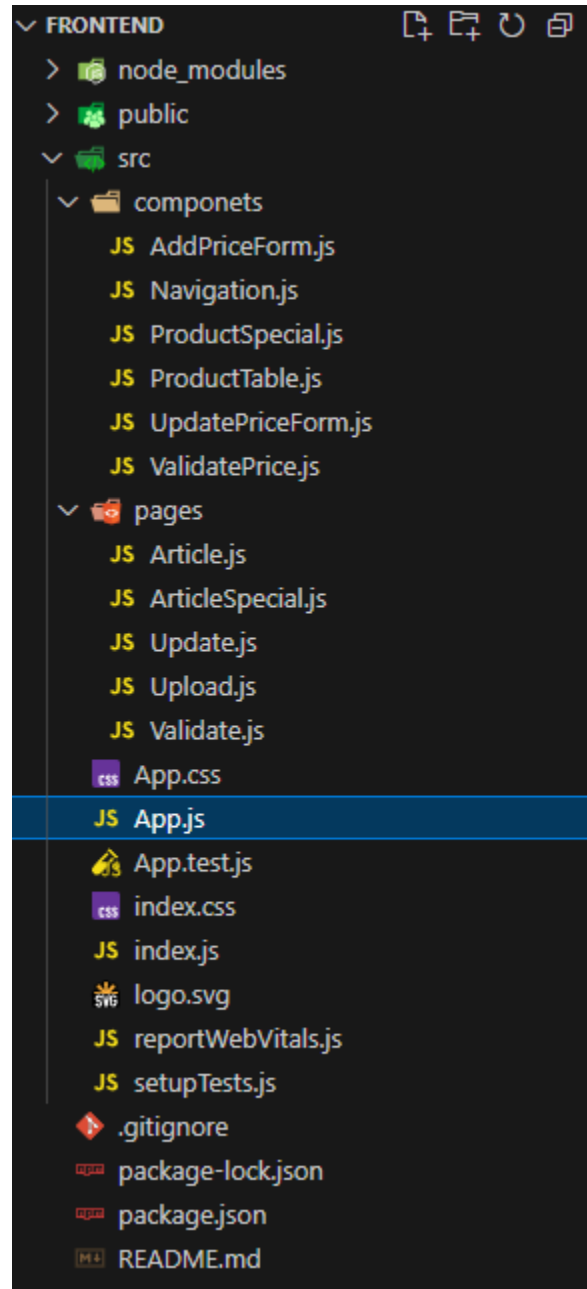


# **Documentacion prueba para vacante desarrollador web**

# Documentación del Frontend

El **frontend** está estructurado en React

Estructura del proyecto



**Carpeta src/ (Código fuente)**

**Subcarpetas:**

1. **components/** → Contiene componentes reutilizables.

- AddPriceForm.js → Formulario para agregar precios.
- Navigation.js → Barra de navegación.
- ProductSpecial.js → Componente relacionado con productos especiales.
- ProductTable.js → Tabla de productos.
- UpdatePriceForm.js → Formulario para actualizar precios.
- ValidatePrice.js → Validación de precios.

## 2. **pages/** → Páginas principales del sitio.

- Article.js y ArticleSpecial.js → Posiblemente muestran artículos.
- Update.js → Página de actualización.
- Upload.js → Página de subida de archivos.
- Validate.js → Página para validación.

### Otros archivos importantes:

- **App.js** → Componente principal donde se gestionan rutas y estructura general.
- **App.css** → Estilos globales.
- **index.js** → Punto de entrada donde se renderiza App.js.
- **package.json** → Contiene dependencias del proyecto.

## Documentación del Backend

Este backend fue desarrollado sobre NodeJS con Express y sigue una arquitectura basada en controladores, modelos y rutas. Se encarga de gestionar productos y precios especiales en una base de datos la cual e encuentra en mongoDB

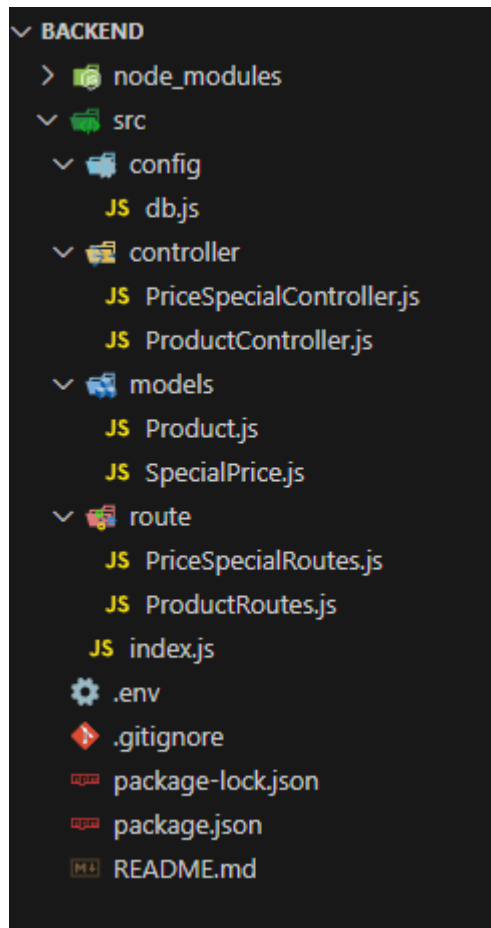


Imagen de la estructura del código

## Controladores

Ubicados en `src/controller/`, estos archivos manejan la lógica de negocio de la aplicación.

- `PriceSpecialController.js`: Controlador para la gestión de precios especiales.
- `ProductController.js`: Controlador para la gestión de productos.

## Modelos

Ubicados en `src/models/`, representan las entidades de la base de datos.

- `Product.js`: Modelo para los productos.
- `SpecialPrice.js`: Modelo para los precios especiales.

## Rutas

Ubicadas en `src/route/`, definen las rutas de la API.

- PriceSpecialRoutes.js: Rutas para manejar precios especiales.
- ProductRoutes.js: Rutas para manejar productos.
- index.js: Archivo que centraliza todas las rutas.

## Conexión a MongoDB

El archivo db.js configura la conexión a MongoDB utilizando Mongoose.

```
JS db.js  X
src > config > JS db.js > ...
1  const mongoose = require("mongoose");
2  require("dotenv").config();
3
4  const connectDB = async () => {
5    try {
6      await mongoose.connect(process.env.MONGO_URI, {
7        useNewUrlParser: true,
8        useUnifiedTopology: true,
9      });
10     console.log("Conectado a MongoDB");
11   } catch (error) {
12     console.error("Error en la conexión:", error);
13     process.exit(1);
14   }
15 };
16
17 module.exports = connectDB;
18
19
```

- **mongoose.connect(process.env.MONGO\_URI, options):** Establece la conexión con la base de datos MongoDB.
- **useNewUrlParser** y **useUnifiedTopology:** Son opciones para evitar advertencias en versiones recientes de Mongoose.
- **console.log("Conectado a MongoDB"):** Mensaje de confirmación en caso de éxito.
- **console.error("Error en la conexión:", error):** Captura y muestra errores en la conexión.
- **process.exit(1):** Finaliza el proceso si ocurre un error.

## PriceSpecialController.js

Este controlador maneja las operaciones CRUD relacionadas con los precios especiales.

```
JS PriceSpecialController.js X
src > controller > JS PriceSpecialController.js > agregarPrecioEspecial > agregarPrecioEspecial
1  const PrecioEspecial = require("../models/SpecialPrice");
2
3  exports.obtenerPreciosEspeciales = async (req, res) => {
4    try {
5      const precios = await PrecioEspecial.find();
6      res.json(precios);
7    } catch (error) {
8      res.status(500).json({ error: "Error al obtener precios especiales" });
9    }
10  };
11
12  exports.agregarPrecioEspecial = async (req, res) => {
13    try {
14      const nuevoPrecio = new PrecioEspecial(req.body);
15      await nuevoPrecio.save();
16      res.json({ mensaje: "Precio especial agregado" });
17    } catch (error) {
18      console.error("Error al agregar precio especial:", error);
19      res.status(500).json({ error: error.message || "Error al agregar precio especial" });
20    }
21  };
22
23  exports.actualizarPrecioEspecial = async (req, res) => {
24    try {
25      const { id } = req.params;
26
27      const precioEspecialActualizado = await PrecioEspecial.findOneAndUpdate(
28        { id_producto: id },
29        req.body,
30        { new: true }
31      );
32
33      if (!precioEspecialActualizado) {
34        return res.status(404).json({ error: "Precio especial no encontrado para este producto" });
35      }
36
37      res.json({
38        mensaje: "Precio especial actualizado",
39        data: precioEspecialActualizado,
40      });
41    } catch (error) {
42      res.status(500).json({ error: "Error al actualizar precio especial" });
43    }
44  };
45
46  exports.validarUsuarioEnPrecios = async (req, res) => {
47    try {
48      const { id_usuario, id_producto } = req.body;
49      const precioEspecial = await PrecioEspecial.findOne({ id_usuario, id_producto });
50      res.json({ precioEspecial });
51    } catch (error) {
52      res.status(500).json({ error: "Error en la validación" });
53    }
54  };
55
```

- Importación de modelo: `const PrecioEspecial = require("../models/SpecialPrice");`
- Funciones del controlador:
  1. Obtener precios especiales: `exports.obtenerPreciosEspeciales = async (req, res) => { ... }`
  2. Agregar un nuevo precio especial: `exports.agregarPrecioEspecial = async (req, res) => { ... }`
  3. Actualizar un precio especial: `exports.actualizarPrecioEspecial = async (req, res) => { ... }`

4. Validar si un usuario tiene un precio especial asignado:  
exports.validarUsuarioEnPrecios = async (req, res) => { ... }

ProductController.js

```
JS ProductController.js X
src > controller > JS ProductController.js > ...
1  const Producto = require("../models/Product");
2
3  exports.obtenerProductos = async (req, res) => {
4    try {
5      const productos = await Producto.find();
6      res.json(productos);
7    } catch (error) {
8      res.status(500).json({ error: "Error al obtener productos" });
9    }
10 };
11
12
```

Este controlador maneja la obtención de productos desde la base de datos.

Importación de modelo : const Producto = require("../models/Product");

Funciones del controlador

Obtener productos: exports.obtenerProductos = async (req, res) => { ... }

Product.js

Este archivo define el modelo de datos para los productos en la base de datos utilizando Mongoose.

```
JS Product.js X
src > models > JS Product.js > ...
1
2  const mongoose = require("mongoose");
3
4  const productoSchema = new mongoose.Schema({
5    nombre: String,
6    descripcion: String,
7    precio: Number,
8    stock: Number,
9  });
10
11  module.exports = mongoose.model("Producto", productoSchema);
12
13
14
```

Definición del esquema

**Descripción:** Se crea un esquema (Schema) que define la estructura de un producto.

**Campos del esquema:**

- nombre: **String** → Nombre del producto.
- descripcion: **String** → Breve descripción del producto.
- precio: **Number** → Precio del producto.
- stock: **Number** → Cantidad disponible en inventario.

Exportación del modelo

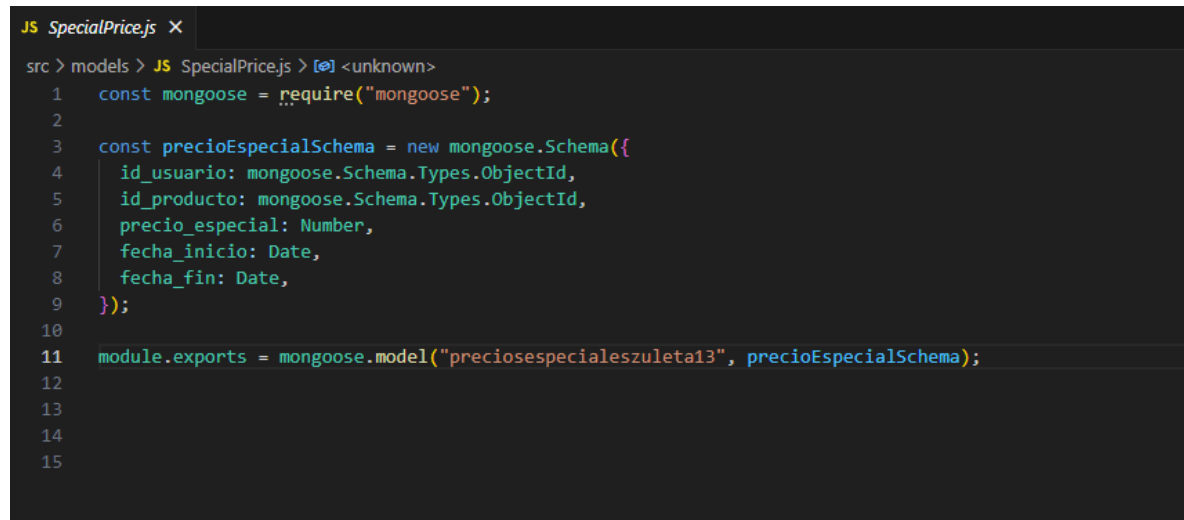
```
module.exports = mongoose.model("Producto", productoSchema);
```

**Descripción:** Se crea y exporta el modelo Producto basado en productoSchema, permitiendo su uso en otros archivos.



## SpecialPrice.js

Este archivo define el modelo de datos para los precios especiales en la base de datos utilizando Mongoose.



```
JS SpecialPrice.js X
src > models > JS SpecialPrice.js > [?] <unknown>
1  const mongoose = require("mongoose");
2
3  const precioEspecialSchema = new mongoose.Schema({
4    id_usuario: mongoose.Schema.Types.ObjectId,
5    id_producto: mongoose.Schema.Types.ObjectId,
6    precio_especial: Number,
7    fecha_inicio: Date,
8    fecha_fin: Date,
9  });
10
11 module.exports = mongoose.model("preciosespecialeszuleta13", precioEspecialSchema);
12
13
14
15
```

### Definición del esquema

**Descripción:** Se crea un esquema (Schema) que define la estructura de un precio especial asociado a un usuario y un producto.

#### Campos del esquema:

- `id_usuario`: **ObjectId** → Identificador del usuario que recibe el precio especial.
- `id_producto`: **ObjectId** → Identificador del producto al que se aplica el precio especial.
- `precio_especial`: **Number** → Precio especial asignado al producto.
- `fecha_inicio`: **Date** → Fecha en que comienza la oferta especial.
- `fecha_fin`: **Date** → Fecha en que finaliza la oferta especial.

### Exportación del modelo

**Descripción:** Se crea y exporta el modelo `preciosespecialeszuleta13` basado en `preciosEspeciales<tuPrimerApellido><dosNúmeros>`.

## PriceSpecialRoutes.js

Este archivo define las rutas relacionadas con la gestión de precios especiales en una API utilizando **Express.js**.

```
JS PriceSpecialRoutes.js X
src > route > JS PriceSpecialRoutes.js > ...
1  const express = require("express");
2  const router = express.Router();
3  const {
4    obtenerPreciosEspeciales,
5    agregarPrecioEspecial,
6    actualizarPrecioEspecial,
7    validarUsuarioEnPrecios,
8  } = require("../controller/PriceSpecialController");
9
10 router.get("/", obtenerPreciosEspeciales);
11 router.post("/", agregarPrecioEspecial);
12 router.put("/:id", actualizarPrecioEspecial);
13 router.post("/validar", validarUsuarioEnPrecios);
14
15 module.exports = router;
16
```

Se importa **Express** y se usa el método Router() para crear un objeto router, que se encargará de manejar las rutas de la API.

Se importan las funciones del **controlador** PriceSpecialController.js, que maneja la lógica de negocio de los precios especiales.

Se definen las rutas get, post, put

## ProductRoutes.js

Este archivo define las rutas para la gestión de productos en una API utilizando **Express.js**.

```
JS ProductRoutes.js X
src > route > JS ProductRoutes.js > ...
1  const express = require("express");
2  const router = express.Router();
3  const { obtenerProductos } = require("../controller/productcontroller");
4
5  router.get("/", obtenerProductos);
6
7  module.exports = router;
8
9
```

Se importa **Express** y se usa `express.Router()` para definir el router que manejará las rutas de productos.

Se importa la función `obtenerProductos` desde el controlador `productcontroller.js`, que contendrá la lógica para obtener los productos.