedores(view === 'proveedores');
```

Despliegue y Pruebas

1. **Base de Datos:** Conexión a base de datos en MongoDB atlas administrado con MongoDB Compass.

2. **Backend:** Inicio del backend con ***mvn spring-boot:run.***

3. **Frontend:** Inicio del frontend con ***npm start.***

4. **Pruebas Locales:** Verifica que las funcionalidades básicas (registro, inicio de sesión, añadir artículos, proveedores, empresas, etc.)