# **Hoja de Ruta Detallada - Fase 2: Gestión y Descubribilidad**

Esta fase se enfoca en permitir a los emprendedores gestionar sus negocios y ofertas (productos/servicios), y hacer que estas sean públicamente descubribles y accesibles para los clientes.

## **Índice de Características Fundamentales a Implementar:**

* **Capítulo 1: Gestión de Negocios (CRUD para Emprendedores)**
  + Creación, lectura (listado propio y detalle), actualización y eliminación de negocios.
  + Soporte para múltiples URLs de fotos de negocio.
* **Capítulo 2: Gestión de Productos/Servicios (CRUD para Emprendedores)**
  + Creación, lectura (listado propio y detalle), actualización y eliminación de productos/servicios.
  + Asociación de productos/servicios a un negocio o directamente al usuario.
  + Soporte para múltiples URLs de fotos de producto/servicio.
* **Capítulo 3: Exploración Pública de Negocios y Productos (Marketplace Básico)**
  + Nueva página de inicio con listado público de negocios y productos/servicios.
  + Funcionalidad de búsqueda y filtros básicos para el listado público.
  + Páginas de detalle públicas para negocios y productos/servicios.
  + Botón "Contactar/Solicitar Encargo" en las vistas públicas (preparación para Fase 4).

## **Capítulo 1: Gestión de Negocios (CRUD para Emprendedores)**

### **Descripción General**

Este capítulo implementa la capacidad para que un usuario autenticado (con tipo\_tier de "microemprendimiento" o "freelancer") pueda crear, ver, actualizar y eliminar sus propios negocios. Se permitirá la carga de múltiples URLs de imágenes para cada negocio.

### **Pseudocódigo**

#### **Backend:**

# CRUD (backend/app/crud/business.py)  
function create\_business(db, user\_id, business\_data):  
 create new Business object with user\_id and business\_data  
 save to db  
 return new Business  
  
function get\_business\_by\_id(db, business\_id):  
 query Business by id  
 return Business or None  
  
function get\_businesses\_by\_user\_id(db, user\_id):  
 query Business by user\_id  
 return list of Businesses  
  
function update\_business(db, business\_id, business\_update\_data):  
 get Business by id  
 if Business not found or not owned by user:  
 raise error  
 update Business fields with business\_update\_data  
 save to db  
 return updated Business  
  
function delete\_business(db, business\_id):  
 get Business by id  
 if Business not found or not owned by user:  
 raise error  
 delete Business from db  
 return success  
  
# Router (backend/app/routers/business\_router.py)  
@router.post("/")  
def create\_business\_endpoint(current\_user, business\_data, db\_session):  
 if current\_user.tipo\_tier is not in ["microemprendimiento", "freelancer"]:  
 raise HTTPException(status\_code=403)  
 call crud.create\_business  
 return new\_business  
  
@router.get("/me")  
def get\_my\_businesses\_endpoint(current\_user, db\_session):  
 call crud.get\_businesses\_by\_user\_id  
 return list\_of\_businesses  
  
@router.get("/{business\_id}")  
def get\_business\_detail\_endpoint(current\_user, business\_id, db\_session):  
 call crud.get\_business\_by\_id  
 if business.user\_id != current\_user.id:  
 raise HTTPException(status\_code=403)  
 return business  
  
@router.put("/{business\_id}")  
def update\_business\_endpoint(current\_user, business\_id, business\_update\_data, db\_session):  
 call crud.update\_business (with ownership check)  
 return updated\_business  
  
@router.delete("/{business\_id}")  
def delete\_business\_endpoint(current\_user, business\_id, db\_session):  
 call crud.delete\_business (with ownership check)  
 return success\_message

#### **Frontend:**

// API Calls (frontend/src/api/businessApi.js)  
function createBusiness(token, businessData):  
 fetch POST /businesses/ with businessData and Auth header  
 return response  
  
function getMyBusinesses(token):  
 fetch GET /businesses/me with Auth header  
 return response  
  
function getBusinessById(token, businessId):  
 fetch GET /businesses/{businessId} with Auth header  
 return response  
  
function updateBusiness(token, businessId, updateData):  
 fetch PUT /businesses/{businessId} with updateData and Auth header  
 return response  
  
function deleteBusiness(token, businessId):  
 fetch DELETE /businesses/{businessId} with Auth header  
 return response  
  
// Screens (frontend/src/screens/)  
// ManageBusinessesScreen:  
function ManageBusinessesScreen():  
 use state for businesses list  
 useEffect to fetch businesses on load  
 map over businesses to display cards/list items  
 button to navigate to CreateBusinessScreen  
 buttons for EditBusinessScreen and Delete for each business  
  
// CreateBusinessScreen:  
function CreateBusinessScreen():  
 use state for form fields (name, description, rubro, location, fotos\_urls)  
 form with Input fields for text, and a way to add multiple URL inputs for fotos\_urls  
 onSubmit: call businessApi.createBusiness  
 redirect to ManageBusinessesScreen on success  
  
// EditBusinessScreen:  
function EditBusinessScreen():  
 get businessId from URL params  
 use state for form fields, initialized with current business data  
 useEffect to fetch business data on load  
 form with pre-filled Input fields  
 onSubmit: call businessApi.updateBusiness  
 redirect to ManageBusinessesScreen on success

### **Documentación**

#### **Modelos (backend/app/models.py)**

* **Negocio**
  + id: UUID, Primary Key
  + usuario\_id: UUID, ForeignKey a usuarios.id
  + nombre: String, Not Null
  + rubro: String, Not Null
  + descripcion: String, Nullable
  + localizacion\_geografica: String, Nullable (placeholder)
  + fotos\_urls: JSONB, Nullable (Almacena List[str])
  + rating\_promedio: Float, Default 0.0
  + reviews\_totales: Integer, Default 0
  + fecha\_creacion: DateTime, Default func.now()
  + fecha\_actualizacion: DateTime, Default func.now(), onupdate=func.now()

#### **Esquemas (backend/app/schemas.py)**

* **NegocioBase(BaseModel)**:
  + nombre: str
  + rubro: str
  + descripcion: Optional[str] = None
  + localizacion\_geografica: Optional[str] = None
  + fotos\_urls: Optional[List[str]] = None
* **NegocioCreate(NegocioBase)**:
  + (Hereda de NegocioBase)
* **NegocioUpdate(NegocioBase)**:
  + (Hereda de NegocioBase, todos los campos opcionales)
* **NegocioResponse(NegocioBase)**:
  + id: UUID
  + usuario\_id: UUID
  + rating\_promedio: float
  + reviews\_totales: int
  + fecha\_creacion: datetime
  + fecha\_actualizacion: datetime
  + model\_config = ConfigDict(from\_attributes=True)

#### **Endpoints API (backend/app/routers/business\_router.py)**

* POST /businesses/
  + **Request Body**: NegocioCreate
  + **Response**: NegocioResponse (201 OK)
  + **Auth**: Requiere JWT (Usuario autenticado)
* GET /businesses/me
  + **Response**: List[NegocioResponse] (200 OK)
  + **Auth**: Requiere JWT
* GET /businesses/{business\_id}
  + **Response**: NegocioResponse (200 OK)
  + **Auth**: Requiere JWT (y ser propietario del negocio)
* PUT /businesses/{business\_id}
  + **Request Body**: NegocioUpdate
  + **Response**: NegocioResponse (200 OK)
  + **Auth**: Requiere JWT (y ser propietario del negocio)
* DELETE /businesses/{business\_id}
  + **Response**: {"message": "Business deleted successfully"} (200 OK)
  + **Auth**: Requiere JWT (y ser propietario del negocio)

### **Backend**

1. **Modificar backend/app/models.py:**
   * Asegúrate de que fotos\_urls en la clase Negocio sea Column(JSONB, nullable=True). Este tipo ya permite almacenar listas en PostgreSQL.
2. **Modificar backend/app/schemas.py:**
   * Define las clases NegocioBase, NegocioCreate, NegocioUpdate, NegocioResponse como se describe en la sección de Documentación. Asegúrate de que fotos\_urls en los esquemas Create y Update sea Optional[List[str]] = None.
3. **Crear backend/app/crud/business.py:**
   * Implementa las funciones create\_business, get\_business\_by\_id, get\_businesses\_by\_user\_id, update\_business, delete\_business utilizando SQLAlchemy ORM. Asegúrate de incluir la lógica para verificar la propiedad del negocio en las funciones de actualización y eliminación.
4. **Crear backend/app/routers/business\_router.py:**
   * Define los endpoints API (POST, GET /me, GET /{id}, PUT, DELETE) como se describe en la sección de Documentación.
   * Utiliza Depends(get\_current\_user) para proteger los endpoints y Depends(get\_db) para la sesión de base de datos.
   * Implementa la lógica para verificar que el usuario autenticado sea el propietario del negocio en los endpoints GET /{id}, PUT, DELETE.
5. **Modificar backend/app/main.py:**
   * Importa el nuevo router: from app.routers import business\_router.
   * Incluye el router en la aplicación FastAPI: app.include\_router(business\_router.router, prefix="/businesses", tags=["Businesses"]).
   * **¡Recuerda reiniciar el servidor FastAPI después de estos cambios!**

### **Frontend**

1. **Modificar frontend/src/App.js:**
   * Añade las rutas protegidas para la gestión de negocios:  
     <Route path="/dashboard/businesses" element={<ManageBusinessesScreen />} />  
     <Route path="/dashboard/businesses/new" element={<CreateBusinessScreen />} />  
     <Route path="/dashboard/businesses/edit/:id" element={<EditBusinessScreen />} />
2. **Crear frontend/src/api/businessApi.js:**
   * Implementa las funciones JavaScript para interactuar con los endpoints del backend de negocios (ej., createBusiness, getMyBusinesses, getBusinessById, updateBusiness, deleteBusiness). Asegúrate de incluir el token JWT en el encabezado Authorization.
3. **Modificar frontend/src/screens/DashboardScreen.js:**
   * En la sección "Acciones Rápidas", añade un Button de Shadcn UI con un Link a /dashboard/businesses y el texto "Gestionar Negocios".
4. **Crear frontend/src/screens/ManageBusinessesScreen.js:**
   * Utiliza useEffect para cargar los negocios del usuario autenticado al montar el componente.
   * Muestra los negocios en una lista o en tarjetas (usando Card de Shadcn UI).
   * Cada negocio debe tener botones "Editar" (con Link a /dashboard/businesses/edit/:id) y "Eliminar".
   * Añade un botón "Crear Nuevo Negocio" (con Link a /dashboard/businesses/new).
   * Maneja los estados de carga y error.
5. **Crear frontend/src/screens/CreateBusinessScreen.js:**
   * Un formulario para crear un nuevo negocio.
   * Utiliza Input de Shadcn UI para nombre, rubro, descripcion, localizacion\_geografica.
   * Para fotos\_urls, implementa una forma de añadir **múltiples URLs**. Puedes usar un Input por cada URL que el usuario quiera añadir (con un botón "Añadir URL") o un textarea donde el usuario pegue URLs separadas por comas o saltos de línea (luego procesar en el onSubmit).
   * Al enviar, llama a businessApi.createBusiness y redirige a ManageBusinessesScreen al éxito.
6. **Crear frontend/src/screens/EditBusinessScreen.js:**
   * Similar a CreateBusinessScreen, pero el formulario debe precargarse con los datos del negocio (obteniendo el id de la URL y haciendo una llamada a businessApi.getBusinessById).
   * Al enviar, llama a businessApi.updateBusiness.

### **Prueba de UX (Experiencia de Usuario)**

1. **Registro/Login:** Inicia sesión como un usuario "Microemprendimiento" o "Freelancer".
2. **Navegación:** Desde el Dashboard, haz clic en "Gestionar Negocios".
3. **Creación:** Haz clic en "Crear Nuevo Negocio". Rellena los campos, añade 2-3 URLs de imágenes válidas (ej. de Placehold.co). Guarda.
   * **Verificación:** ¿El negocio aparece en la lista? ¿Los datos son correctos?
4. **Edición:** Haz clic en "Editar" un negocio existente. Cambia algunos campos, añade o quita una URL de imagen. Guarda.
   * **Verificación:** ¿Los cambios se reflejan en la lista?
5. **Eliminación:** Haz clic en "Eliminar" un negocio. Confirma.
   * **Verificación:** ¿El negocio desaparece de la lista?
6. **Validación:** Intenta crear un negocio con campos vacíos requeridos o datos inválidos.
   * **Verificación:** ¿El frontend muestra un mensaje de error amigable?

### **Debugging**

* **500 Internal Server Error en el backend al crear/actualizar:**
  + **Causa:** Problemas con los tipos de datos en el modelo/esquema, o la lógica CRUD.
  + **Solución:** Revisa el traceback en la terminal del backend. Asegúrate de que fotos\_urls se maneje como List[str] en Pydantic y como JSONB en SQLAlchemy.
* **422 Unprocessable Entity en el frontend al enviar el formulario:**
  + **Causa:** Los datos del formulario no coinciden con el esquema Pydantic esperado por el backend.
  + **Solución:** Revisa la pestaña "Red" (Network) en el navegador, en la solicitud POST o PUT. Observa el "Payload" enviado y la "Respuesta" del servidor para ver qué campo específico está causando el error de validación.
* **Negocios no aparecen en la lista o no se actualizan:**
  + **Causa:** Problemas en las llamadas a la API (token JWT, URL incorrecta), o en el estado de React.
  + **Solución:** Usa la pestaña "Red" para verificar que las solicitudes GET se envían correctamente y que el backend devuelve los datos esperados. Revisa la consola para errores de React.

## **Capítulo 2: Gestión de Productos/Servicios (CRUD para Emprendedores)**

### **Descripción General**

Este capítulo implementa la capacidad para que un usuario autenticado pueda crear, ver, actualizar y eliminar sus propios productos o servicios. Los productos pueden asociarse a un negocio existente del usuario o existir de forma independiente (para freelancers). Se permitirá la carga de múltiples URLs de imágenes para cada producto/servicio.

### **Pseudocódigo**

#### **Backend:**

# CRUD (backend/app/crud/product.py)  
function create\_product(db, user\_id, product\_data):  
 if product\_data.negocio\_id:  
 verify business ownership by user\_id  
 create new Product object  
 save to db  
 return new Product  
  
function get\_product\_by\_id(db, product\_id):  
 query Product by id  
 return Product or None  
  
function get\_products\_by\_user\_id(db, user\_id):  
 query Product by user\_id  
 return list of Products  
  
function get\_products\_by\_business\_id(db, business\_id):  
 query Product by business\_id  
 return list of Products  
  
function update\_product(db, product\_id, product\_update\_data):  
 get Product by id  
 if Product not found or not owned by user:  
 raise error  
 update Product fields  
 save to db  
 return updated Product  
  
function delete\_product(db, product\_id):  
 get Product by id  
 if Product not found or not owned by user:  
 raise error  
 delete Product from db  
 return success  
  
# Router (backend/app/routers/product\_router.py)  
@router.post("/")  
def create\_product\_endpoint(current\_user, product\_data, db\_session):  
 # Validate if negocio\_id belongs to current\_user if provided  
 call crud.create\_product  
 return new\_product  
  
@router.get("/me")  
def get\_my\_products\_endpoint(current\_user, db\_session):  
 call crud.get\_products\_by\_user\_id  
 return list\_of\_products  
  
@router.get("/{product\_id}")  
def get\_product\_detail\_endpoint(current\_user, product\_id, db\_session):  
 call crud.get\_product\_by\_id  
 if product.user\_id != current\_user.id:  
 raise HTTPException(status\_code=403)  
 return product  
  
@router.put("/{product\_id}")  
def update\_product\_endpoint(current\_user, product\_id, product\_update\_data, db\_session):  
 call crud.update\_product (with ownership check)  
 return updated\_product  
  
@router.delete("/{product\_id}")  
def delete\_product\_endpoint(current\_user, product\_id, db\_session):  
 call crud.delete\_product (with ownership check)  
 return success\_message

#### **Frontend:**

// API Calls (frontend/src/api/productApi.js)  
function createProduct(token, productData):  
 fetch POST /products/ with productData and Auth header  
 return response  
  
function getMyProducts(token):  
 fetch GET /products/me with Auth header  
 return response  
  
function getProductById(token, productId):  
 fetch GET /products/{productId} with Auth header  
 return response  
  
function updateProduct(token, productId, updateData):  
 fetch PUT /products/{productId} with updateData and Auth header  
 return response  
  
function deleteProduct(token, productId):  
 fetch DELETE /products/{productId} with Auth header  
 return response  
  
// Screens (frontend/src/screens/)  
// ManageProductsScreen:  
function ManageProductsScreen():  
 use state for products list  
 useEffect to fetch products on load  
 map over products to display cards/list items  
 button to navigate to CreateProductScreen  
 buttons for EditProductScreen and Delete for each product  
  
// CreateProductScreen:  
function CreateProductScreen():  
 use state for form fields (name, description, tipo\_producto, fotos\_urls, precio\_sugerido, negocio\_id)  
 useEffect to fetch user's businesses (for dropdown)  
 form with Input fields, Select for tipo\_producto, and multiple URL inputs for fotos\_urls  
 Select for associating with a business (optional)  
 onSubmit: call productApi.createProduct  
 redirect to ManageProductsScreen on success  
  
// EditProductScreen:  
function EditProductScreen():  
 get productId from URL params  
 use state for form fields, initialized with current product data  
 useEffect to fetch product data on load and user's businesses  
 form with pre-filled Input fields  
 onSubmit: call productApi.updateProduct  
 redirect to ManageProductsScreen on success

### **Documentación**

#### **Modelos (backend/app/models.py)**

* **Producto**
  + id: UUID, Primary Key
  + usuario\_id: UUID, ForeignKey a usuarios.id
  + negocio\_id: UUID, ForeignKey a negocios.id, Nullable
  + nombre: String, Not Null
  + descripcion: String, Nullable
  + tipo\_producto: SQLEnum(ProductType), Not Null
  + fotos\_urls: JSONB, Nullable (Almacena List[str])
  + precio\_sugerido: Float, Nullable
  + cogs: Float, Nullable
  + margen\_ganancia\_porcentaje: Float, Nullable
  + stock: Integer, Nullable (solo para PHYSICAL\_GOOD)
  + unidad\_medida: String, Nullable
  + atributos\_especificos: JSONB, Nullable
  + rating\_promedio: Float, Default 0.0
  + reviews\_count: Integer, Default 0
  + fecha\_creacion: DateTime, Default func.now()
  + fecha\_actualizacion: DateTime, Default func.now(), onupdate=func.now()

#### **Esquemas (backend/app/schemas.py)**

* **ProductoBase(BaseModel)**:
  + negocio\_id: Optional[UUID] = None
  + nombre: str
  + descripcion: Optional[str] = None
  + tipo\_producto: ProductType
  + fotos\_urls: Optional[List[str]] = None
  + precio\_sugerido: Optional[float] = None
  + cogs: Optional[float] = None
  + margen\_ganancia\_porcentaje: Optional[float] = None
  + stock: Optional[int] = None
  + unidad\_medida: Optional[str] = None
  + atributos\_especificos: Optional[Dict] = None
* **ProductoCreate(ProductoBase)**:
  + (Hereda de ProductoBase)
* **ProductoUpdate(ProductoBase)**:
  + (Hereda de ProductoBase, todos los campos opcionales)
* **ProductoResponse(ProductoBase)**:
  + id: UUID
  + usuario\_id: UUID
  + rating\_promedio: float
  + reviews\_count: int
  + fecha\_creacion: datetime
  + fecha\_actualizacion: datetime
  + model\_config = ConfigDict(from\_attributes=True)

#### **Endpoints API (backend/app/routers/product\_router.py)**

* POST /products/
  + **Request Body**: ProductoCreate
  + **Response**: ProductoResponse (201 OK)
  + **Auth**: Requiere JWT
* GET /products/me
  + **Response**: List[ProductoResponse] (200 OK)
  + **Auth**: Requiere JWT
* GET /products/{product\_id}
  + **Response**: ProductoResponse (200 OK)
  + **Auth**: Requiere JWT (y ser propietario del producto)
* PUT /products/{product\_id}
  + **Request Body**: ProductoUpdate
  + **Response**: ProductoResponse (200 OK)
  + **Auth**: Requiere JWT (y ser propietario del producto)
* DELETE /products/{product\_id}
  + **Response**: {"message": "Product deleted successfully"} (200 OK)
  + **Auth**: Requiere JWT (y ser propietario del producto)

### **Backend**

1. **Modificar backend/app/models.py:**
   * Asegúrate de que fotos\_urls en la clase Producto sea Column(JSONB, nullable=True).
2. **Modificar backend/app/schemas.py:**
   * Define las clases ProductoBase, ProductoCreate, ProductoUpdate, ProductoResponse como se describe en la sección de Documentación. Asegúrate de que fotos\_urls en los esquemas Create y Update sea Optional[List[str]] = None.
3. **Crear backend/app/crud/product.py:**
   * Implementa las funciones create\_product, get\_product\_by\_id, get\_products\_by\_user\_id, get\_products\_by\_business\_id, update\_product, delete\_product utilizando SQLAlchemy ORM.
   * Asegúrate de incluir la lógica para verificar la propiedad del producto y, si se proporciona negocio\_id, que ese negocio también pertenezca al usuario actual.
4. **Crear backend/app/routers/product\_router.py:**
   * Define los endpoints API (POST, GET /me, GET /{id}, PUT, DELETE) como se describe en la sección de Documentación.
   * Utiliza Depends(get\_current\_user) para proteger los endpoints y Depends(get\_db) para la sesión de base de datos.
   * Implementa la lógica para verificar la propiedad del producto.
5. **Modificar backend/app/main.py:**
   * Importa el nuevo router: from app.routers import product\_router.
   * Incluye el router en la aplicación FastAPI: app.include\_router(product\_router.router, prefix="/products", tags=["Products & Services"]).
   * **¡Recuerda reiniciar el servidor FastAPI después de estos cambios!**

### **Frontend**

1. **Modificar frontend/src/App.js:**
   * Añade las rutas protegidas para la gestión de productos:  
     <Route path="/dashboard/products" element={<ManageProductsScreen />} />  
     <Route path="/dashboard/products/new" element={<CreateProductScreen />} />  
     <Route path="/dashboard/products/edit/:id" element={<EditProductScreen />} />
2. **Crear frontend/src/api/productApi.js:**
   * Implementa las funciones JavaScript para interactuar con los endpoints del backend de productos.
3. **Modificar frontend/src/screens/DashboardScreen.js:**
   * En la sección "Acciones Rápidas", añade un Button de Shadcn UI con un Link a /dashboard/products y el texto "Gestionar Productos/Servicios".
4. **Crear frontend/src/screens/ManageProductsScreen.js:**
   * Muestra una lista de los productos/servicios del usuario.
   * Botones "Crear Nuevo Producto", "Editar" y "Eliminar".
5. **Crear frontend/src/screens/CreateProductScreen.js:**
   * Un formulario para crear un nuevo producto.
   * Utiliza Input para campos de texto, Select para tipo\_producto (usando ProductType del backend, mapeado a opciones).
   * Para fotos\_urls, implementa una forma de añadir **múltiples URLs** (similar a negocios).
   * Añade un Select para negocio\_id (opcional), que liste los negocios del usuario (requerirá una llamada a businessApi.getMyBusinesses).
   * Al enviar, llama a productApi.createProduct y redirige.
6. **Crear frontend/src/screens/EditProductScreen.js:**
   * Similar a CreateProductScreen, pero precargado con datos del producto existente.

### **Prueba de UX (Experiencia de Usuario)**

1. **Registro/Login:** Inicia sesión como un usuario "Microemprendimiento" o "Freelancer".
2. **Navegación:** Desde el Dashboard, haz clic en "Gestionar Productos/Servicios".
3. **Creación:**
   * Haz clic en "Crear Nuevo Producto".
   * Rellena los campos, selecciona un tipo\_producto.
   * Añade 2-3 URLs de imágenes.
   * Si eres "Microemprendimiento" y ya creaste un negocio, intenta asociar el producto a ese negocio.
   * Guarda.
   * **Verificación:** ¿El producto aparece en la lista? ¿Los datos son correctos? ¿Se asocia al negocio si lo seleccionaste?
4. **Edición:** Haz clic en "Editar" un producto existente. Cambia campos, añade/quita URLs. Guarda.
   * **Verificación:** ¿Los cambios se reflejan?
5. **Eliminación:** Haz clic en "Eliminar" un producto. Confirma.
   * **Verificación:** ¿El producto desaparece?

### **Debugging**

* **500 Internal Server Error en el backend al crear/actualizar:**
  + **Causa:** Tipos de datos, validación de negocio\_id (si no pertenece al usuario), o problemas en la lógica CRUD.
  + **Solución:** Revisa el traceback del backend.
* **422 Unprocessable Entity en el frontend:**
  + **Causa:** Datos del formulario no coinciden con el esquema.
  + **Solución:** Inspecciona "Payload" y "Response" en la pestaña "Red" del navegador.
* **Productos no se asocian correctamente a negocios:**
  + **Causa:** negocio\_id no se envía correctamente desde el frontend, o la lógica de validación/creación en el backend falla.
  + **Solución:** Depura el payload de la solicitud POST en el frontend y la lógica de create\_product en el backend.

## **Capítulo 3: Exploración Pública de Negocios y Productos (Marketplace Básico)**

### **Descripción General**

Este capítulo transforma la página de inicio en un marketplace básico, permitiendo a los usuarios no autenticados explorar negocios y productos/servicios. Incluirá búsqueda y filtros básicos, y la capacidad de ver detalles públicos y preparar un contacto/encargo.

### **Pseudocódigo**

#### **Backend:**

# Router (backend/app/routers/business\_router.py)  
@router.get("/public")  
def get\_public\_businesses\_endpoint(db\_session, search\_query=None, rubro\_filter=None):  
 query businesses, apply search/filters  
 return list of Businesses (no auth required)  
  
@router.get("/{business\_id}/public")  
def get\_public\_business\_detail\_endpoint(business\_id, db\_session):  
 query business by id  
 return Business (no auth required)  
  
# Router (backend/app/routers/product\_router.py)  
@router.get("/public")  
def get\_public\_products\_endpoint(db\_session, search\_query=None, product\_type\_filter=None):  
 query products, apply search/filters  
 return list of Products (no auth required)  
  
@router.get("/{product\_id}/public")  
def get\_public\_product\_detail\_endpoint(product\_id, db\_session):  
 query product by id  
 return Product (no auth required)

#### **Frontend:**

// API Calls (frontend/src/api/businessApi.js, productApi.js)  
function getPublicBusinesses(params):  
 fetch GET /businesses/public with search/filter params  
 return response  
  
function getPublicBusinessById(businessId):  
 fetch GET /businesses/{businessId}/public  
 return response  
  
function getPublicProducts(params):  
 fetch GET /products/public with search/filter params  
 return response  
  
function getPublicProductById(productId):  
 fetch GET /products/{productId}/public  
 return response  
  
// App.js  
<Route path="/" element={<PublicListingScreen />} />  
<Route path="/businesses/:id" element={<BusinessDetailScreen />} />  
<Route path="/products/:id" element={<ProductDetailScreen />} />  
  
// Screens (frontend/src/screens/)  
// PublicListingScreen: (New Home Page)  
function PublicListingScreen():  
 use state for combined list of businesses/products  
 use state for search query and filters  
 useEffect to fetch public listings on load and when search/filters change  
 Input for search bar  
 Select/checkboxes for filters (rubro, tipo\_producto)  
 Display listings as cards (BusinessCard, ProductCard components)  
 Each card links to its respective public detail page  
  
// BusinessDetailScreen:  
function BusinessDetailScreen():  
 get businessId from URL params  
 useEffect to fetch public business data  
 display details (including image gallery)  
 Button "Contactar Emprendedor":  
 onClick: if not logged in, redirect to login/register with state (e.g., { from: `/businesses/${businessId}` }).  
 if logged in, show a basic contact form or redirect to a "Solicitar Encargo" form (Fase 4 prep).  
  
// ProductDetailScreen:  
function ProductDetailScreen():  
 get productId from URL params  
 useEffect to fetch public product data  
 display details (including image gallery)  
 Button "Contactar Emprendedor":  
 onClick: similar logic as BusinessDetailScreen.

### **Documentación**

#### **Endpoints API (backend/app/routers/business\_router.py)**

* GET /businesses/public
  + **Query Params**: search: Optional[str] = None, rubro: Optional[str] = None, limit: int = 10, skip: int = 0
  + **Response**: List[NegocioResponse] (200 OK)
  + **Auth**: Ninguna
* GET /businesses/{business\_id}/public
  + **Response**: NegocioResponse (200 OK)
  + **Auth**: Ninguna

#### **Endpoints API (backend/app/routers/product\_router.py)**

* GET /products/public
  + **Query Params**: search: Optional[str] = None, tipo\_producto: Optional[ProductType] = None, limit: int = 10, skip: int = 0
  + **Response**: List[ProductoResponse] (200 OK)
  + **Auth**: Ninguna
* GET /products/{product\_id}/public
  + **Response**: ProductoResponse (200 OK)
  + **Auth**: Ninguna

### **Backend**

1. **Modificar backend/app/routers/business\_router.py:**
   * Añade los endpoints GET /businesses/public y GET /businesses/{business\_id}/public.
   * Implementa la lógica de búsqueda (usando filter(Business.nombre.ilike(f"%{search}%")) o similar) y filtrado por rubro en get\_public\_businesses.
   * Asegúrate de que no requieran Depends(get\_current\_user).
2. **Modificar backend/app/routers/product\_router.py:**
   * Añade los endpoints GET /products/public y GET /products/{product\_id}/public.
   * Implementa la lógica de búsqueda y filtrado por tipo\_producto en get\_public\_products.
   * Asegúrate de que no requieran Depends(get\_current\_user).
3. **Modificar backend/app/crud/business.py y backend/app/crud/product.py:**
   * Añade funciones para obtener listados públicos con búsqueda y filtros (ej., get\_public\_businesses, get\_public\_products).
   * **¡Recuerda reiniciar el servidor FastAPI después de estos cambios!**

### **Frontend**

1. **Modificar frontend/src/App.js:**
   * Cambia la ruta principal: <Route path="/" element={<PublicListingScreen />} />.
   * Añade las rutas para los detalles públicos:  
     <Route path="/businesses/:id" element={<BusinessDetailScreen />} />  
     <Route path="/products/:id" element={<ProductDetailScreen />} />
2. **Modificar frontend/src/api/businessApi.js y frontend/src/api/productApi.js:**
   * Añade funciones para las nuevas llamadas a la API pública (ej., getPublicBusinesses, getPublicBusinessById, getPublicProducts, getPublicProductById). Estas funciones no deben enviar el token JWT.
3. **Crear frontend/src/screens/PublicListingScreen.js:**
   * Este será el nuevo componente de la página de inicio.
   * Utiliza useEffect para cargar datos de getPublicBusinesses y getPublicProducts (puedes combinarlos o mostrarlos en secciones separadas).
   * Implementa una barra de búsqueda (Input de Shadcn UI) y lógica para filtrar los resultados.
   * Muestra los negocios y productos en tarjetas atractivas (puedes crear componentes BusinessCard y ProductCard para esto).
   * Cada tarjeta debe tener un Link a su respectiva página de detalle pública.
4. **Crear frontend/src/components/BusinessCard.js y frontend/src/components/ProductCard.js:**
   * Componentes reutilizables para mostrar la información clave de un negocio/producto en el listado.
   * **Implementa un carrusel o galería simple** para mostrar las fotos\_urls (si hay múltiples).
5. **Crear frontend/src/screens/BusinessDetailScreen.js:**
   * Obtiene el id del negocio de los parámetros de la URL.
   * Utiliza useEffect para cargar los detalles del negocio usando businessApi.getPublicBusinessById.
   * Muestra toda la información del negocio, incluyendo la galería de imágenes.
   * Añade un Button de Shadcn UI "Contactar Emprendedor" o "Solicitar Encargo".
     + **Lógica del botón:** Si el usuario **no está logueado**, al hacer clic, redirige a /login o /register, pasando el id del negocio en el estado de la ruta (state: { from: location.pathname }) para que, al volver, sepa a qué negocio quería contactar.
     + Si el usuario **está logueado**, por ahora, puede mostrar un mensaje o un modal simple que diga "Funcionalidad de contacto/encargo en desarrollo (Fase 4)".
6. **Crear frontend/src/screens/ProductDetailScreen.js:**
   * Similar a BusinessDetailScreen, pero para productos.

### **Prueba de UX (Experiencia de Usuario)**

1. **Acceso Público:** Abre http://localhost:3000 en un navegador **sin iniciar sesión**.
   * **Verificación:** ¿Ves un listado de negocios y productos?
2. **Búsqueda:** Escribe un término de búsqueda en la barra.
   * **Verificación:** ¿Los resultados se filtran dinámicamente?
3. **Filtros:** Aplica un filtro (ej., por rubro o tipo de producto).
   * **Verificación:** ¿Los resultados se ajustan?
4. **Detalle Público:** Haz clic en una tarjeta de negocio o producto.
   * **Verificación:** ¿Se abre la página de detalle con toda la información y la galería de imágenes?
5. **Botón "Contactar":** Haz clic en el botón "Contactar Emprendedor" en una página de detalle pública **sin iniciar sesión**.
   * **Verificación:** ¿Te redirige a la página de Login/Registro?
6. **Botón "Contactar" (Logueado):** Inicia sesión, luego vuelve a una página de detalle pública y haz clic en "Contactar Emprendedor".
   * **Verificación:** ¿Se muestra el mensaje de "funcionalidad en desarrollo"?

### **Debugging**

* **Listados públicos vacíos:**
  + **Causa:** El backend no devuelve datos públicos, o el frontend no los está cargando/renderizando correctamente.
  + **Solución:** Usa la pestaña "Red" para verificar la respuesta de GET /businesses/public y GET /products/public. Asegúrate de que los datos se muestren en el PublicListingScreen.
* **Búsqueda/Filtros no funcionan:**
  + **Causa:** Los parámetros de búsqueda/filtro no se envían correctamente al backend, o la lógica de filtrado en el backend es incorrecta.
  + **Solución:** Revisa el "Payload" de la solicitud GET en la pestaña "Red" y la lógica de consulta en el backend.
* **Imágenes no se muestran:**
  + **Causa:** URLs de imágenes incorrectas, o el componente de galería no las renderiza bien.
  + **Solución:** Revisa las URLs en la respuesta del backend. Asegúrate de que el componente de galería itere correctamente sobre la lista de URLs.
* **Redirección del botón "Contactar" no funciona:**
  + **Causa:** Problemas con react-router-dom o la lógica de useAuth.
  + **Solución:** Revisa la consola para errores de React Router.