Mapeo de Requisitos al Código

A continuación, se muestra cada punto de la guía original y dónde está implementado en tu proyecto:

Guía (Sección)	Implementación	Archivo / Ubicación
1.1 Validación de datos	Endpoint y UI de validación	`backend/index.js` (Joi schema) `frontend/src/components/UserForm.jsx`
1.2 Manipulación del DOM (Productos)	Tabla de productos y sumatoria	`frontend/src/components/ProductTable.jsx` `frontend/src/index.css` (estilos `.product-table`)
1.3 CRUD Básico	Endpoints protegidos con JWT / UI CRUD	`backend/index.js` (rutas `/api/tasks`) `frontend/src/components/TaskManager.jsx`
2.1 Autenticación con JWT	Login, middleware, JWT issuance	`backend/index.js` (`/api/login`, `authenticate`) `frontend/src/components/LoginForm.jsx`
2.2 Optimización Frontend	Hook centralizado de peticiones	`frontend/src/hooks/useApi.js` Usado en `LoginForm.jsx` y `TaskManager.jsx`
2.3 Componentización	Componentes reutilizables	Formularios: `UserForm.jsx`, `LoginForm.jsx` ProductTable, TaskManager
3.1 Diseño de Base de Datos (Bonus)	Scripts DDL PostgreSQL	Archivo SQL propuesto en sección de BD

---

1. Lógica y Codificación

1.1 Validación de Datos

Requisito: Validar `name`, `email`, `age` con reglas:

- `name`: string no vacío
- `email`: formato email
- `age`: integer ≥ 18

Implementación:

- **Backend**: `backend/index.js` con Joi schema en `/api/validate-user`.
- **Frontend:** `UserForm.jsx` en `frontend/src/components/UserForm.jsx`:

1.2 Listado de Productos y Sumatoria

Requisito: Mostrar un listado de productos con su nombre y precio, y calcular el total.

Implementación:

- **Componente**
  React: `ProductTable.jsx` en `frontend/src/components/ProductTable.jsx`
  - Renderiza productos recibidos por `props` (nombre y precio).
  - Calcula la sumatoria usando `reduce()`.
  - Estructura HTML con `<table>` y filas por cada producto.
- **Estilos**: `frontend/src/index.css`
  - Clase `.product-table` define bordes, márgenes y estilo visual de la tabla.

Demostración:

- Tras hacer login, se muestra automáticamente `ProductTable` junto a `TaskManager`.
- Los datos provienen de un array local `sampleProducts` para efectos de prueba.
- Se muestra una fila adicional con el total sumado de todos los precios.

Notas Técnicas:

- El componente `ProductTable` es totalmente reutilizable.
- Se puede adaptar fácilmente a una fuente de datos dinámica (por ejemplo, API REST).

2.1 Autenticación con JWT

- **Login**: `POST /api/login` en `backend/index.js`
- **Middleware**: función `authenticate(req, res, next)` protege rutas.
- **UI**: `frontend/src/components/LoginForm.jsx` usa `useApi` para el login.

2.2 Optimización Frontend

- **Hook**: `frontend/src/hooks/useApi.js`
- **Uso**: en `LoginForm.jsx` y `TaskManager.jsx` para `request` y `authRequest`.

2.3 Componentización

- **Formularios**: `UserForm.jsx`, `LoginForm.jsx`.
- **Tablas**: `ProductTable.jsx`, `TaskManager.jsx`.

---

3. Bonus: Diseño de Base de Datos

DDL PostgreSQL:

```
-- users
table users (
  id SERIAL PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  created_at TIMESTAMP DEFAULT NOW()
);
-- tasks
CREATE TABLE tasks (
  id SERIAL PRIMARY KEY,
  title VARCHAR(255) NOT NULL,
  description TEXT,
  status VARCHAR(50) DEFAULT 'pendiente',
  user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
  created_at TIMESTAMP DEFAULT NOW()
);
-- products
CREATE TABLE products (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  price DECIMAL(10,2) NOT NULL CHECK (price >= 0),
  created_at TIMESTAMP DEFAULT NOW()
);
```

Respuestas a los Puntos del Documento Original

Algunos enunciados del examen estaban formulados como preguntas. A continuación se responden directamente:

1. **¿Cómo validar `name`, `email` y `age`?**
   - Se usa Joi en el backend dentro de `backend/index.js` con reglas `.required()`, `.email()`, `.min(18)`.
   - El formulario `UserForm.jsx` muestra los errores devueltos (status 422)

1.2. ¿Cómo mostrar productos y sumar precios?

- Componente `ProductTable.jsx` renderiza una tabla con productos pasados vía props y calcula el total con `reduce`.
- Los estilos en `index.css` definen bordes y formato de la tabla.

A continuación, se listan los puntos de la prueba con una breve explicación de cómo fueron implementados o respondidos en la solución:

1. **Validación de usuario (name, email, age)**
   - Se implementó una ruta `/api/validate-user` en el backend que usa Joi para validar los datos.
   - En el frontend, `UserForm.jsx` gestiona el formulario y muestra los errores devueltos por el backend.
2. **Login y autenticación con JWT**
   - Ruta `/api/login` en el backend que genera un token JWT.
   - Se implementó `authenticate()` como middleware para proteger las rutas.
   - En el frontend, el formulario de login (`LoginForm.jsx`) recibe el token y lo guarda en estado.
3. **Acceso a recursos protegidos y control del token**
   - El CRUD de tareas (`/api/tasks`) está protegido por JWT.
   - En el frontend se prueba el acceso con token válido, y se implementó control para expiración/token inválido.
4. **Listado de productos y sumatoria de precios**
   - `ProductTable.jsx` renderiza productos con nombre y precio, y muestra la suma total.
   - Incluye estilos en `index.css`.
5. **Componentización y reutilización**
   - Se construyeron componentes reutilizables como `UserForm`, `LoginForm`, `TaskManager`, y `ProductTable`.
6. **Optimización del frontend**
   - Se implementó el hook `useApi.js` para centralizar el manejo de peticiones, errores y autenticación.
7. **Diseño de base de datos (Bonus)**
   - Se entregó un script SQL con las estructuras `users`, `tasks` y `products`, incluyendo llaves primarias y foráneas, y validaciones.

Cada punto se vincula a una sección funcional del proyecto. Los detalles técnicos y su ubicación exacta están descritos en el mapeo de requisitos al código en la parte superior del documento.

Instrucciones de Instalación y Uso

1. Sube el proyecto a **GitHub**.
2. Clona el repositorio:
3. `git clone https://github.com/Bryandrm/fullstack-project.gitcd proyecto-fullstack`
4. Instala dependencias en la raíz (con script `concurrently` configurado):
5. `npm install`
6. Levanta la aplicación (frontend + backend):
7. `npm run dev`
8. Abre en el navegador `http://localhost:3000` y sigue el flujo.