



Instituto

**Emiliani
Somascos**

Presentación de la Segunda Fase **Expotec 2025**

5to Bachillerato Industrial y Perito en Computación “B”

Tecnología II – Área Técnica

Unidad III

Instructor: Dilan Suy

Introducción

En un mundo donde la tecnología y la competencia están creciendo, los métodos de contratación necesitan herramientas rápidas, precisas y flexibles. Con este requisito, emerge JobQuest, un sistema para automatizar entrevistas destinadas a mejorar y actualizar cómo las empresas manejan la contratación. Esta iniciativa tiene como objetivo proporcionar una herramienta digital que ayude en la primera evaluación de los solicitantes, lo que permite a las empresas conservar el tiempo y mejorar su proceso de elección de elección.

En la creación del sistema, se han organizado piezas básicas como roles de usuario, un modelo de base de datos que respalda la lógica del sistema y un menú central para las funciones principales. Estos componentes forman la base para un instrumento flexible y adecuados para varios requisitos de reclutamiento.

Esta documentación tiene la intención de explicar las características técnicas y operativas del proyecto, y también actuar como un manual para su comprensión, ejecución y crecimiento.

Indice

Desarrollo de roles: roles dentro de la app (4-5)

Usuarios, empresas y administradores

Modelo de base de datos relacionada (6-23)

Entidades en la base de datos, modelo entidad relación, fases de normalización, modelo lógico, relaciones clavé

Menú principal: los primeros forms de la plataforma (24-37)

Backend y Frontend

Roles dentro de la plataforma:

La plataforma será un **sistema de entrevistas automatizado** y los **roles** que se desarrollarán serán los siguientes:

Usuarios

Empresas

Administradores

Cada **rol** dentro de la plataforma tiene una jerarquía, los **usuarios** podrán hacer uso de la plataforma para ser **entrevistados**. Los **administradores** velarán por los **usuarios** y **empresas** que utilicen la plataforma. Las **empresas** serán aquellas entidades que hagan uso de la plataforma para entrevistar a los **usuarios**.

Usuarios

Los usuarios constituirán a las personas entrevistadas, básicamente será el usuario por defecto dentro de la plataforma. Cada usuario tendrá un espacio dentro de la base de datos, es decir podrá registrar un usuario con contraseña y correo electrónico.

Empresas

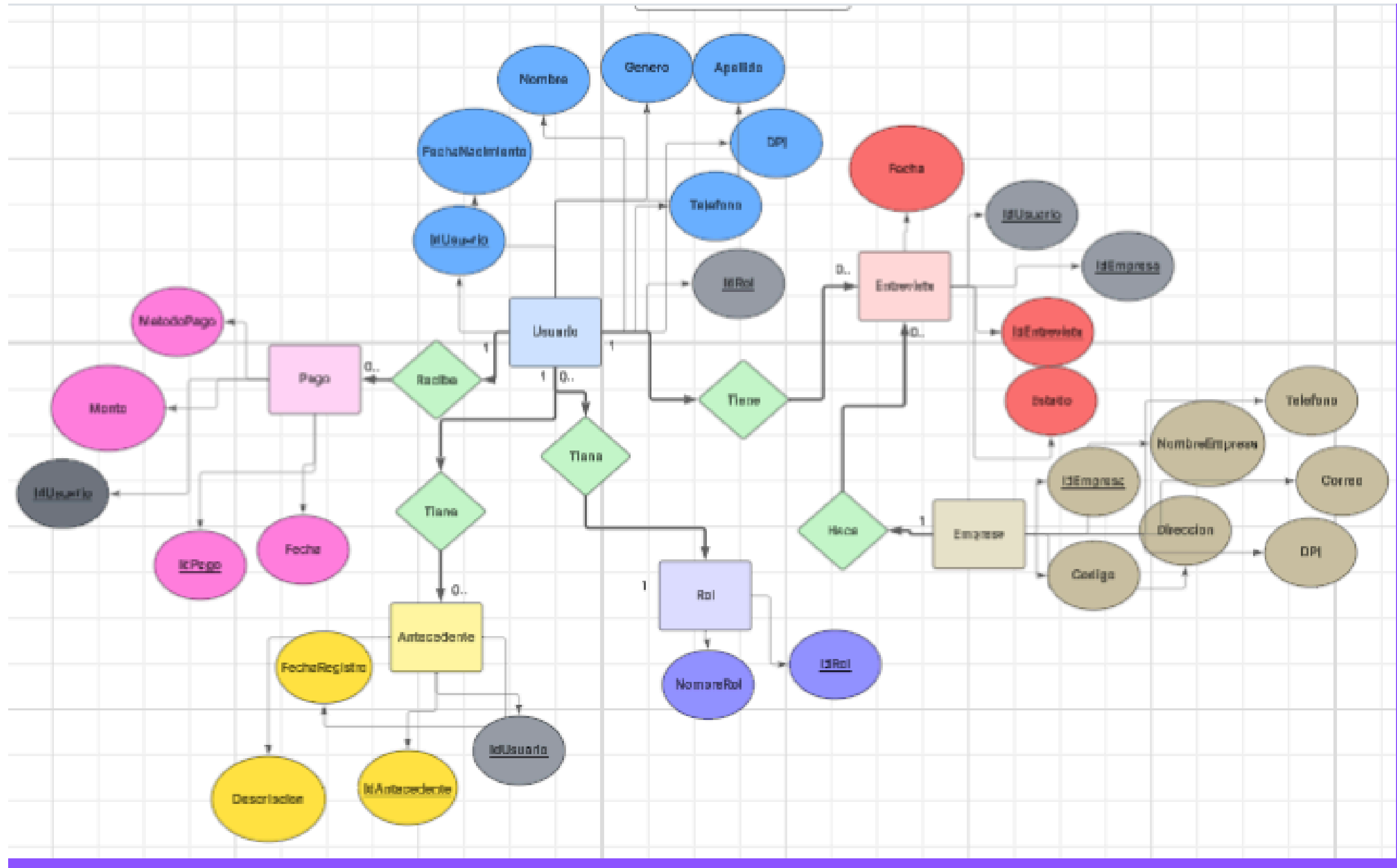
Las empresas representan aquellas entidades que utilizarán el servicio como tal, las empresas manejan a un conjunto de posibles candidatos de personal laboral.

Administradores

Los administradores representan aquellas personas que manejan la plataforma como tal, básicamente son los desarrolladores, las personas que prestan el servicio como tal. Estos podrán administrar a las empresas y usuarios dentro de la plataforma.

Modelo de base de datos relacionada

MODELO ENTIDAD-RELACIÓN



Entidades de la Base de Datos

Usuarios

IdUsuario (PK)
Nombre
Apellido
Telefono
FechaNacimiento
Genero
DPI
IdRol (FK)

Rol

IdRol (PK)
NombreRol

Empresas

IdEmpresa (PK)
Nombre
Direccion
Codigo
Telefono
Correo
DPI

MetodoDePago

IdMetodo
Metodo

Antecedentes

IdAntecedente
IdUsuario
Descripcion

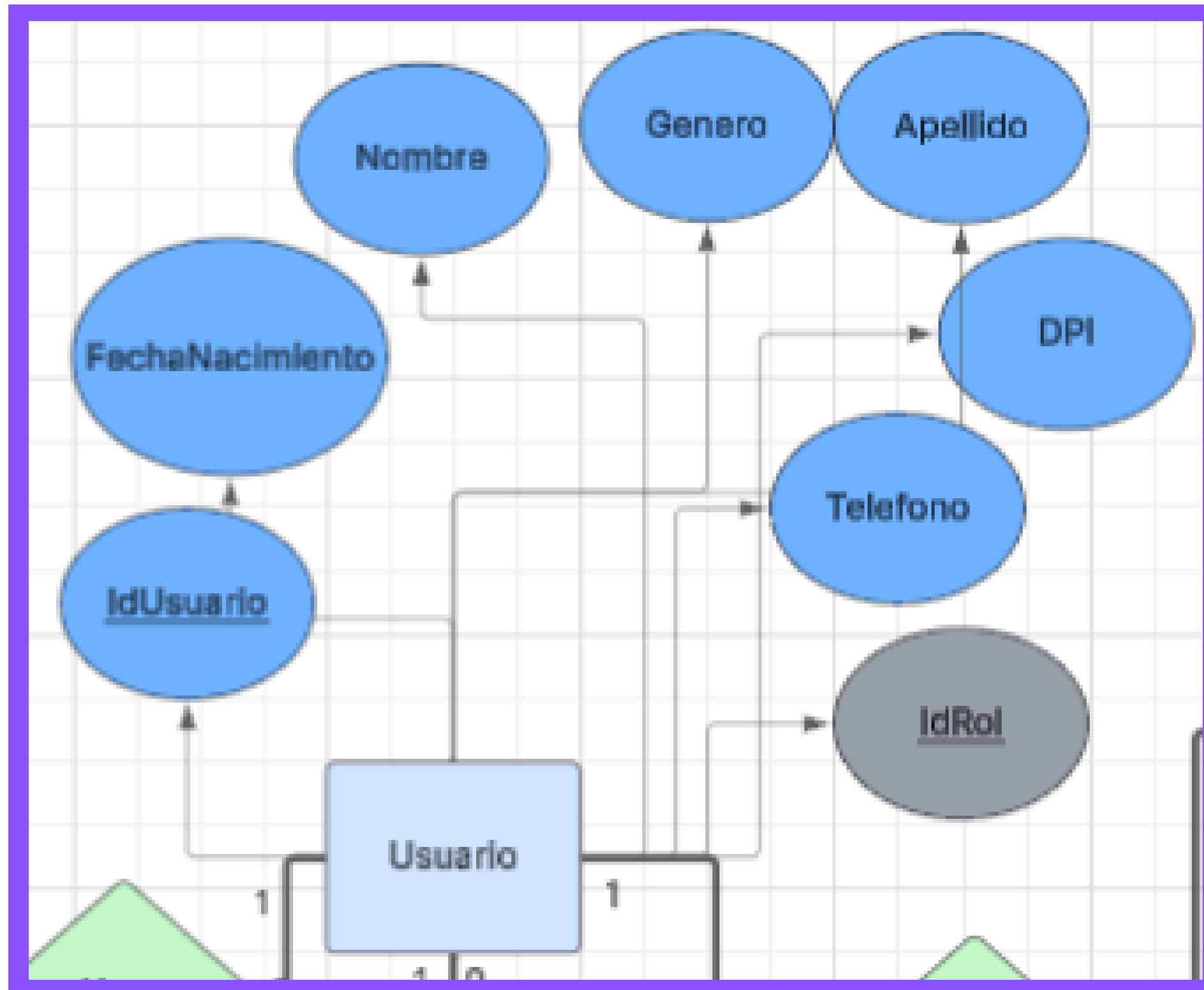
Entrevista

IdEntrevista (PK)
Fecha
IdUsuario (FK)
IdEmpresa (FK)

Pago

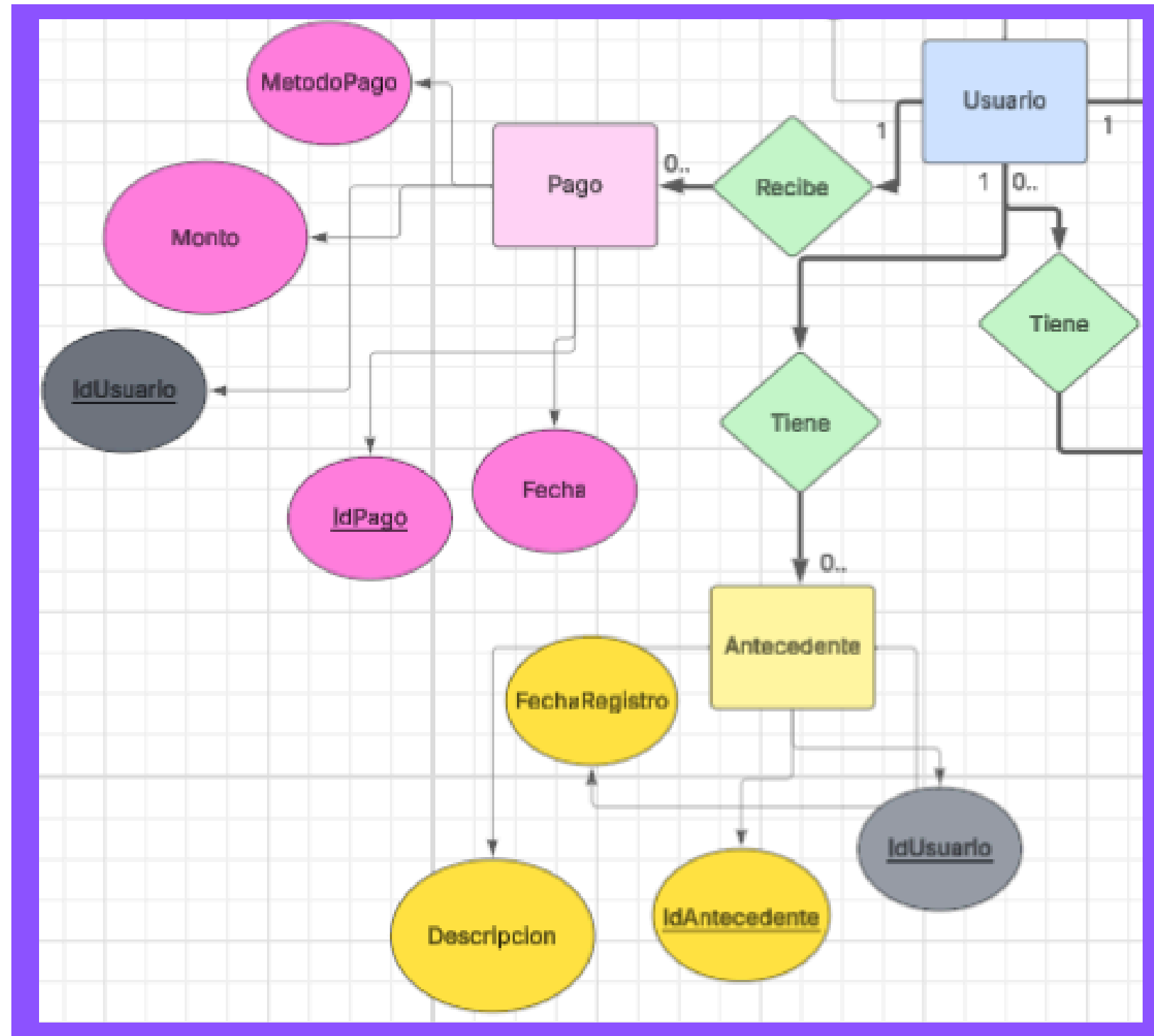
IdPago (PK)
NombreEmpresa
Monto
IdUsuario
IdMetodo

MODELO ENTIDAD RELACIÓN-USUARIOS



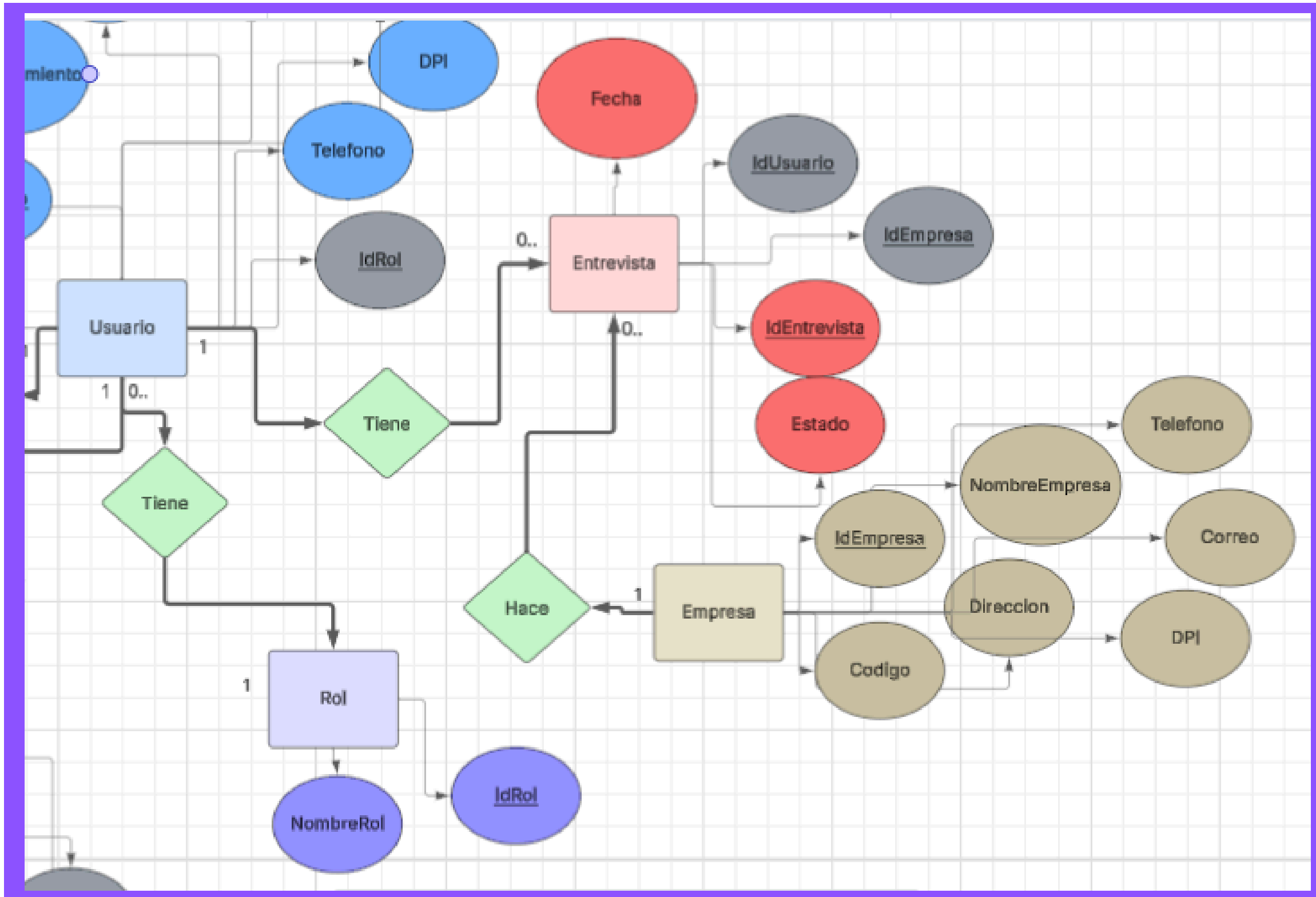
Este modelo representa la colección de usuarios, incluyendo sus atributos, la clave primaria (IdUsuario) y una clave foránea (IdRol) que enlaza con la colección ROL.

MODELO ENTIDAD RELACIÓN-USUARIOS



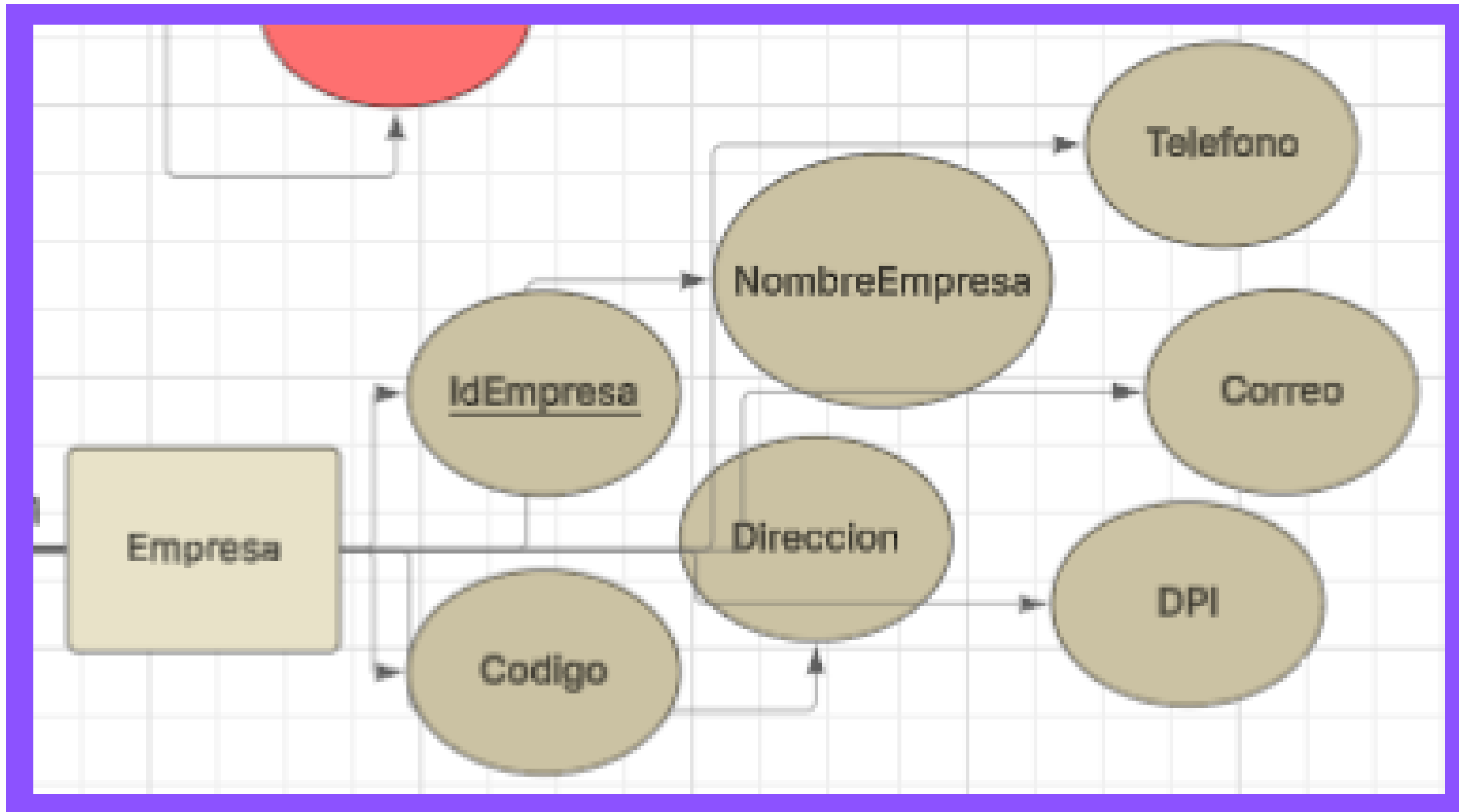
Este modelo representa como el usuario tiene la acción de poder recibir diferentes pagos y así mismo un usuario puede tener diversos antecedentes.

MODELO ENTIDAD RELACIÓN-USUARIOS



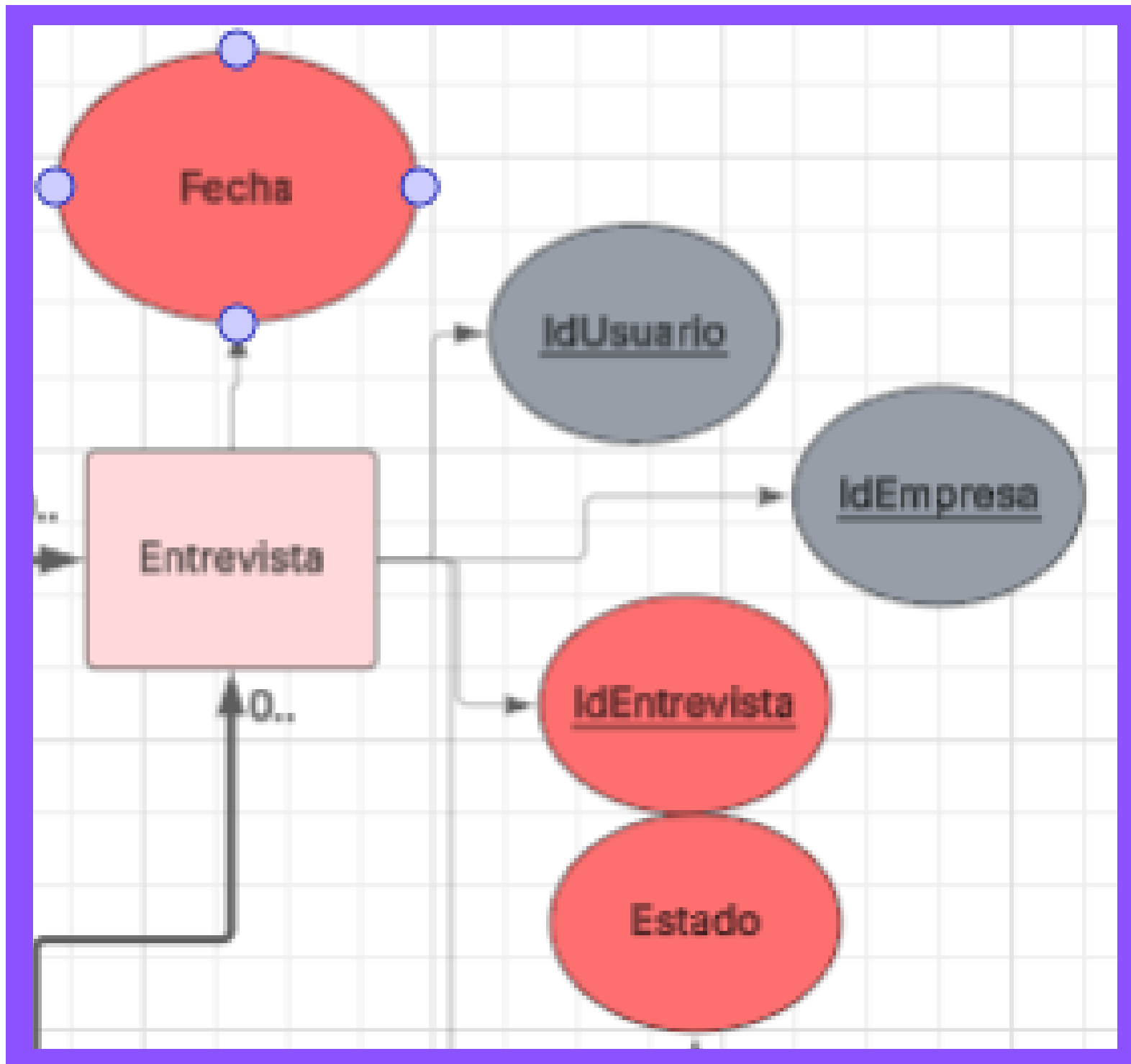
El apartado de ROL puede tener varios usuarios, un usuario puede tener diversas entrevistas y una empresa puede hacer o realizar varias entrevistas.

MODELO ENTIDAD RELACIÓN-EMPRESAS



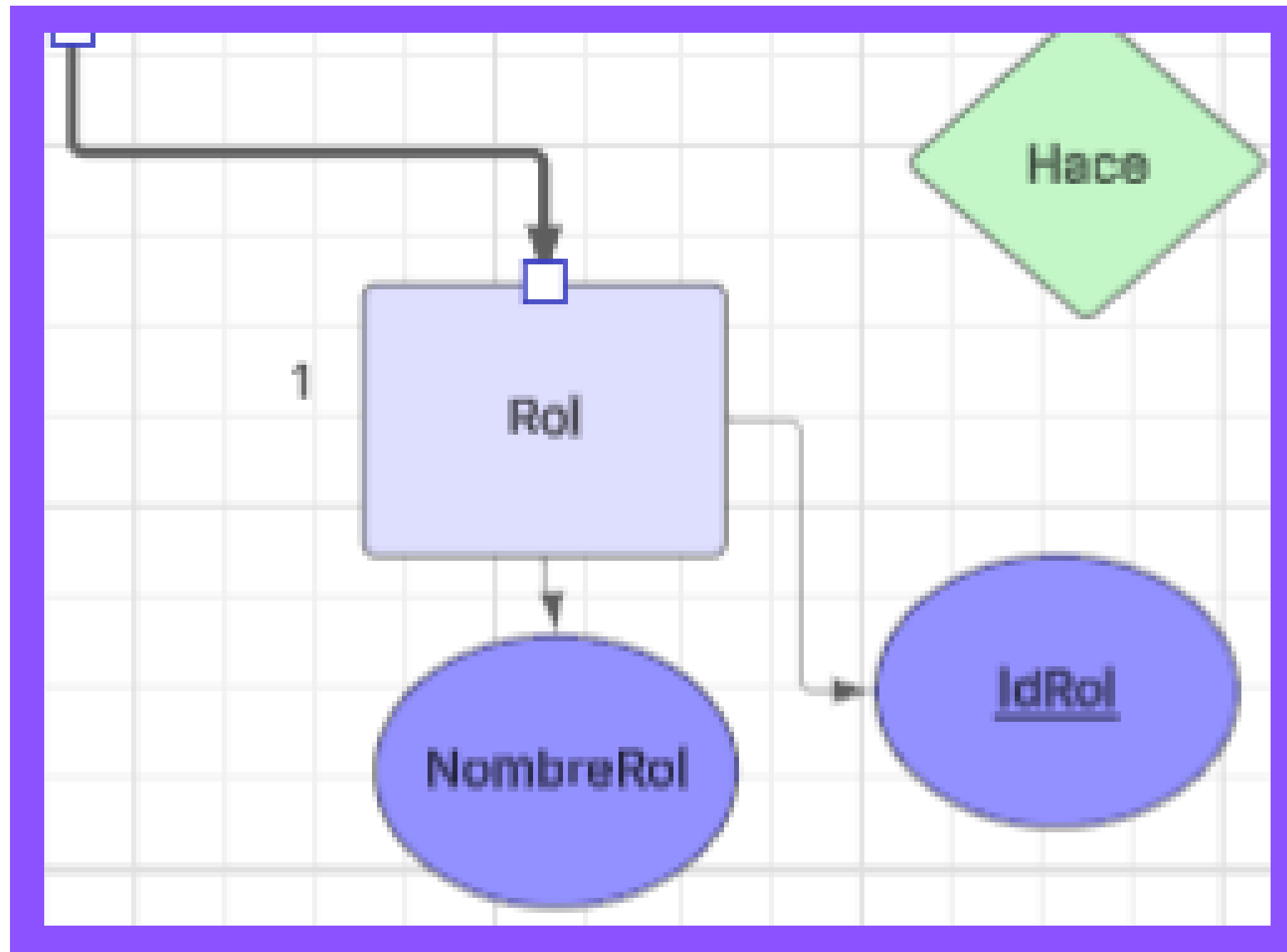
Este modelo contiene la colección de la empresa, con ella sus atributos y la clave primaria de la misma.

MODELO ENTIDAD RELACIÓN-ENTREVISTAS



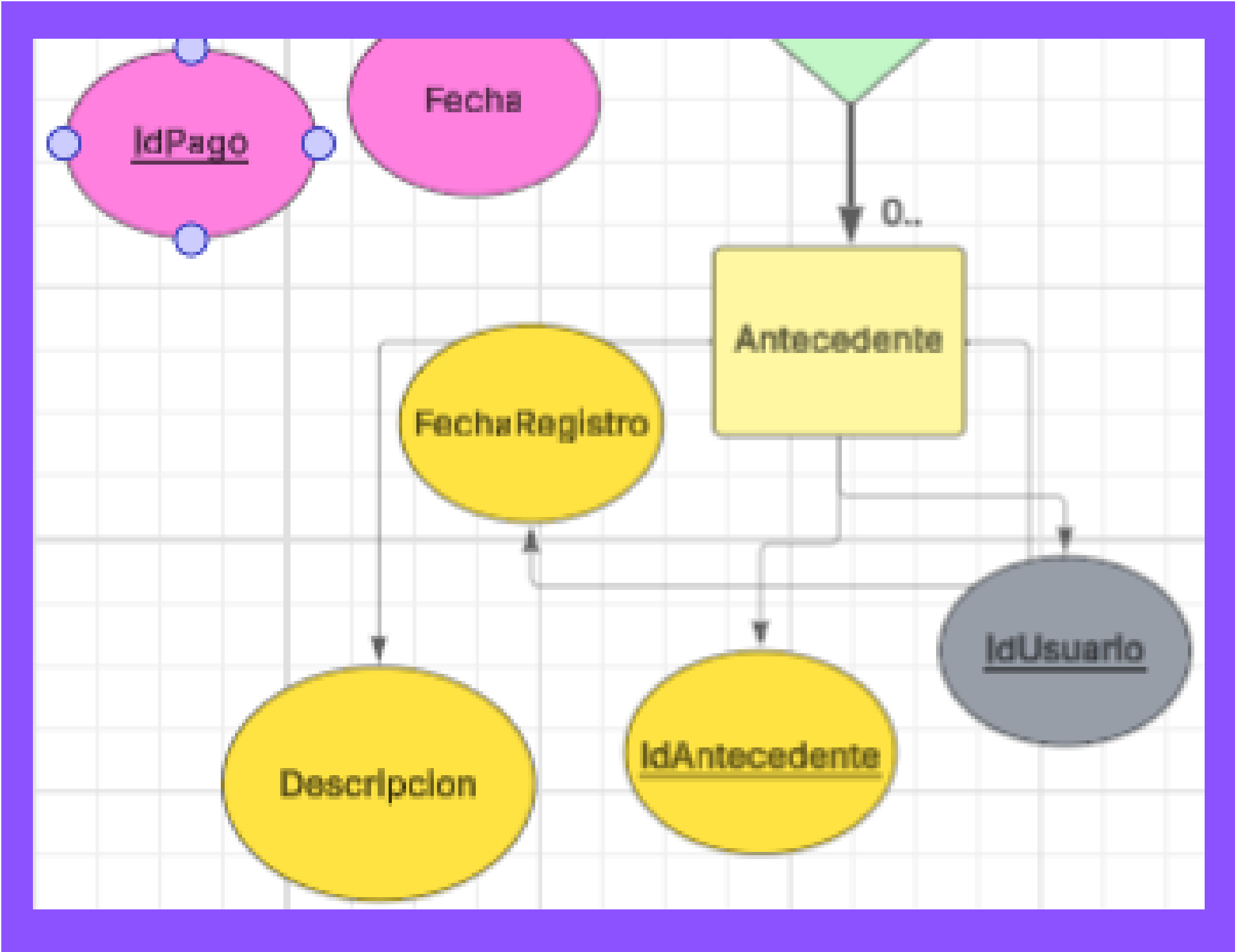
Se contiene la colección de entrevistas, con el sus atributos, su clave primaria y dos claves foraneas que son de las colecciones **USUARIO** y **EMPRESA**.

MODELO ENTIDAD RELACIÓN-ROL



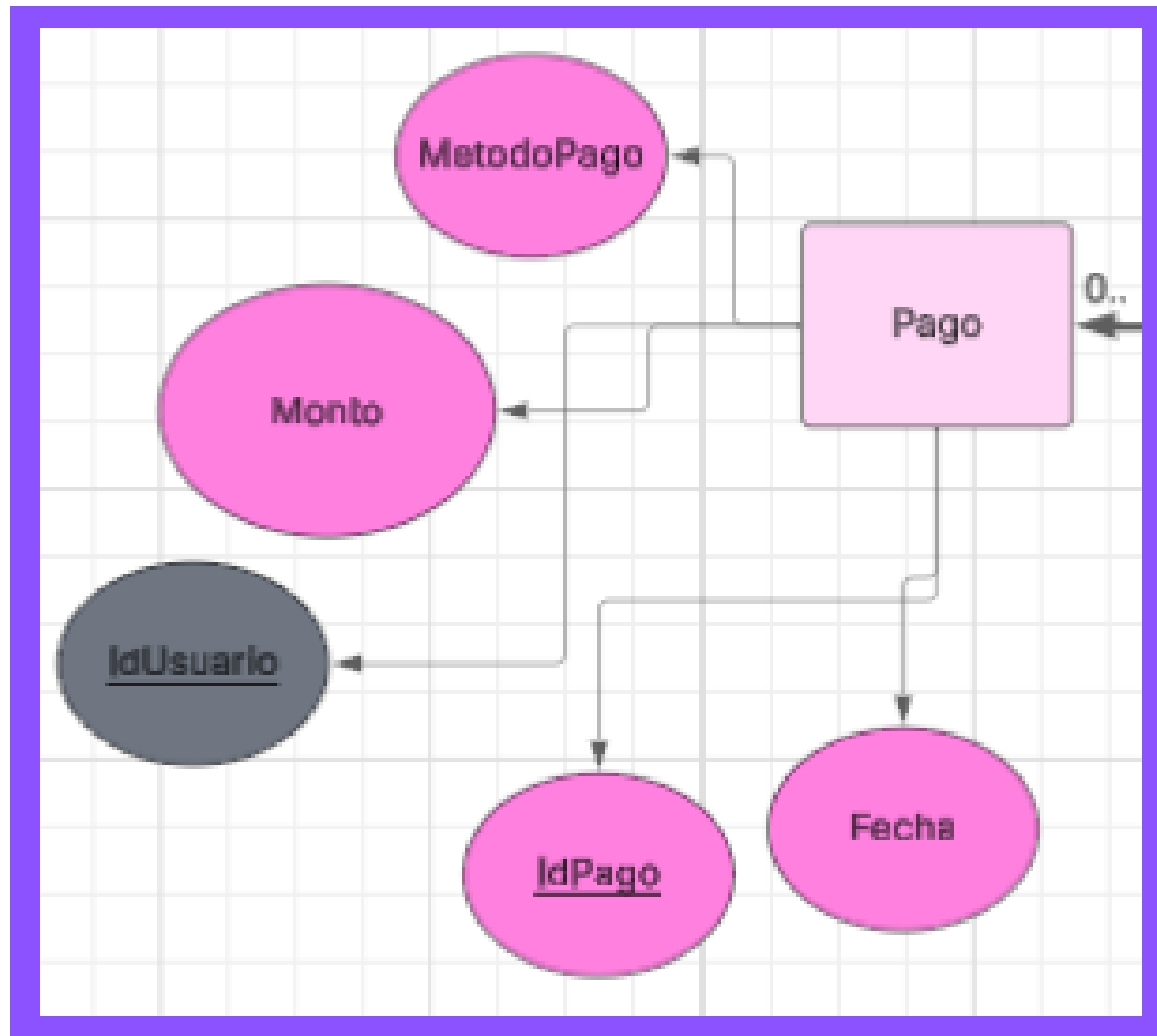
Se contiene la colección de **ROL**, con el sus atributos y la clave primaria de la misma.

MODELO ENTIDAD RELACIÓN-ROL



Se contiene la
colección de
antecedentes, con el sus
atributos, su clave
primaria y la clave
foránea de la colección
USUARIO.

MODELO ENTIDAD RELACIÓN-PAGOS



Se contiene la colección de **pago**, con el sus atributos, su clave primaria y la clave foránea de la colección **USUARIO**.

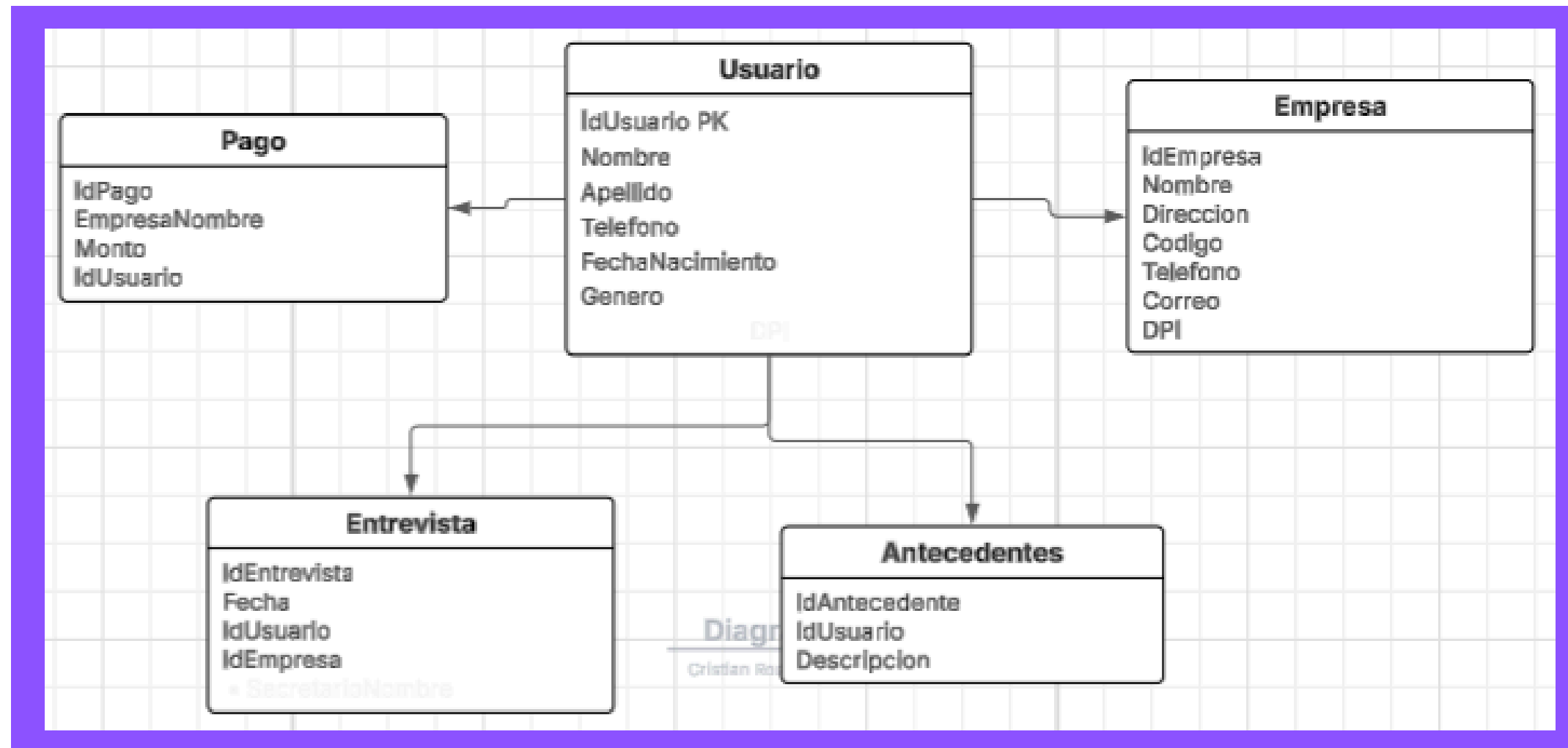
FASES DE NORMALIZACIÓN

PRIMERA FASE (1NF)



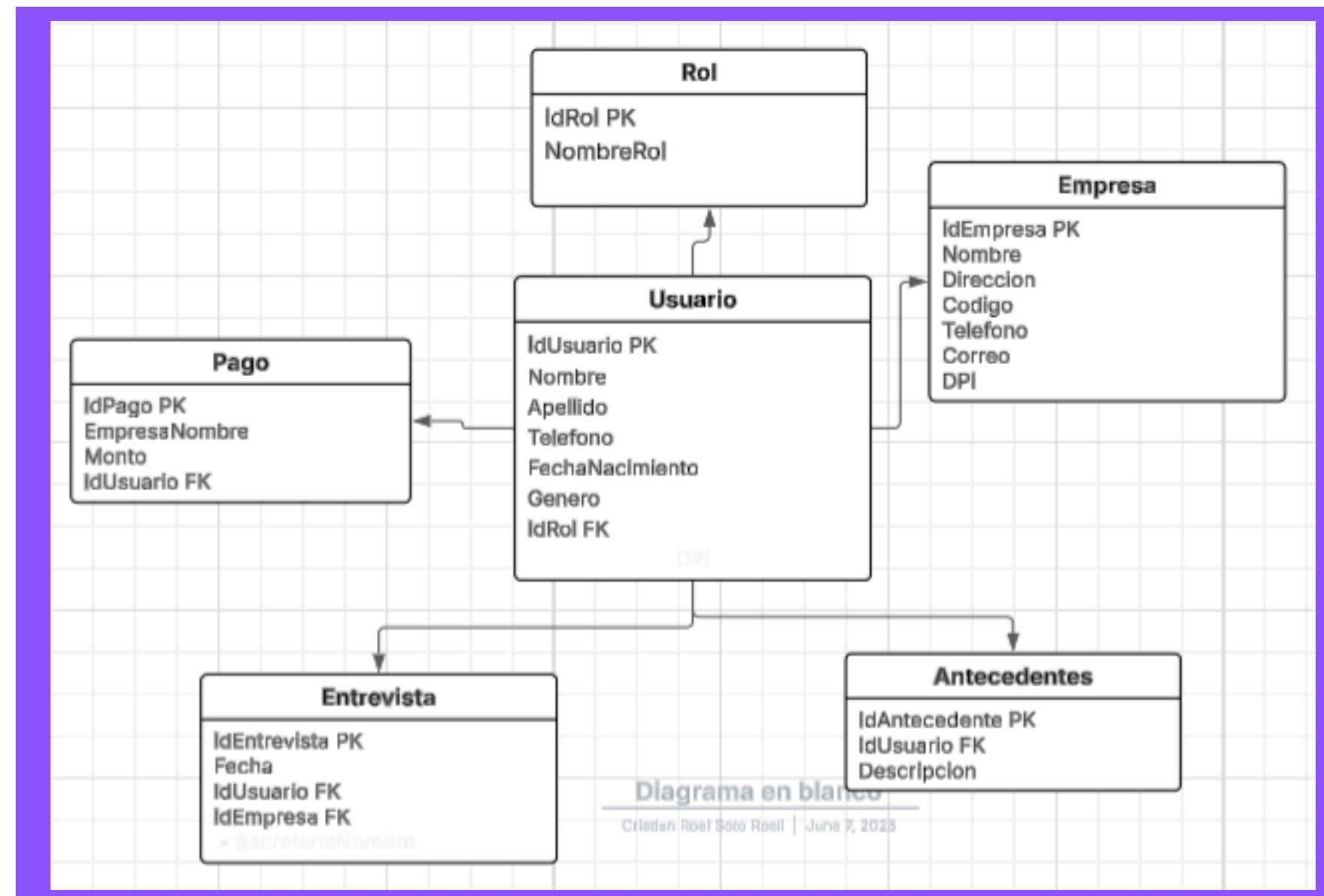
La primera fase de la normalización constituye las colecciones [administradores, pagos, secretarios, empresa, entrevista, candidatos] y se tomaron en cuenta tales colecciones para tener de forma clara los entes que participaran dentro del sistema. La **fase#1** se busca **eliminar datos repetitivos** y que cada colección tenga **valores atómicos** (información no redundante).

Segunda Fase (2NF)



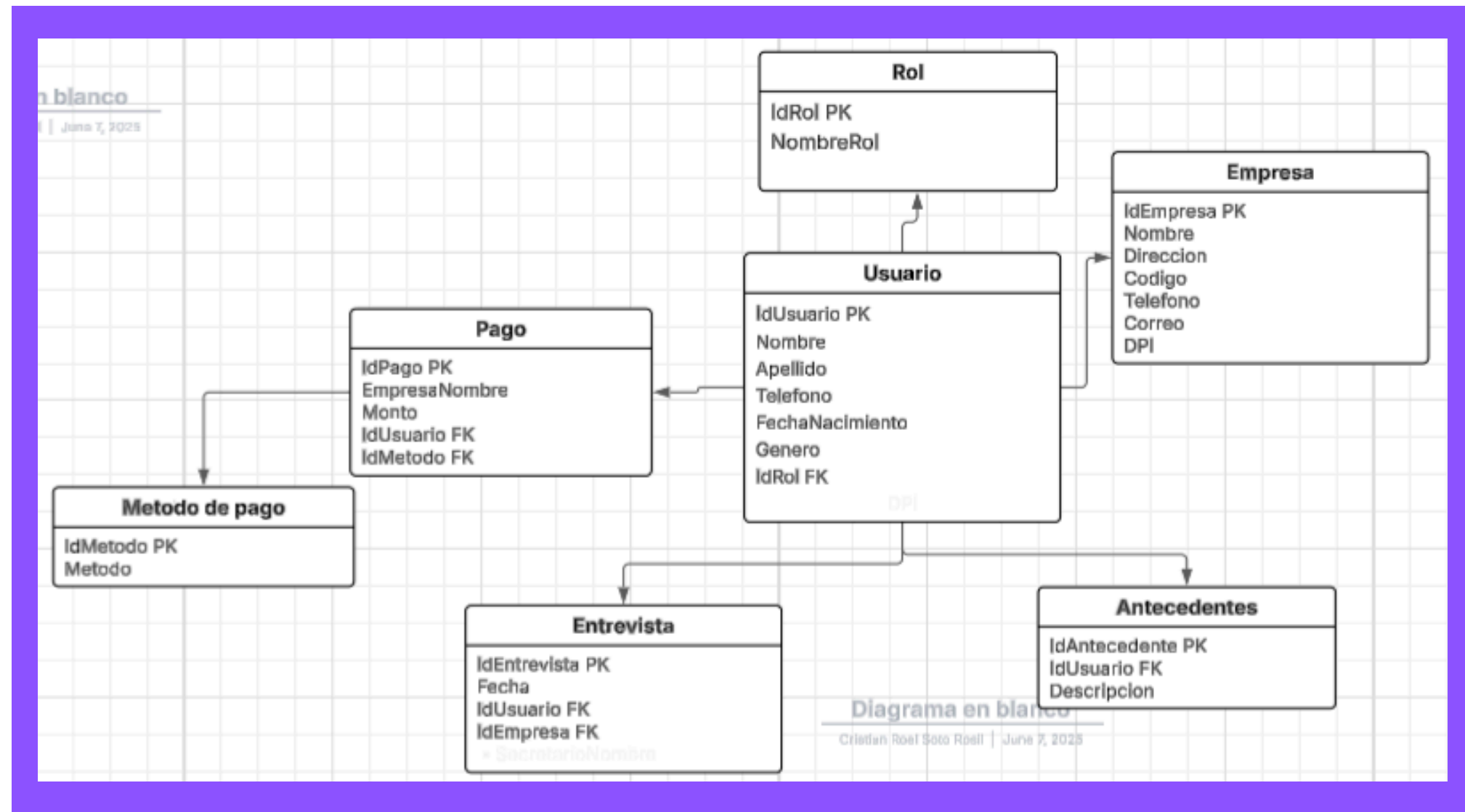
En la **segunda fase** se unificaron las colecciones de [administradores, secretarios, candidatos] en una sola colección que se llamará “**usuario**” para evitar la redundancia que pueda existir entre las mismas y ya **no se repetirán los campos del nombre de la empresa ni el del nombre de los candidatos**.

Tercera Fase (3NF)



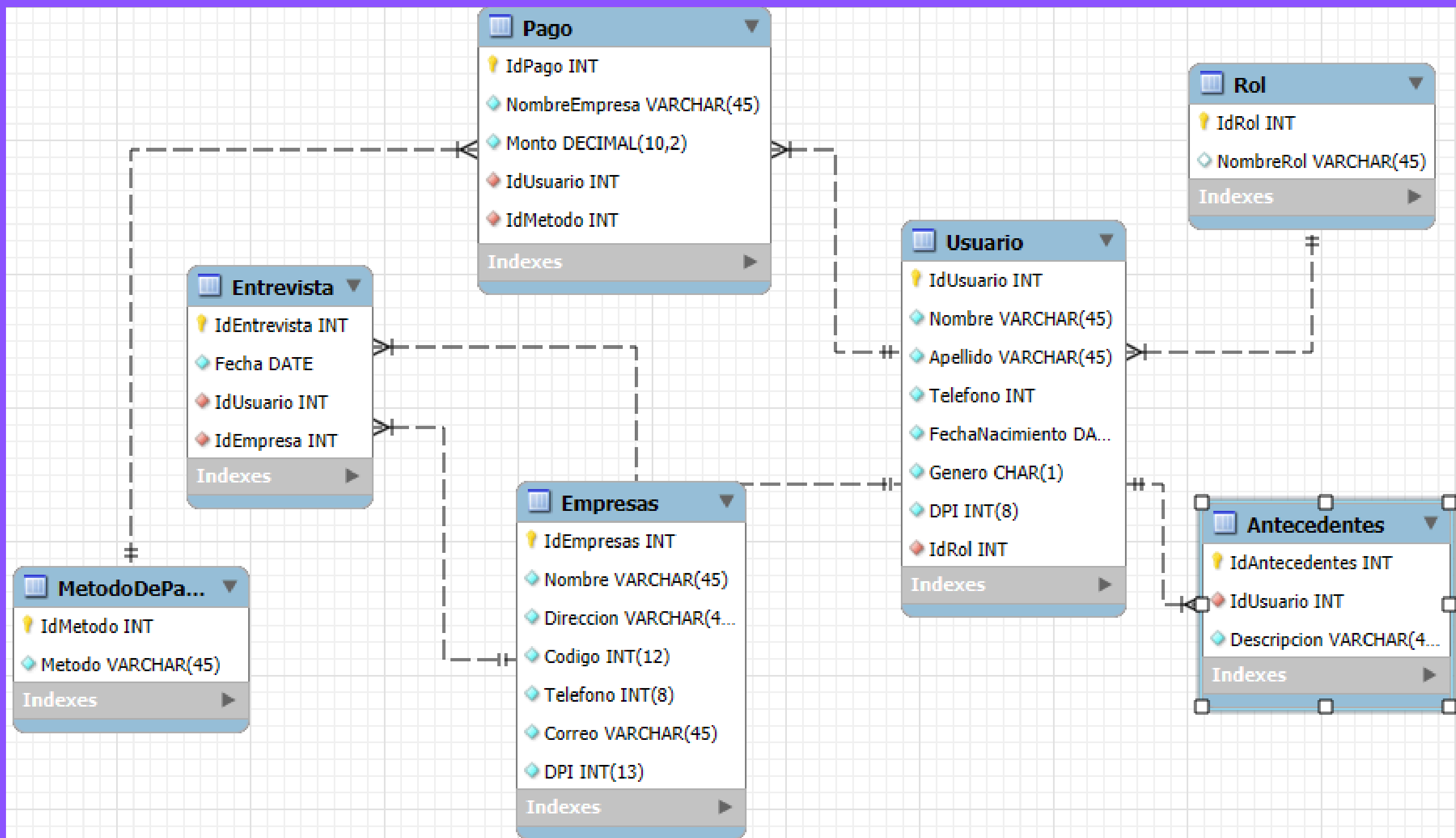
La tercera fase de normalización muestra como se contiene el **Rol** como un atributo de la colección de usuarios, se crea una colección independiente para el **ROL** buscando evitar repetición de texto en el mismo.

Cuarta Fase (4NF)



La cuarta fase de normalización se centra en crear una colección independiente de pago llamada “**métodos de pago**” que busca **evitar tener múltiples valores**. Cada colección contiene una **clave única** y se especifican las **claves foráneas**.

Modelo lógico



Relaciones clavé

Relación	Tipo	Detalle
Usuario → Rol	Muchos a uno	Muchos usuarios pueden tener un mismo rol.
Usuario → Pago	Uno a muchos	Un usuario puede tener múltiples pagos.
Pago → MetodoDePago	Muchos a uno	Varios pagos pueden usar el mismo método.
Usuario → Entrevista	Uno a muchos	Un usuario puede asistir a varias entrevistas.
Entrevista → Empresa	Muchos a uno	Muchas entrevistas pueden estar asociadas a una empresa.
Usuario → Antecedente	Uno a muchos	Un usuario puede tener varios antecedentes.

Menú Principal:

Backend

Backend: Análisis de la estructura en general

Las tres imágenes posteriores a la actual constituyen al **backend** de la plataforma hasta el momento.

Básicamente se trata de un **componente de React** llamado: **LoginFrame**

Se utiliza el lenguaje: **JavaScript**

Biblioteca usada: **React**

Que se logra?

Muestra un formulario modal para que el usuario introduzca su **email y contraseña**.

Envía esos **datos** a un **servidor backend** vía **fetch** a **http://localhost:5000/login**.

Si el **login** es **exitoso**, muestra un **alert** de **bienvenida**.

Si **falla**, muestra un **error**.

Backend: Lógica del Componente

```
1 import React, { useState } from 'react';
2
3 function LoginForm({ onClose }) {
4   const [email, setEmail] = useState('');
5   const [password, setPassword] = useState('');
6   const [error, setError] = useState(null);
7
8   const handleSubmit = async (e) => {
9     e.preventDefault();
10    setError(null);
11
12    try {
13      const response = await fetch('http://localhost:5000/login', {
14        method: 'POST',
15        headers: { 'Content-Type': 'application/json' },
16        body: JSON.stringify({ email, password }),
17      });
18
19      const data = await response.json();
20
21      if (response.ok) {
22        alert(`Login exitoso. Bienvenido, ${data.nombre}`);
23        onClose();
24      } else {
25        setError(data.error || 'Error desconocido');
26      }
27    } catch (err) {
28      setError('No se pudo conectar con el servidor');
29    }
30  };
31
32  return (
33    <div className="modal-overlay" onClick={onClose}>
34      <div className="modal-content" onClick={e => e.stopPropagation()}>
35        <h2>Iniciar Sesión</h2>
36        <form onSubmit={handleSubmit}>
```

```
37 <label>
38   Email:
39   <input
40     type="email"
41     placeholder="tuemail@ejemplo.com"
42     value={email}
43     onChange={(e) => setEmail(e.target.value)}
44     required
45   />
46 </label>
47 <label>
48   Contraseña:
49   <input
50     type="password"
51     placeholder="....."
52     value={password}
53     onChange={(e) => setPassword(e.target.value)}
54     required
55   />
56 </label>
57
58   {error && <p style={{ color: 'red' }}>{error}</p>}
59
60   <button type="submit" className="btn login">Entrar</button>
61 </form>
62 <button className="btn close-btn" onClick={onClose}>Cerrar</button>
63 </div>
64 </div>
65 );
66 }
67
68 export default LoginForm;
69
```

```
1 import React from 'react';
2
3 function RegisterForm({ onClose }) {
4   return (
5     <div className="modal-overlay" onClick={onClose}>
6       <div className="modal-content" onClick={e => e.stopPropagation()}>
7         <h2>Regístrate</h2>
8         <form>
9           <label>
10            Nombre completo:
11            <input type="text" placeholder="Tu nombre" required />
12          </label>
13          <label>
14            Email:
15            <input type="email" placeholder="tuemail@ejemplo.com" required />
16          </label>
17          <label>
18            Contraseña:
19            <input type="password" placeholder="....." required />
20          </label>
21          <button type="submit" className="btn register">Crear cuenta</button>
22        </form>
23        <button className="btn close-btn" onClick={onClose}>Cerrar</button>
24      </div>
25    </div>
26  );
27 }
28
29 export default RegisterForm;
30
```

Lógica del componente

```
import React, { useState } from 'react';
```

Importa React y el hook useState para manejar el estado del componente.

Hook de estado

```
const [email, setEmail] = useState('');  
const [password, setPassword] = useState('');  
const [error, setError] = useState(null);
```

Email, password: para guardar los datos ingresados en el formulario.

Error: para mostrar errores al usuario.

Modal de fondo

```
<div className="modal-overlay" onClick={onClose}>
```

Un click fuera del modal lo cierra.

Modal de contenido

```
<div className="modal-content" onClick={(e) => e.stopPropagation()}>
```

Evita que al hacer clic dentro del modal se cierre (detiene propagación).

Formulario

```
<form onSubmit={handleSubmit}>
```

Contiene dos campos: Email, Contraseña

Email

```
Email:
<input
  type="email"
  placeholder="tuemail@ejemplo.com"
  value={email}
  onChange={(e) => setEmail(e.target.value)}
  required
/>
```

Contraseña

```
Contraseña:
<input
  type="password"
  placeholder="....."
  value={password}
  onChange={(e) => setPassword(e.target.value)}
  required
/>
```

Conexión a la base de datos

```
backend > main.py > add_permit
1  from flask import Flask, request, jsonify
2  from pymongo import MongoClient, errors
3
4  # Inicializando la aplicación Flask
5  app = Flask(__name__)
6
7  # Conexión a la base de datos MongoDB
8  try:
9      client = MongoClient("mongodb://localhost:27017/")
10     db = client["ProyectoJo"]
11     user_collection = db["users"]
12     company_collection = db["companies"]
13     permit_collection = db["permits"]
14 except errors.ConnectionFailure as e:
15     print(f"No se pudo conectar a la BD: {e}")
16     db = None
17
18 # Helpers para convertir documentos Mongo en diccionarios
19 def user_to_dict(user):
20     return {
21         "code": user.get("codigo"),
22         "nameUser": user.get("nombre"),
23         "lastName": user.get("apellido"),
24         "phone": user.get("telefono"),
25         "birthDate": user.get("fecha_nacimiento"),
26         "gender": user.get("genero"),
27         "dpi": user.get("dpi"),
28         "rol": user.get("rol"),
29         "email": user.get("correo"),
30         "password": user.get("password"),
31         "laboralData": user.get("datos_laborales"),
32         "educationData": user.get("datos_educativos"),
```

```
19 def user_to_dict(user):
20     return {
21         "status": user.get("estado"),
22         "creationDate": user.get("fecha_creacion"),
23         "modificationDate": user.get("fecha_modificacion")
24     }
25
26 def company_to_dict(company):
27     return {
28         "code": company.get("codigo"),
29         "nameCompany": company.get("nombre"),
30         "direction": company.get("direccion"),
31         "phone": company.get("telefono"),
32         "nit": company.get("nit"),
33         "environment": company.get("entorno"),
34         "sector": company.get("sector"),
35         "email": company.get("correo"),
36         "date": company.get("fecha"),
37         "description": company.get("descripcion"),
38         "status": company.get("estado"),
39         "managerRH": company.get("encargadoRH"),
40         "state": company.get("estadoPerteneciente")
41     }
42
43 def permit_to_dict(permit):
44     return {
45         "code": permit.get("code"),
46         "administrator": permit.get("administrador"),
47         "recruiter": permit.get("reclutador"),
48         "employee": permit.get("empleado"),
49         "createDate": permit.get("fecha_creacion"),
50         "modificationDate": permit.get("fecha_modificacion")
51     }
```


Conexión a la base de datos

```
64
65 # Rutas
66 @app.route("/")
67 def home():
68     return "<h1>BACKEND DE RECLUTAMIENTO</h1>"
69
70 @app.route("/colecciones")
71 def obtener_colecciones():
72     if db is None:
73         return jsonify({"error": "No existe conexión a Mongo"}), 500
74     colecciones = db.list_collection_names()
75     return jsonify(colecciones), 200
76
77 # ----- USUARIOS -----
78
79 @app.route("/adduser", methods=["POST"])
80 def add_user():
81     try:
82         data = request.get_json()
83         required = ["code", "nameUser", "lastName", "phone", "birthDate", "gender", "dpi", "rol", "email", "password"]
84         if not all(k in data for k in required):
85             return jsonify({"error": "Faltan campos requeridos"}), 400
86         if user_collection.find_one({"dpi": data["dpi"]}):
87             return jsonify({"error": "El usuario ya existe"}), 400
88         user_collection.insert_one(data)
89         return jsonify({"mensaje": "Usuario agregado"}), 201
90     except Exception as e:
91         return jsonify({"error": str(e)}), 500
92
93 @app.route("/showuser/<string:dpi>", methods=["GET"])
94 def show_user(dpi):
```

```
95
96     if not user:
97         return jsonify({"error": "Usuario no encontrado"}), 404
98     return jsonify(user_to_dict(user)), 200
99
100 @app.route("/deleteuser/<string:dpi>", methods=["DELETE"])
101 def delete_user(dpi):
102     result = user_collection.delete_one({"dpi": dpi})
103     if result.deleted_count == 1:
104         return jsonify({"mensaje": "Usuario eliminado"}), 200
105     return jsonify({"error": "Usuario no encontrado"}), 404
106
107 @app.route("/updateuser/<string:dpi>", methods=["PUT"])
108 def update_user(dpi):
109     data = request.get_json()
110     result = user_collection.update_one({"dpi": dpi}, {"$set": data})
111     if result.matched_count:
112         return jsonify({"mensaje": "Usuario actualizado"}), 200
113     return jsonify({"error": "Usuario no encontrado"}), 404
114
115 # ----- EMPRESAS -----
116
117 @app.route("/addcompany", methods=["POST"])
118 def add_company():
119     try:
120         data = request.get_json()
121         required = ["code", "nameCompany", "direction", "phone", "nit", "environment", "sector", "email", "date"]
122         if not all(k in data for k in required):
123             return jsonify({"error": "Faltan campos requeridos"}), 400
124         if company_collection.find_one({"nit": data["nit"]}):
125             return jsonify({"error": "La empresa ya existe"}), 400
126         company_collection.insert_one(data)
```

Conexión a la base de datos

```
127     return jsonify({"mensaje": "Empresa agregada"}), 201
128 except Exception as e:
129     return jsonify({"error": str(e)}), 500
130
131 @app.route("/showcompany/<string:nit>", methods=["GET"])
132 def show_company(nit):
133     company = company_collection.find_one({"nit": nit})
134     if not company:
135         return jsonify({"error": "Empresa no encontrada"}), 404
136     return jsonify(company_to_dict(company)), 200
137
138 @app.route("/deletecompany/<string:nit>", methods=["DELETE"])
139 def delete_company(nit):
140     result = company_collection.delete_one({"nit": nit})
141     if result.deleted_count == 1:
142         return jsonify({"mensaje": "Empresa eliminada"}), 200
143     return jsonify({"error": "Empresa no encontrada"}), 404
144
145 @app.route("/updatecompany/<string:nit>", methods=["PUT"])
146 def update_company(nit):
147     data = request.get_json()
148     result = company_collection.update_one({"nit": nit}, {"$set": data})
149     if result.matched_count:
150         return jsonify({"mensaje": "Empresa actualizada"}), 200
151     return jsonify({"error": "Empresa no encontrada"}), 404
152
153 # ----- PERMISOS -----
154
155 @app.route("/addpermit", methods=["POST"])
156 def add_permit():
157     trv:
```

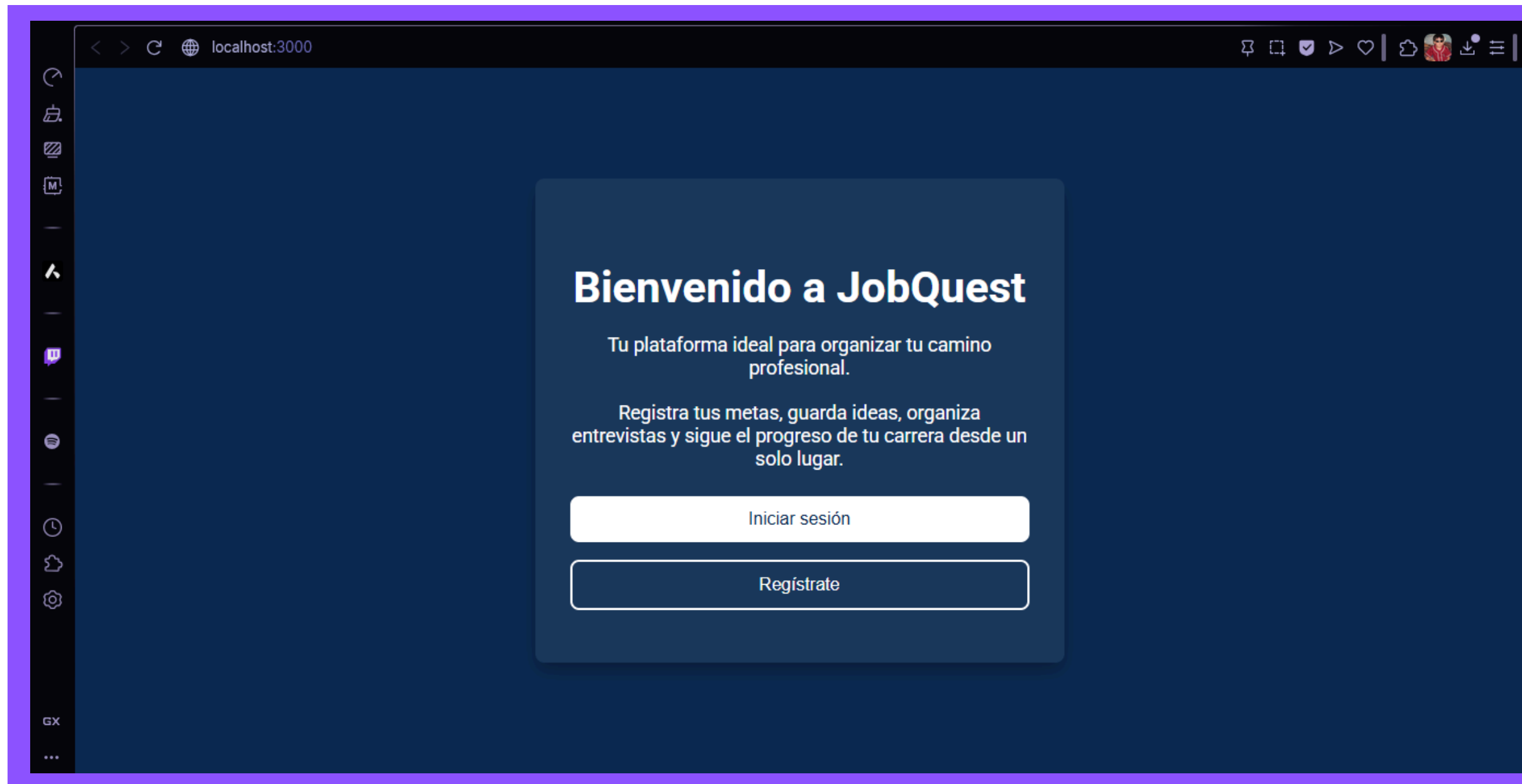
```
158     data = request.get_json()
159     required = ["code", "administrator", "recruiter", "employee", "createDate", "modificationDate"]
160     if not all(k in data for k in required):
161         return jsonify({"error": "Faltan campos requeridos"}), 400
162     if permit_collection.find_one({"code": data["code"]}):
163         return jsonify({"error": "El permiso ya existe"}), 400
164     permit_collection.insert_one(data)
165     return jsonify({"mensaje": "Permiso agregado"}), 201
166 except Exception as e:
167     return jsonify({"error": str(e)}), 500
168
169 @app.route("/showpermit/<string:code>", methods=["GET"])
170 def show_permit(code):
171     permit = permit_collection.find_one({"code": code})
172     if not permit:
173         return jsonify({"error": "Permiso no encontrado"}), 404
174     return jsonify(permit_to_dict(permit)), 200
175
176 @app.route("/deletepermit/<string:code>", methods=["DELETE"])
177 def delete_permit(code):
178     result = permit_collection.delete_one({"code": code})
179     if result.deleted_count == 1:
180         return jsonify({"mensaje": "Permiso eliminado"}), 200
181     return jsonify({"error": "Permiso no encontrado"}), 404
182
183 @app.route("/updatepermit/<string:code>", methods=["PUT"])
184 def update_permit(code):
185     data = request.get_json()
186     result = permit_collection.update_one({"code": code}, {"$set": data})
187     if result.matched_count:
188         return jsonify({"mensaje": "Permiso actualizado"}), 200
```


Inicialización de la plataforma

```
188         return jsonify({"mensaje": "Permiso actualizado"}), 200
189     return jsonify({"error": "Permiso no encontrado"}), 404
190
191     # Iniciar la app
192     if __name__ == "__main__":
193         app.run(debug=True)
194
```

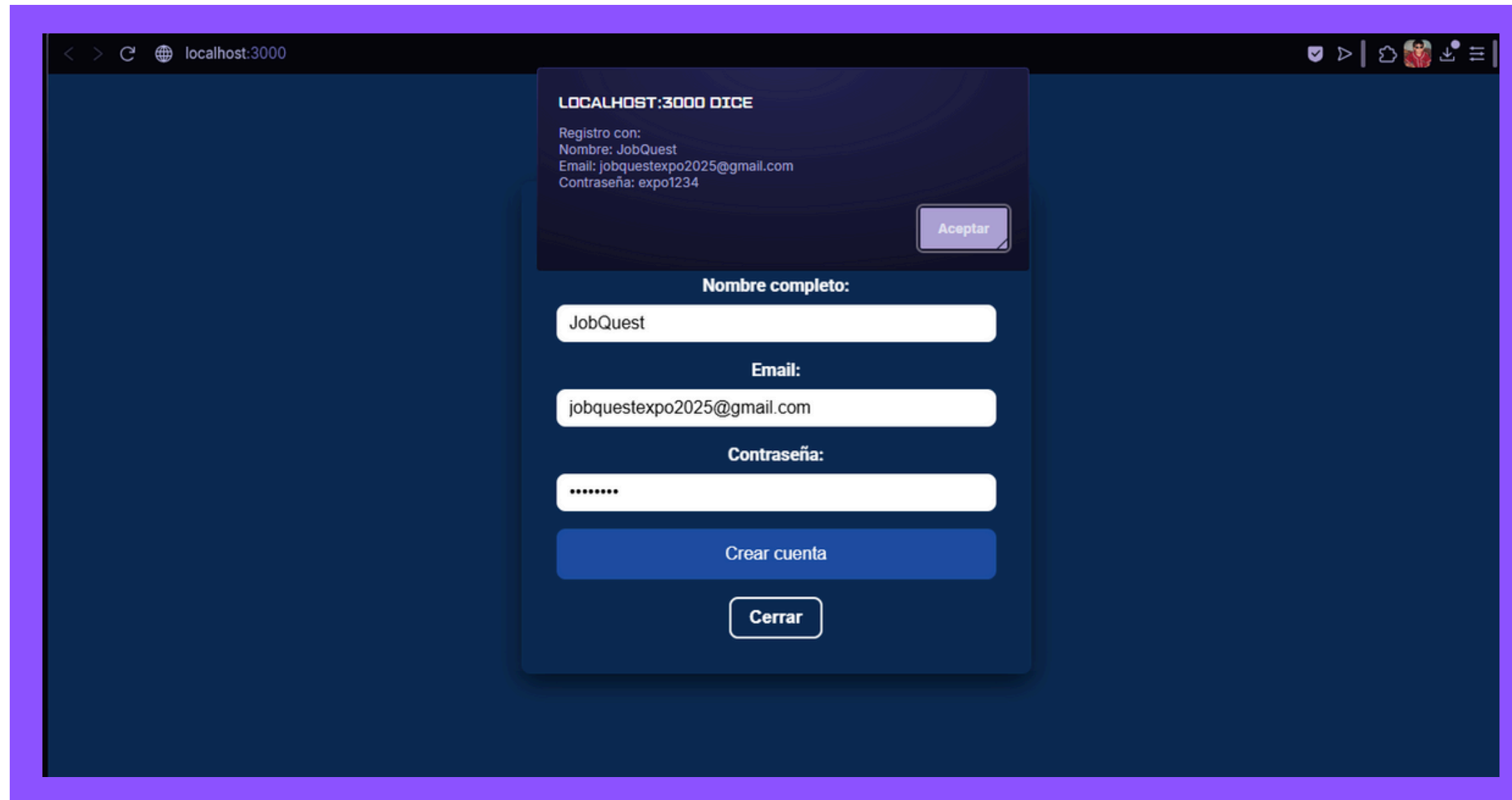
Menú Principal: Frontend

Frontend - Bienvenida



En la presente imagen se puede apreciar de manera clara un **mensaje de bienvenida**, la misma cuenta con: un **titulo claro** posicionado de manera estratégica para impactar en la percepción de todo aquel que lo llegue a apreciar, además de un texto que brinda una **pequeña introducción** de lo que es la plataforma en si. La parte inferior del frame esta ocupada por **dos botones**: “**Iniciar sesión**” y “**Registrarse**”.

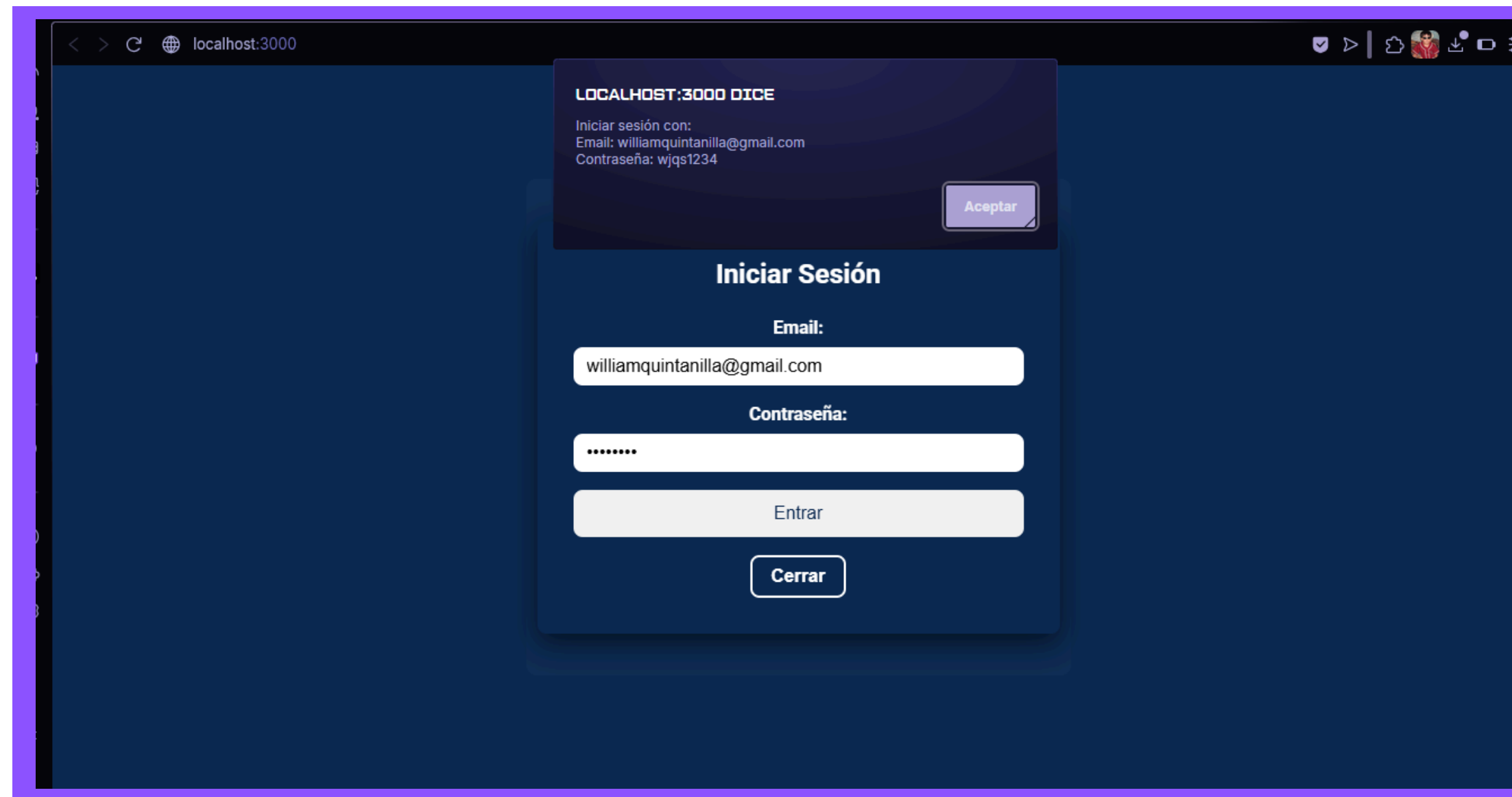
Frontend - Login (Usuario nuevo)



The screenshot displays a web browser window at localhost:3000. A dark blue modal window is centered on the screen, containing a registration form. At the top of the modal, a header reads 'LOCALHOST:3000 DICE'. Below this, the text 'Registro con:' is followed by the details: 'Nombre: JobQuest', 'Email: jobquestexpo2025@gmail.com', and 'Contraseña: expo1234'. An 'Aceptar' button is located to the right of these details. The main form area has three input fields: 'Nombre completo:' with the value 'JobQuest', 'Email:' with the value 'jobquestexpo2025@gmail.com', and 'Contraseña:' with masked characters '*****'. Below the password field is a blue 'Crear cuenta' button. At the bottom of the modal is a 'Cerrar' button. The background of the browser window is a dark blue gradient.

La imagen presente constituye al registro de un usuario nuevo, la diferencia que tiene con el frame que tendrá un usuario ya registrado es que en este, además de pedir una correo electrónico y una contraseña, pide un nombre. En la parte inferior podemos apreciar un botón que creara la cuenta si todos los parámetros ingresados son correctos.

Frontend - Login (Usuario ya registrado)



En la siguiente imagen se puede apreciar un **login**, este cuenta con un apartado para ingresar un **email** y una **contraseña**, además de varios **componentes** que, en conjunto, constituyen una de las primeras **interfaces de la plataforma**, podemos apreciar que los **colores seleccionados** forman parte de la **paleta** seleccionada en la **primera fase** del proyecto.