FreeBridge - Documentación Completa de Arquitectura

Versión: 1.0
Fecha: Noviembre 2024
Plataforma: React + Flask

Tabla de Contenidos

1. Visión General

1.1 ¿Qué es FreeBridge?

FreeBridge es una plataforma que conecta **freelancers** con **empresas**, permitiendo:

- **Empresas**: Publicar vacantes y gestionar postulaciones
- **Freelancers**: Buscar y postularse a oportunidades laborales

1.2 Stack Tecnológico

```
Frontend:  React 18 + Vite + React Router + Axios
Backend:   Flask (Python) + JWT
Base de Datos: PostgreSQL/MySQL
Estilos:   CSS Modules
```

1.3 Arquitectura de Capas

```
┌─────────────────────────────────────────┐
│         PAGES (Páginas)                 │ ← Vistas completas
├─────────────────────────────────────────┤
│      COMPONENTS (Componentes)           │ ← UI reutilizable
├─────────────────────────────────────────┤
│    CONTEXT + HOOKS (Lógica)             │ ← Estado y comportamiento
├─────────────────────────────────────────┤
```

```
│         API (Comunicación)          │ ← Backend requests
├─────────────────────────────────────┤
│       UTILS (Herramientas)          │ ← Funciones puras
└─────────────────────────────────────┘
```

---

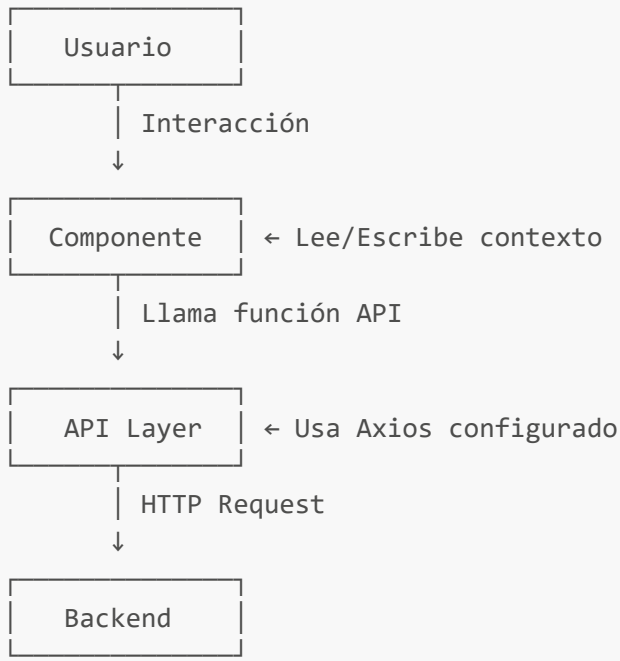
2. Arquitectura Frontend

2.1 Estructura de Carpetas

```
client-react/
├── public/                    # Archivos estáticos
├── src/
│   ├── App.jsx               # Componente raíz
│   ├── main.jsx              # Punto de entrada
│   ├── index.css             # Estilos globales
│   │
│   ├── api/                  # 🌐 Comunicación HTTP
│   │   ├── axiosConfig.js    # Configuración base Axios
│   │   ├── authApi.js        # Endpoints autenticación
│   │   ├── vacancyApi.js     # Endpoints vacantes
│   │   ├── companyApi.js     # Endpoints empresas
│   │   ├── profileApi.js     # Endpoints perfil
│   │   └── cityApi.js        # Endpoints ciudades
│   │
│   ├── components/           # 🧩 Componentes reutilizables
│   │   ├── Navbar.jsx
│   │   ├── Footer.jsx
│   │   ├── Modal.jsx
│   │   ├── loginForm.jsx
│   │   ├── registerForm.jsx
│   │   ├── vacancyCard.jsx
│   │   ├── vacancyList.jsx
│   │   └── vacancyForm.jsx
│   │
│   ├── pages/                # 📄 Páginas completas
│   │   ├── Home.jsx
│   │   ├── Login.jsx
│   │   ├── Register.jsx
│   │   ├── Vacancies.jsx
│   │   ├── Profile.jsx
│   │   └── CompanyDashboard.jsx
│   │
│   ├── context/              # 🔄 Estado global
│   │   └── AuthContext.jsx
│   │
│   ├── hooks/                # 🪝 Lógica reutilizable
│   │   └── useSessionTimeout.js
│   │
│   ├── router/               # 🧭 Navegación
│   │   └── appRouter.jsx
```

```
│   │
│   ├── styles/               # 🎨 CSS Modules
│   │   ├── Navbar.module.css
│   │   ├── Footer.module.css
│   │   └── ...
│   │
│   └── utils/                # 🛠️ Utilidades
│       └── sessionManager.js
│
├── .gitignore
├── package.json
├── vite.config.js
├── ARQUITECTURA_AUTH.md
└── GUIA_ESTILOS.md
```

2.2 Flujo de Datos



---

3. Componentes del Sistema

3.1 Sistema de Autenticación

📁 **utils/sessionManager.js**

Propósito: Gestión centralizada de localStorage

```javascript
// Funciones principales
export const getToken = () => localStorage.getItem("token");

export const setSessionData = (token, userId, role, name) => {
```

```
  localStorage.setItem("token", token);
  localStorage.setItem("userId", userId);
  localStorage.setItem("userRole", role);
  localStorage.setItem("userName", name);
  localStorage.setItem("lastActivity", Date.now());
};

export const clearSession = () => {
  localStorage.clear();
};

export const getLastActivity = () => {
  return parseInt(localStorage.getItem("lastActivity")) || Date.now();
};

export const updateLastActivity = () => {
  localStorage.setItem("lastActivity", Date.now());
};
```

Ventajas:

- ☑ Evita código duplicado
- ☑ Funciones puras (sin dependencias React)
- ☑ Fácil de testear

📁 `context/AuthContext.jsx`

Propósito: Estado global de autenticación

```
import { createContext, useContext, useState, useEffect } from "react";
import {
  getToken,
  setSessionData,
  clearSession,
} from "../utils/sessionManager";

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [isAuthenticated, setIsAuthenticated] = useState(false);
  const [userRole, setUserRole] = useState(null);
  const [userId, setUserId] = useState(null);

  // Verificar sesión al cargar
  useEffect(() => {
    const token = getToken();
    if (token) {
      setIsAuthenticated(true);
      setUserRole(localStorage.getItem("userRole"));
      setUserId(localStorage.getItem("userId"));
```

```
    }
  }, []);

  const login = (token, userId, role, name) => {
    setSessionData(token, userId, role, name);
    setIsAuthenticated(true);
    setUserRole(role);
    setUserId(userId);
  };

  const logout = () => {
    clearSession();
    setIsAuthenticated(false);
    setUserRole(null);
    setUserId(null);
  };

  return (
    <AuthContext.Provider
      value={{
        isAuthenticated,
        userRole,
        userId,
        login,
        logout,
      }}
    >
      {children}
    </AuthContext.Provider>
  );
};

export const useAuth = () => useContext(AuthContext);
```

Uso en componentes:

```
import { useAuth } from "../context/AuthContext";

const MyComponent = () => {
  const { isAuthenticated, userRole, login, logout } = useAuth();

  if (!isAuthenticated) {
    return <p>Debes iniciar sesión</p>;
  }

  return <p>Bienvenido, tu rol es: {userRole}</p>;
};
```

---

hooks/useSessionTimeout.js

Propósito: Cerrar sesión automáticamente por inactividad

```
import { useEffect } from "react";
import { useAuth } from "../context/AuthContext";
import { getLastActivity, updateLastActivity } from "../utils/sessionManager";

const useSessionTimeout = (timeout = 30 * 60 * 1000) => {
  // 30 min
  const { logout } = useAuth();

  useEffect(() => {
    // Actualizar actividad con eventos del usuario
    const events = ["mousedown", "keydown", "scroll", "touchstart"];

    const handleActivity = () => {
      updateLastActivity();
    };

    events.forEach((event) => {
      document.addEventListener(event, handleActivity);
    });

    // Verificar cada minuto si expiró
    const interval = setInterval(() => {
      const lastActivity = getLastActivity();
      const now = Date.now();

      if (now - lastActivity > timeout) {
        logout();
        window.location.href = "/login?timeout=true";
      }
    }, 60000); // Cada 60 segundos

    return () => {
      events.forEach((event) => {
        document.removeEventListener(event, handleActivity);
      });
      clearInterval(interval);
    };
  }, [logout, timeout]);
};

export default useSessionTimeout;
```

Uso:

```
import useSessionTimeout from "../hooks/useSessionTimeout";

const App = () => {
  useSessionTimeout(30 * 60 * 1000); // 30 minutos
```

```
  return <Router>...</Router>;
};
```

3.2 Navegación y Rutas

router/appRouter.jsx

```
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Home from "../pages/Home";
import Login from "../pages/Login";
import Register from "../pages/Register";
import Vacancies from "../pages/Vacancies";
import CompanyDashboard from "../pages/CompanyDashboard";
import Profile from "../pages/Profile";

const AppRouter = () => {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/login" element={<Login />} />
        <Route path="/register" element={<Register />} />
        <Route path="/vacantes" element={<Vacancies />} />
        <Route path="/company-dashboard" element={<CompanyDashboard />} />
        <Route path="/profile" element={<Profile />} />
      </Routes>
    </BrowserRouter>
  );
};

export default AppRouter;
```

---

components/Navbar.jsx

Propósito: Barra de navegación adaptativa según estado de sesión

```
import { Link, useNavigate } from "react-router-dom";
import { useAuth } from "../context/AuthContext";
import styles from "../styles/Navbar.module.css";

const Navbar = ({ isCompact = false }) => {
  const { isAuthenticated, userRole, logout } = useAuth();
  const navigate = useNavigate();

  const handleLogout = () => {
    logout();
```

```
    navigate("/");
  };

  return (
    <nav className={isCompact ? styles.navbarCompact : styles.navbar}>
      <Link to="/" className={styles.logo}>
        FreeBridge
      </Link>

      <div className={styles.navLinks}>
        <Link to="/vacantes">Vacantes</Link>

        {!isAuthenticated ? (
          <>
            <Link to="/login">Iniciar Sesión</Link>
            <Link to="/register" className={styles.btnRegister}>
              Registrarse
            </Link>
          </>
        ) : (
          <>
            {userRole === "Empresa" && (
              <Link to="/company-dashboard">Dashboard</Link>
            )}
            <Link to="/profile">Perfil</Link>
            <button onClick={handleLogout} className={styles.btnLogout}>
              Cerrar Sesión
            </button>
          </>
        )}
      </div>
    </nav>
  );
};

export default Navbar;
```

3.3 Páginas Principales

📁 `pages/Home.jsx`

Propósito: Landing page con hero section y categorías

```
import Navbar from "../components/Navbar";
import Footer from "../components/Footer";
import styles from "../styles/Home.module.css";

const Home = () => {
  return (
    <>
```

```
      <Navbar />

      <section className={styles.hero}>
        <h1>Conecta tu talento con oportunidades</h1>
        <p>La plataforma que une freelancers con empresas</p>
        <Link to="/vacantes" className={styles.btnCTA}>
          Ver Vacantes
        </Link>
      </section>

      <section className={styles.categories}>
        <h2>Categorías Populares</h2>
        <div className={styles.categoryGrid}>
          {/* Tarjetas de categorías */}
        </div>
      </section>

      <Footer />
    </>
  );
};
```

---

pages/Login.jsx

Propósito: Página de inicio de sesión

```
import { useState } from "react";
import { useNavigate } from "react-router-dom";
import { useAuth } from "../context/AuthContext";
import { loginUser } from "../api/authApi";
import LoginForm from "../components/loginForm";
import Navbar from "../components/Navbar";

const Login = () => {
  const [error, setError] = useState("");
  const { login } = useAuth();
  const navigate = useNavigate();

  const handleLogin = async (credentials) => {
    try {
      const data = await loginUser(credentials);
      login(data.token, data.id_usuario, data.rol, data.nombre);

      // Redirigir según rol
      if (data.rol === "Empresa") {
        navigate("/company-dashboard");
      } else {
        navigate("/vacantes");
      }
    } catch (err) {
```

```
      setError(err.message || "Error de autenticación");
    }
  };

  return (
    <>
      <Navbar isCompact />
      <div className="container">
        {error && <div className="alert-error">{error}</div>}
        <LoginForm onSubmit={handleLogin} />
      </div>
    </>
  );
};
```

---

pages/CompanyDashboard.jsx

Propósito: Panel de control para empresas

```
import { useState, useEffect } from "react";
import { useAuth } from "../context/AuthContext";
import { getCompanyProfile, saveCompanyProfile } from "../api/companyApi";
import VacancyForm from "../components/vacancyForm";
import Navbar from "../components/Navbar";

const CompanyDashboard = () => {
  const [hasProfile, setHasProfile] = useState(false);
  const [companyData, setCompanyData] = useState(null);
  const { userId } = useAuth();

  useEffect(() => {
    const loadProfile = async () => {
      try {
        const profile = await getCompanyProfile(userId);
        setCompanyData(profile);
        setHasProfile(true);
      } catch {
        setHasProfile(false);
      }
    };
    loadProfile();
  }, [userId]);

  if (!hasProfile) {
    return (
      <>
        <Navbar isCompact />
        <div>
          <h2>Completa tu perfil de empresa</h2>
          {/* Formulario de perfil */}
```

```
        </div>
      </>
    );
  }

  return (
    <>
      <Navbar isCompact />
      <div>
        <h1>Dashboard de {companyData.nombre_empresa}</h1>

        <section>
          <h2>Publicar Nueva Vacante</h2>
          <VacancyForm embedded={true} />
        </section>

        <section>
          <h2>Postulaciones Recibidas</h2>
          {/* Lista de postulaciones */}
        </section>
      </div>
    </>
  );
};
```

3.4 Componentes Reutilizables

📁 `components/vacancyCard.jsx`

Propósito: Tarjeta individual de vacante

```
import styles from "../styles/vacancyCard.module.css";

const VacancyCard = ({ vacancy }) => {
  const { titulo, descripcion, salario, ciudad, empresa } = vacancy;

  return (
    <div className={styles.card}>
      <h3>{titulo}</h3>
      <p className={styles.company}>{empresa}</p>
      <p className={styles.description}>{descripcion}</p>
      <div className={styles.footer}>
        <span className={styles.location}>📍 {ciudad}</span>
        <span className={styles.salary}>💰 ${salario}</span>
      </div>
      <button className={styles.btnApply}>Postularse</button>
    </div>
  );
};
```

components/Modal.jsx

Propósito: Modal genérico reutilizable

```
import { useEffect } from "react";
import styles from "../styles/Modal.module.css";

const Modal = ({ isOpen, onClose, title, children }) => {
  useEffect(() => {
    if (isOpen) {
      document.body.style.overflow = "hidden";
    } else {
      document.body.style.overflow = "unset";
    }

    return () => {
      document.body.style.overflow = "unset";
    };
  }, [isOpen]);

  if (!isOpen) return null;

  return (
    <div className={styles.overlay} onClick={onClose}>
      <div className={styles.modal} onClick={(e) => e.stopPropagation()}>
        <div className={styles.header}>
          <h2>{title}</h2>
          <button onClick={onClose} className={styles.closeBtn}>
            ×
          </button>
        </div>
        <div className={styles.content}>{children}</div>
      </div>
    </div>
  );
};
```

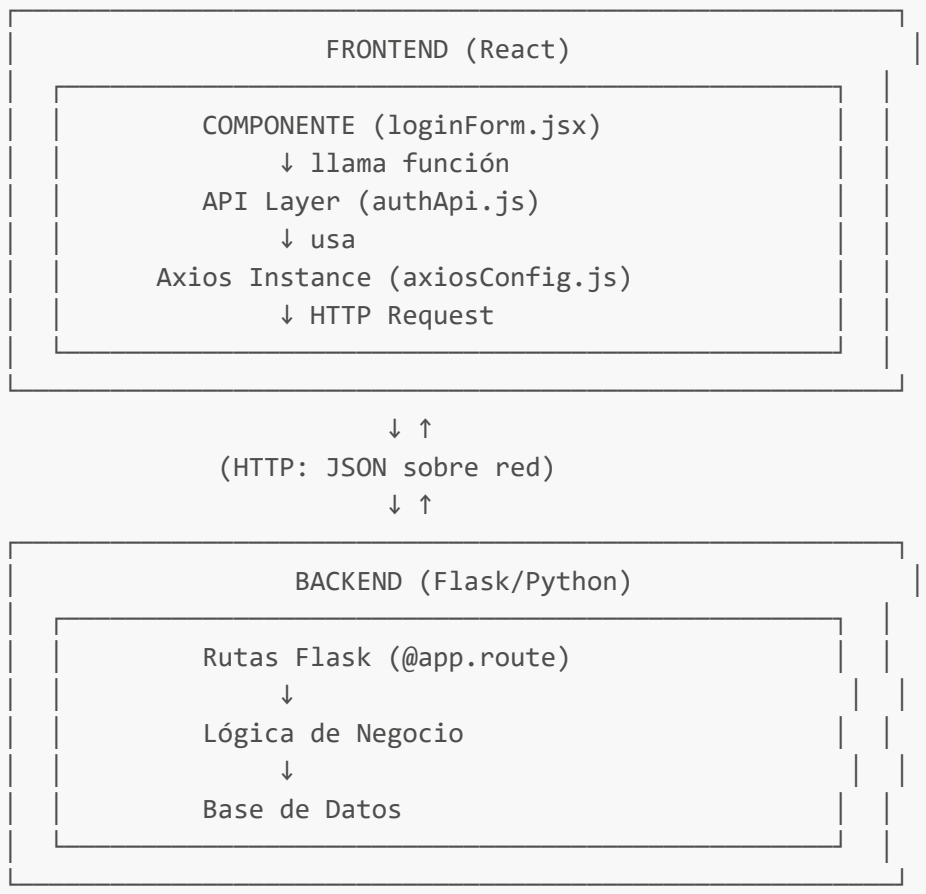
Uso:

```
const [showTerms, setShowTerms] = useState(false);

<Modal
  isOpen={showTerms}
  onClose={() => setShowTerms(false)}
  title="Términos y Condiciones"
>
  <TermsAndConditions />
</Modal>;
```

4. Comunicación Backend-Frontend

4.1 Arquitectura de Comunicación



4.2 Configuración Base - axiosConfig.js

```
import axios from "axios";

// Crear instancia personalizada
const api = axios.create({
  baseURL: "http://localhost:5000",
  headers: {
    "Content-Type": "application/json",
  },
});

// INTERCEPTOR: Se ejecuta antes de cada request
api.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem("token");
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
```

```
    return config;
  },
  (error) => Promise.reject(error)
);

// INTERCEPTOR: Se ejecuta después de cada response
api.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      // Token expirado - redirigir a login
      localStorage.clear();
      window.location.href = "/login?expired=true";
    }
    return Promise.reject(error);
  }
);

export default api;
```

¿Por qué usar interceptors?

- ☑ **DRY**: No repetir headers en cada request
- ☑ **Seguridad**: Token se agrega automáticamente
- ☑ **Manejo global de errores**: 401 → logout automático

4.3 Funciones API por Módulo

📁 `api/authApi.js`

```
import api from "./axiosConfig";

export const registerUser = async (userData) => {
  try {
    const response = await api.post("/api/registro", userData);
    // userData = { email, password, nombre, apellido, rol }
    return response.data;
  } catch (error) {
    throw error.response?.data || error.message;
  }
};

export const loginUser = async (credentials) => {
  try {
    const response = await api.post("/api/login", credentials);
    // credentials = { email, password }
    return response.data;
    // Retorna: { token, id_usuario, rol, nombre, apellido }
  } catch (error) {
    throw error.response?.data || error.message;
```

```
  }
};
```

---

api/vacancyApi.js

```
import api from "./axiosConfig";

export const getVacantes = async () => {
  try {
    const response = await api.get("/vacantes/");
    return response.data;
  } catch (error) {
    throw error.response?.data || error.message;
  }
};

export const crearVacante = async (vacancyData) => {
  try {
    const response = await api.post("/vacantes/crear", vacancyData);
    return response.data;
  } catch (error) {
    if (error.response?.status === 401) {
      throw new Error("Sesión expirada. Inicia sesión nuevamente.");
    }
    throw error.response?.data || error.message;
  }
};

export const getVacantePorId = async (id) => {
  try {
    const response = await api.get(`/vacantes/${id}`);
    return response.data;
  } catch (error) {
    throw error.response?.data || error.message;
  }
};
```

---

api/companyApi.js

```
import api from "./axiosConfig";

export const saveCompanyProfile = async (companyData) => {
  try {
    const response = await api.post("/api/empresa/perfil", companyData);
    return response.data;
  } catch (error) {
```

```
    throw error.response?.data || error.message;
  }
};

export const getCompanyProfile = async (userId) => {
  try {
    const response = await api.get(`/api/empresa/perfil/${userId}`);
    return response.data;
  } catch (error) {
    throw error.response?.data || error.message;
  }
};
```

4.4 Flujo Completo de un Request

Ejemplo: Login de Usuario

```
┌─────────────────────────────────────────────────────────┐
│ 1. USUARIO                                              │
│    Ingresa email y password en formulario               │
│    ↓                                                    │
│ 2. COMPONENTE (loginForm.jsx)                           │
│    handleSubmit() → llama loginUser(credentials)        │
│    ↓                                                    │
│ 3. API (authApi.js)                                     │
│    export const loginUser = async (credentials) => {    │
│      return api.post('/api/login', credentials);        │
│    }                                                    │
│    ↓                                                    │
│ 4. INTERCEPTOR (axiosConfig.js)                         │
│    Agrega headers (si existe token previo)              │
│    ↓                                                    │
│ 5. HTTP REQUEST                                         │
│    POST http://localhost:5000/api/login                 │
│    Headers: { Content-Type: application/json }          │
│    Body: { email: "user@example.com", password: "123" } │
│    ↓                                                    │
└─────────────────────────────────────────────────────────┘
                    (VIAJA POR RED)
┌─────────────────────────────────────────────────────────┐
│ 6. BACKEND FLASK                                        │
│    @app.route('/api/login', methods=['POST'])           │
│    def login():                                         │
│        data = request.get_json()                        │
│        # Validar credenciales en DB                     │
│        # Generar JWT token                              │
│        return jsonify({                                 │
│          "token": "eyJhbGc...",                         │
│          "id_usuario": 5,                               │
│          "rol": "Empresa",                              │
```

```
│          "nombre": "Juan"                                   │
│        }), 200                                              │
└─────────────────────────────────────────────────────────────┘

                    (VIAJA POR RED)

┌─────────────────────────────────────────────────────────────┐
│ 7. FRONTEND RECIBE RESPUESTA                                │
│    const data = await loginUser(credentials);               │
│    // data = { token: "eyJ...", id_usuario: 5, ... }        │
│    ↓                                                        │
│ 8. COMPONENTE PROCESA                                       │
│    login(data.token, data.id_usuario, data.rol, ...)        │
│    ↓                                                        │
│ 9. AUTH CONTEXT                                             │
│    setSessionData() → Guarda en localStorage                │
│    setIsAuthenticated(true)                                 │
│    ↓                                                        │
│ 10. NAVEGACIÓN                                              │
│     navigate('/company-dashboard')                          │
└─────────────────────────────────────────────────────────────┘
```

4.5 Anatomía de un HTTP Request

```
await api.post('/api/login', { email: 'user@example.com', password: '123' })
        │        │               └─ BODY (request payload)
        │        └─ RUTA (se concatena con baseURL)
        └─ MÉTODO HTTP (GET, POST, PUT, DELETE)
```

Request HTTP resultante:

```
POST http://localhost:5000/api/login
Headers:
  Content-Type: application/json
  Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Body:
  {
    "email": "user@example.com",
    "password": "123"
  }
```

Response del Backend:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
```

```
  "id_usuario": 5,
  "rol": "Empresa",
  "nombre": "Juan",
  "apellido": "Pérez"
}
```

4.6 Tipos de Requests HTTP

Método	Propósito	Ejemplo FreeBridge	Archivo API
GET	Obtener datos	`getVacantes()`	vacancyApi.js
POST	Crear recurso	`crearVacante()`	vacancyApi.js
PUT	Actualizar completo	`updateProfile()`	profileApi.js
PATCH	Actualizar parcial	*(no implementado)*	-
DELETE	Eliminar recurso	*(no implementado)*	-

Ejemplos:

```
// GET con parámetros de query
const response = await api.get("/vacantes?ciudad=Bogotá&salario_min=2000");

// GET con parámetros de ruta
const response = await api.get(`/vacantes/${vacanteId}`);

// POST con datos
const response = await api.post("/vacantes/crear", {
  titulo: "Desarrollador React",
  descripcion: "...",
  salario: 3000,
});

// PUT (actualización completa)
const response = await api.put(`/usuario/${userId}`, {
  nombre: "Juan",
  apellido: "Pérez",
  email: "juan@example.com",
});
```

4.7 Manejo de Errores

En el Frontend

```
try {
  const data = await loginUser(credentials);
```

```
    // Éxito
    console.log("Login exitoso", data);
  } catch (error) {
    // Error

    if (error.response) {
      // Backend respondió con error (4xx, 5xx)
      console.error("Error del servidor:", error.response.data);
      setError(error.response.data.message || "Error desconocido");
    } else if (error.request) {
      // Request enviado pero sin respuesta (backend caído)
      console.error("Sin respuesta del servidor:", error.request);
      setError("No se pudo conectar al servidor");
    } else {
      // Error al configurar request
      console.error("Error:", error.message);
      setError("Error inesperado");
    }
  }
```

En el Backend (Flask)

```
@app.route('/api/login', methods=['POST'])
def login():
    try:
        data = request.get_json()

        # Validaciones
        if not data.get('email') or not data.get('password'):
            return jsonify({"error": "Campos incompletos"}), 400

        # Lógica de autenticación...

        return jsonify({
            "token": token,
            "id_usuario": user_id,
            "rol": rol
        }), 200

    except Exception as e:
        return jsonify({"error": str(e)}), 500
```

4.8 Autenticación con JWT

Flujo de Token

```
1. LOGIN EXITOSO
   Frontend → Backend: { email, password }
   Backend → Frontend: { token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..." }

2. FRONTEND GUARDA TOKEN
   localStorage.setItem('token', token)

3. REQUESTS POSTERIORES
   Interceptor agrega automáticamente:
   headers: { Authorization: "Bearer eyJhbGc..." }

4. BACKEND VALIDA TOKEN
   - Decodifica JWT
   - Verifica firma
   - Extrae payload (id_usuario, rol)
   - Permite/deniega acceso

5. TOKEN EXPIRA
   Backend → 401 Unauthorized
   Interceptor detecta → clearSession() → redirect('/login')
```

Ejemplo: Endpoint Protegido

Frontend:

```
export const crearVacante = async (vacancyData) => {
  try {
    // El interceptor agrega: Authorization: Bearer <token>
    const response = await api.post("/vacantes/crear", vacancyData);
    return response.data;
  } catch (error) {
    if (error.response?.status === 401) {
      throw new Error("Sesión expirada");
    }
    throw error.response?.data || error.message;
  }
};
```

Backend:

```
from functools import wraps
import jwt

def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        token = request.headers.get('Authorization')
```

```
        if not token:
            return jsonify({"error": "Token faltante"}), 401

        try:
            token = token.split(" ")[1]  # Quitar "Bearer "
            data = jwt.decode(token, app.config['SECRET_KEY'], algorithms=
["HS256"])
            current_user_id = data['id_usuario']

        except jwt.ExpiredSignatureError:
            return jsonify({"error": "Token expirado"}), 401
        except:
            return jsonify({"error": "Token inválido"}), 401

        return f(current_user_id, *args, **kwargs)

    return decorated

@app.route('/vacantes/crear', methods=['POST'])
@token_required
def crear_vacante(current_user_id):
    # current_user_id extraído del token
    data = request.get_json()
    # Crear vacante en DB...
    return jsonify({"message": "Vacante creada"}), 201
```

---

5. Sistema de Autenticación

5.1 Flujo Completo de Autenticación

```
┌─────────────────────────────────────────────────────────┐
│  1. Usuario llena formulario en loginForm.jsx          │
│     ↓                                                   │
│  2. Componente llama loginUser(credentials)            │
│     ↓                                                   │
│  3. authApi.js hace POST a /api/login                  │
│     ↓                                                   │
│  4. Backend valida credenciales en DB                  │
│     ↓                                                   │
│  5. Backend genera JWT token                           │
│     ↓                                                   │
│  6. Backend retorna { token, id_usuario, rol, nombre }  │
│     ↓                                                   │
│  7. loginForm llama login() del AuthContext            │
│     ↓                                                   │
│  8. AuthContext llama setSessionData() de utils        │
│     ↓                                                   │
│  9. sessionManager guarda en localStorage              │
│     ↓                                                   │
│ 10. Estado global se actualiza (isAuthenticated=true)   │
```

```
│     ↓                                                    │
│ 11. Navbar re-renderiza (muestra "Perfil/Logout")        │
│     ↓                                                    │
│ 12. useSessionTimeout inicia monitoreo de inactividad    │
│     ↓                                                    │
│ 13. Usuario redirigido a dashboard según rol             │
└──────────────────────────────────────────────────────────┘
```

5.2 Registro de Usuario

```
┌──────────────────────────────────────────────────────────┐
│  1. Usuario selecciona rol (Freelancer/Empresa)          │
│     ↓                                                    │
│  2. Llena formulario de registro                         │
│     ↓                                                    │
│  3. Acepta términos y condiciones                        │
│     ↓                                                    │
│  4. registerForm llama registerUser(userData)            │
│     ↓                                                    │
│  5. authApi.js hace POST a /api/registro                 │
│     ↓                                                    │
│  6. Backend crea usuario en DB                           │
│     ↓                                                    │
│  7. Backend retorna { message, id_usuario }              │
│     ↓                                                    │
│  8. Frontend muestra SuccessModal                        │
│     ↓                                                    │
│  9. Usuario redirigido a /login                          │
└──────────────────────────────────────────────────────────┘
```

5.3 Protección de Rutas

```jsx
// Ejemplo de componente protegido
import { useAuth } from "../context/AuthContext";
import { useNavigate } from "react-router-dom";
import { useEffect } from "react";

const CompanyDashboard = () => {
  const { isAuthenticated, userRole } = useAuth();
  const navigate = useNavigate();

  useEffect(() => {
    if (!isAuthenticated) {
      navigate("/login");
    } else if (userRole !== "Empresa") {
      navigate("/");
    }
  }, [isAuthenticated, userRole, navigate]);
```

```
  if (!isAuthenticated || userRole !== "Empresa") {
    return null; // o un spinner
  }

  return <div>{/* Contenido del dashboard */}</div>;
};
```

---

6. Guía de Desarrollo

6.1 Agregar Nueva Página

1. Crear componente en `/pages`:

```
import Navbar from "../components/Navbar";
import Footer from "../components/Footer";

const NewPage = () => {
  return (
    <>
      <Navbar isCompact />
      <div className="container">
        <h1>Nueva Página</h1>
      </div>
      <Footer />
    </>
  );
};

export default NewPage;
```

2. Crear estilos (opcional):

```
.container {
  max-width: 1200px;
  margin: 0 auto;
  padding: 2rem;
}
```

3. Agregar ruta:

```
import NewPage from "../pages/NewPage";

<Route path="/new-page" element={<NewPage />} />;
```

6.2 Agregar Nuevo Endpoint

1. Crear función en archivo API correspondiente:

```
import api from "./axiosConfig";

export const getFeatureData = async (id) => {
  try {
    const response = await api.get(`/api/feature/${id}`);
    return response.data;
  } catch (error) {
    throw error.response?.data || error.message;
  }
};

export const createFeature = async (data) => {
  try {
    const response = await api.post("/api/feature", data);
    return response.data;
  } catch (error) {
    throw error.response?.data || error.message;
  }
};
```

2. Usar en componente:

```
import { useState, useEffect } from "react";
import { getFeatureData } from "../api/newFeatureApi";

const MyComponent = () => {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const result = await getFeatureData(123);
        setData(result);
      } catch (error) {
        console.error(error);
      } finally {
        setLoading(false);
      }
    };
    fetchData();
  }, []);

  if (loading) return <p>Cargando...</p>;
```

```
  return <div>{JSON.stringify(data)}</div>;
};
```

6.3 Crear Componente Reutilizable

```
import styles from "../styles/Button.module.css";

const Button = ({
  children,
  onClick,
  variant = "primary",
  disabled = false,
}) => {
  return (
    <button
      className={`${styles.btn} ${styles[variant]}`}
      onClick={onClick}
      disabled={disabled}
    >
      {children}
    </button>
  );
};

export default Button;
```

```
.btn {
  padding: 0.75rem 1.5rem;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  font-size: 1rem;
  transition: all 0.3s;
}

.primary {
  background-color: var(--primary);
  color: white;
}

.primary:hover {
  background-color: var(--dark);
}

.secondary {
  background-color: #6c757d;
  color: white;
}
```

6.4 Convenciones de Código

Nombres de Archivos

```
Componentes:     PascalCase.jsx (Navbar.jsx, Footer.jsx)
Hooks:           camelCase.js (useAuth.js, useSessionTimeout.js)
Utils:           camelCase.js (sessionManager.js)
Estilos:         PascalCase.module.css (Navbar.module.css)
APIs:            camelCase.js (authApi.js, vacancyApi.js)
```

Importaciones

```
// 1. Librerías externas
import React, { useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";

// 2. APIs y servicios
import { loginUser } from "../api/authApi";

// 3. Contextos y hooks
import { useAuth } from "../context/AuthContext";

// 4. Componentes
import Navbar from "../components/Navbar";

// 5. Estilos
import styles from "../styles/Login.module.css";
```

Estructura de Componentes

```
const MyComponent = () => {
  // 1. Hooks de estado
  const [data, setData] = useState(null);

  // 2. Hooks de contexto
  const { isAuthenticated } = useAuth();

  // 3. Hooks de navegación
  const navigate = useNavigate();

  // 4. Efectos
  useEffect(() => {
    // ...
  }, []);
```

```
  // 5. Funciones de manejo
  const handleSubmit = () => {
    // ...
  };

  // 6. Renderizado condicional
  if (!data) return <p>Cargando...</p>;

  // 7. Renderizado principal
  return <div>{/* JSX */}</div>;
};
```

---

7. Debugging y Testing

7.1 Debugging con DevTools

Network Tab

```
F12 → Network → Filtro: XHR/Fetch

Verificar:
☑ Request URL correcta
☑ Request Method (GET, POST, etc.)
☑ Status Code (200, 400, 401, 500)
☑ Request Headers (Authorization presente)
☑ Request Payload (datos enviados)
☑ Response (datos recibidos)
```

Console Logs Estratégicos

```
// En funciones API
export const loginUser = async (credentials) => {
  console.log("🔄 [AUTH] Iniciando login con:", credentials);

  try {
    const response = await api.post("/api/login", credentials);
    console.log("✅ [AUTH] Login exitoso:", response.data);
    return response.data;
  } catch (error) {
    console.error("❌ [AUTH] Error en login:", error.response?.data);
    throw error.response?.data || error.message;
  }
};

// En componentes
const handleLogin = async () => {
```

```
  console.log("🟢 [LOGIN] Enviando formulario...");
  try {
    const data = await loginUser(formData);
    console.log("☑️ [LOGIN] Respuesta recibida:", data);
  } catch (error) {
    console.error("❌ [LOGIN] Error:", error);
  }
};
```

7.2 React DevTools

```
Instalar extensión: React Developer Tools

Uso:
1. Inspeccionar componentes
2. Ver props y state
3. Ver contexto (AuthContext)
4. Profiler para performance
```

7.3 Testing de Endpoints

Con Thunder Client (VS Code)

```
{
  "method": "POST",
  "url": "http://localhost:5000/api/login",
  "headers": {
    "Content-Type": "application/json"
  },
  "body": {
    "email": "test@example.com",
    "password": "123456"
  }
}
```

Con cURL

```
# Login
curl -X POST http://localhost:5000/api/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@example.com","password":"123456"}'

# Endpoint protegido
```

```
curl -X GET http://localhost:5000/api/empresa/perfil/5 \
  -H "Authorization: Bearer eyJhbGc..."
```

7.4 Errores Comunes

Error	Causa	Solución
CORS Error	Backend no permite requests del frontend	Configurar CORS en Flask
401 Unauthorized	Token inválido o expirado	Verificar token en localStorage
404 Not Found	Ruta incorrecta	Verificar baseURL + endpoint
500 Internal Server	Error en backend	Ver logs del servidor Flask
Network Error	Backend no está corriendo	Iniciar servidor Flask

---

8. Anexos

8.1 Variables de Entorno

Desarrollo:

```
# .env.development
VITE_API_URL=http://localhost:5000
```

Producción:

```
# .env.production
VITE_API_URL=https://api.freebridge.com
```

Uso:

```
const api = axios.create({
  baseURL: import.meta.env.VITE_API_URL || "http://localhost:5000",
});
```

8.2 Paleta de Colores

```
:root {
  /* Colores Primarios */
  --primary: #16a085; /* Cyan principal */
  --dark: #16685a; /* Cyan oscuro */
```

```
  --light: #b8f2e6; /* Cyan claro */

  /* Colores Secundarios */
  --secondary: #f39c12; /* Naranja */
  --success: #27ae60; /* Verde */
  --danger: #e74c3c; /* Rojo */

  /* Neutros */
  --gray-100: #f8f9fa;
  --gray-200: #e9ecef;
  --gray-700: #495057;
  --gray-900: #212529;
}
```

8.3 Comandos Útiles

```
# Instalar dependencias
npm install

# Iniciar servidor de desarrollo
npm run dev

# Build para producción
npm run build

# Preview de build
npm run preview

# Linting
npm run lint
```

8.4 Estructura de Base de Datos (Referencia)

```
-- Tabla usuarios
CREATE TABLE usuarios (
  id_usuario SERIAL PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  nombre VARCHAR(100),
  apellido VARCHAR(100),
  rol VARCHAR(20) CHECK (rol IN ('Freelancer', 'Empresa')),
  created_at TIMESTAMP DEFAULT NOW()
);

-- Tabla empresas
CREATE TABLE empresas (
  id_empresa SERIAL PRIMARY KEY,
```

```
  id_usuario INT REFERENCES usuarios(id_usuario),
  nombre_empresa VARCHAR(200),
  descripcion TEXT,
  sitio_web VARCHAR(255),
  telefono VARCHAR(20)
);

-- Tabla vacantes
CREATE TABLE vacantes (
  id_vacante SERIAL PRIMARY KEY,
  id_empresa INT REFERENCES empresas(id_empresa),
  titulo VARCHAR(200) NOT NULL,
  descripcion TEXT,
  salario DECIMAL(10,2),
  id_ciudad INT REFERENCES ciudades(id_ciudad),
  created_at TIMESTAMP DEFAULT NOW()
);
```

8.5 Checklist de Implementación

Nueva Feature Completa

- ☐ Diseñar endpoint en backend
- ☐ Crear función en archivo API
- ☐ Crear/modificar componente
- ☐ Crear estilos CSS Module
- ☐ Agregar ruta (si es página)
- ☐ Testear en navegador
- ☐ Verificar en Network Tab
- ☐ Manejar errores
- ☐ Agregar loading states
- ☐ Documentar código

8.6 Recursos Adicionales

Documentación Oficial:

- React Docs
- React Router
- Axios
- Vite

Archivos de Referencia Interna:

- `ARQUITECTURA_AUTH.md` - Detalles de autenticación
- `GUIA_ESTILOS.md` - Convenciones de CSS

🎓 Conclusión

Esta documentación cubre la arquitectura completa de FreeBridge frontend. Para consultas específicas sobre:

- **Backend Flask**: Consultar documentación del servidor
- **Base de Datos**: Ver esquema SQL
- **Despliegue**: Ver guía de deployment

Autor: Equipo FreeBridge
Última actualización: Noviembre 2024
Contacto: dev@freebridge.com

Este documento debe actualizarse cada vez que se agreguen nuevas features o se modifique la arquitectura.