



LAURA HIGUERA ROMERO

ELENA BALLESTA GORDILLO

ERIC EMMANUEL RAMIREZ DUANCA

ALEJANDRO CASTRO RODRÍGUEZ

NEBRIMATCH

En el mundo de la programación, muchos estudiantes y desarrolladores se enfrentan a un mismo problema: aprender por su cuenta puede ser difícil cuando no tienes a alguien que te guíe o te explique ciertos temas que se te atragantan.

NEBRIMATCH nace como una plataforma web que conecta a programadores con otros programadores que desean enseñar y aprender lenguajes o formar grupos de trabajo. El objetivo es crear una comunidad de aprendizaje colaborativo, donde cualquier persona pueda encontrar a un “profe” o mentor especializado en el lenguaje o tecnología que necesite reforzar.

La idea principal es que cada usuario pueda registrarse, indicar su nivel de experiencia, las áreas en las que quiere mejorar y las tecnologías que domina. A partir de ahí, el sistema recomienda perfiles compatibles para que ambos puedan ponerse en contacto y empezar a aprender juntos, además **NEBRIMATCH** cuenta con comunidades para formar grupos de trabajos académicos.

El **producto mínimo viable (PMV)** de **NEBRIMATCH** es una versión inicial de la web que incluye las funciones esenciales para comprobar que la idea funciona y que los usuarios realmente encuentran valor en ella.

Este primer prototipo se centrará en tres pilares principales:

1. Registro y perfiles de usuario:

Los usuarios podrán crear una cuenta y completar su perfil con información básica: nombre, nivel de experiencia, lenguajes que dominan y los que quieren aprender.

2. Sistema de emparejamiento básico:

A través de un formulario o algoritmo sencillo, la plataforma mostrará coincidencias entre personas que buscan aprender y otras que ofrecen enseñar.

Por ejemplo, si un usuario quiere aprender *JavaScript* y otro lo domina, el sistema los conecta.

3. Comunicación inicial entre usuarios:

Habrá una forma básica de contacto, un chat en línea para que puedan coordinar las clases o mentorías.

A partir del PMV, **NEBRIMATCH** podría evolucionar hacia una plataforma más completa, con características como:

- Sistema de valoración o reputación para los mentores.
- Filtros avanzados por nivel, horario o idioma.

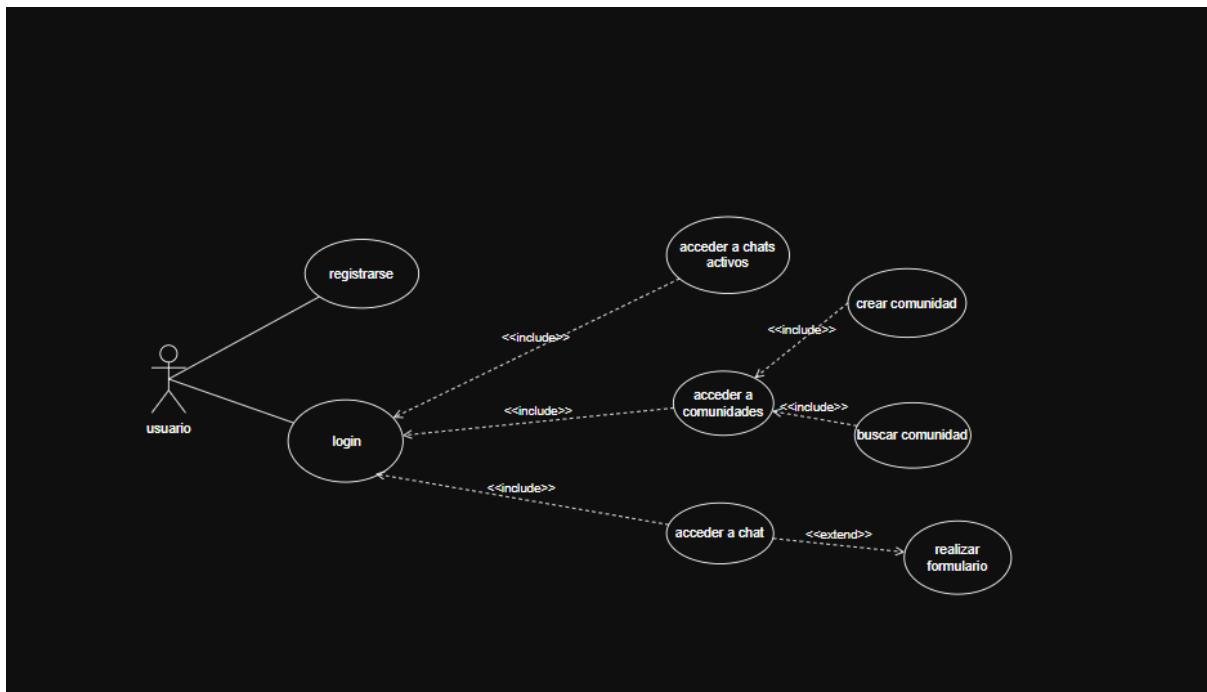
- Integración de videollamadas o chat en tiempo real.
- Un apartado de comunidad donde los usuarios comparten proyectos o recursos.

En resumen, **NEBRIMATCH** busca crear un espacio donde aprender programación sea más humano, directo y colaborativo, rompiendo la barrera entre quienes saben y quienes quieren aprender.

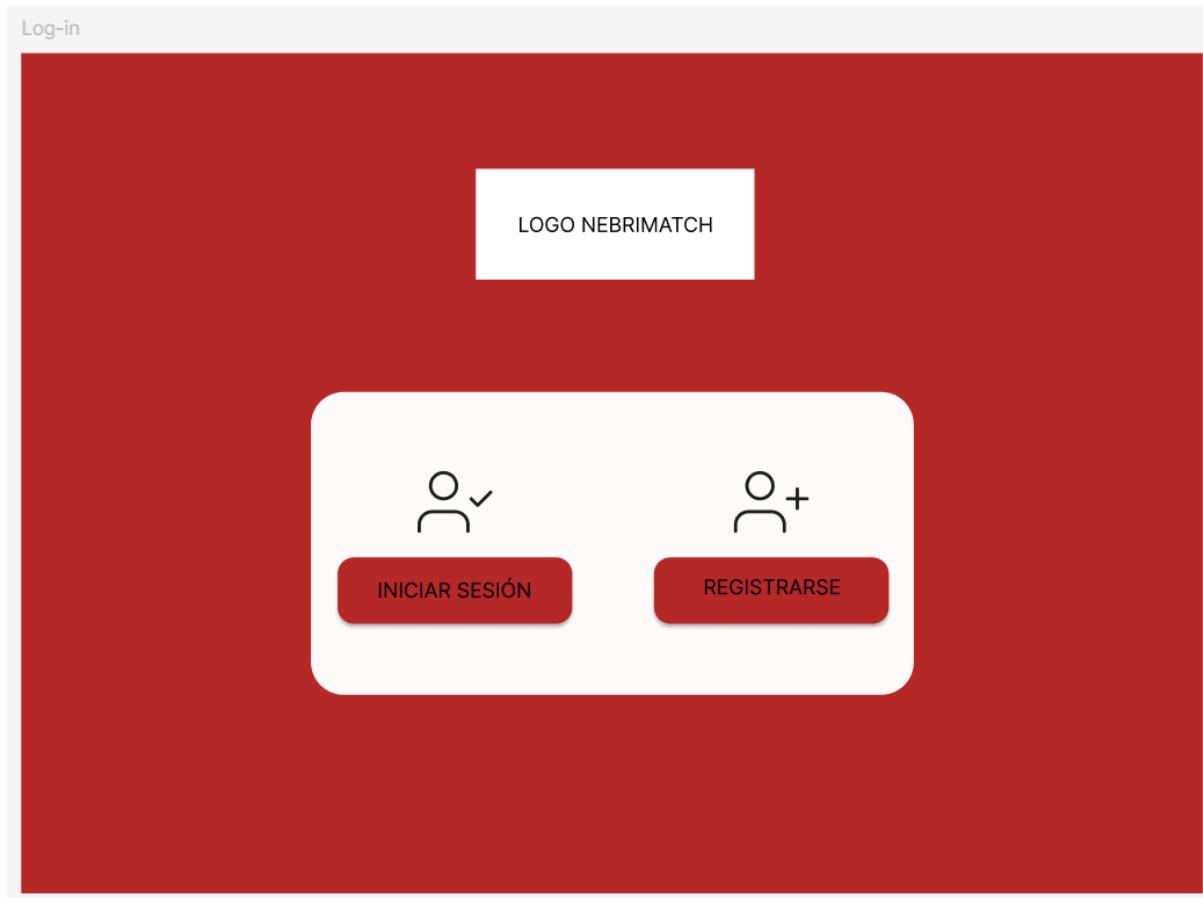
Análisis de requisitos funcionales:

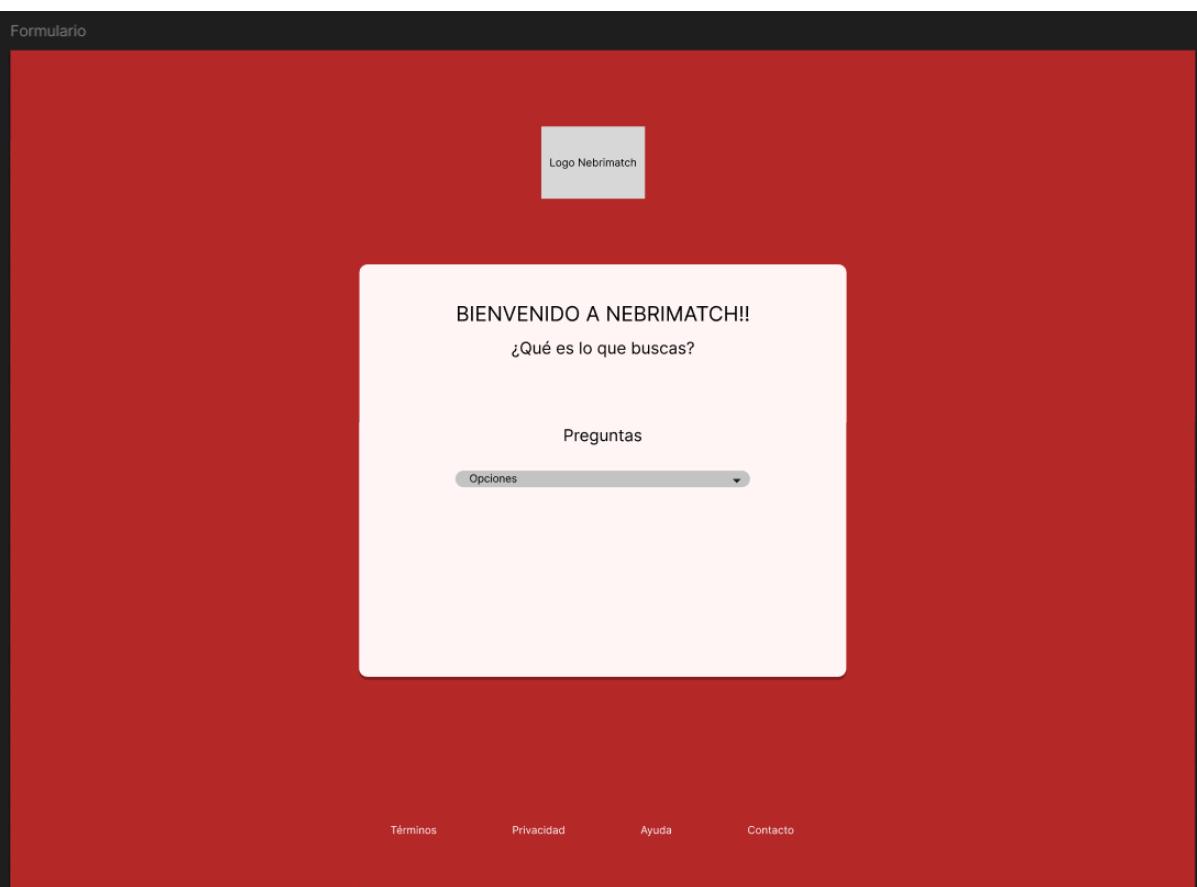
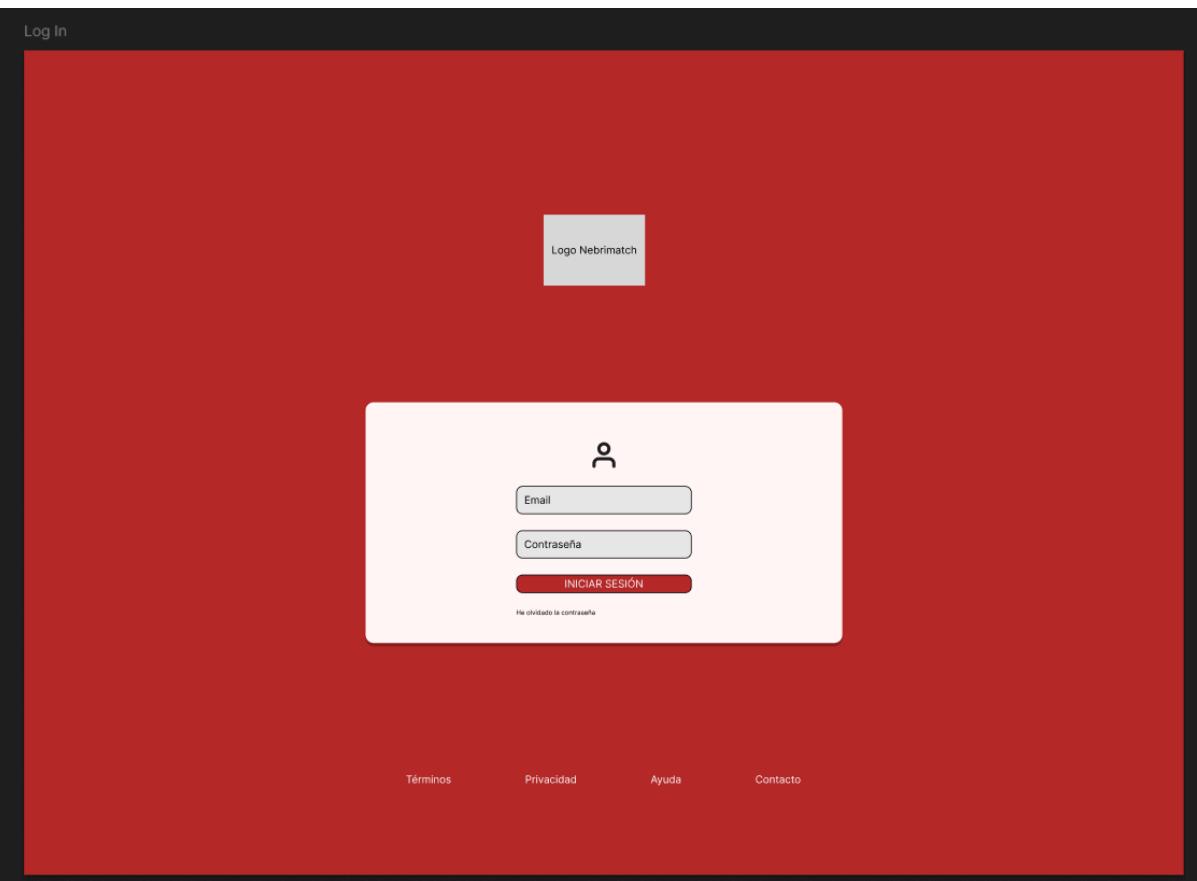
- a. Sistema de usuarios
- b. Chat individual entre usuarios
- c. Comunidades/grupos de trabajo
- d. Sistema de conexiones/matching

Casos de uso:



Prototipado en Figma:





Comunidad

Logo Nebrimatch Hinted search text

COMUNIDADES

Únete a una comunidad

Descripción

BUSCAR

Crea tu comunidad

Descripción

CREAR

Términos Privacidad Ayuda Contacto

Chats

Logo nebrimatch Hinted search text

Personas que quizá conozcas

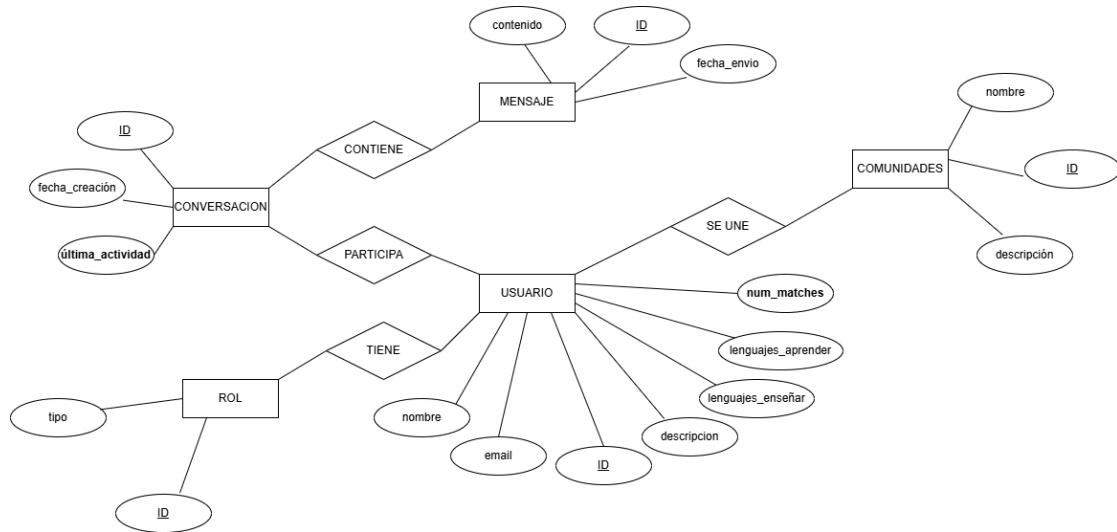
María Perez Álvarez Laura Rubio Perez Elena Ramírez Bou Eric Ballesta Castro María Perez Álvarez

Grupos que pueden interesarle

JavaScript Python Java SQL React

Términos Privacidad Ayuda Contacto

Bases de datos:



Para la base de datos, lo haremos de forma nativa en un proyecto node, instalando la librería *mysql2*.

Contamos con las siguientes tablas:

- **usuario**
- **rol_usuario**: ya que los usuarios pueden tener dos roles, rol estudiante o rol profesor dependiendo de si están enseñando un lenguaje o lo están aprendiendo.
- **conversaciones**
- **mensajes**
- **comunidades**
- **comunidades_miembros**

Haremos uso de una base de datos no relacional (MongoDB) para demostrar los conocimientos adquiridos en la recopilación de datos del Login de NebriMatch.

Estructuración de carpetas:

La estructuración de carpetas que hemos planteado para el proyecto es la siguiente:

- **Frontend**: donde crearemos un proyecto react con las carpetas necesarias según el número de componentes que vamos a incluir.
- **Backend**: dentro de la estructuración de carpetas del backend, tenemos la carpeta config para todo lo relacionado con la base de datos ([database.js](#)). En este archivo se llama a la librería express de npm instalada, se crea el pool, las tablas y un array incluyendo todas estas para ejecutar solo de una vez en vez de ir yendo una por una:

```
// ====== ARRAY DE TABLAS ======
```

```
const tablas = [
  usuario,
  rol_usuario,
  conversaciones,
  mensajes,
  comunidades,
  comunidad_miembros
];
```

```
// ===== FUNCIÓN CREAR TABLAS =====

const crearTablas = async () => {
  try {
    const connection = await pool.getConnection();
    console.log("Creando tablas...");

    for (const tabla of tablas) {
      await connection.query(tabla);
    }

    console.log(" Todas las tablas creadas exitosamente");

    // para liberar conexiones:
    connection.release();
  } catch (error) {
    console.error(" Error al crear tablas:", error);
  }
};


```

Volviendo a la carpeta backend, tenemos el archivo [server.js](#) para todo el tema del enrutado y sentencias sql. Además de las librerías express y cors (esta última para poder interactuar en un futuro con el front de manera segura) tenemos que incluir el archivo [database.js](#) que hemos creado en la carpeta config que hemos mencionado anteriormente:

```
backend > JS server.js > ...
1  const express = require("express");
2  const cors = require("cors");
3  const { crearTablas } = require("./config/database");
4
5  const app = express();
6  const port = 4004;
7
8  app.use(cors());
9  app.use(express.json());
10
11  crearTablas();
12
```

En aspectos de enrutado, contamos con una ruta principal “/”; pasamos a archivo json (res.json) para enviar las respuestas al frontend futuro de la siguiente manera ⇒

```
app.get("/", (req, res) => {
  res.json({
    mensaje: "NebriMatch funcionando",
    version: "1.0.0",
    endpoints: {
      usuarios: "/api/usuarios",
      perfilUsuario: "/api/usuarios/:usuario",
      chats: "/api/usuarios/:usuario/chats",
      paraTi: "/api/para-ti",
      ayuda: "/api/ayuda",
    },
  });
});
```

En todas las funciones async que vamos a realizar a continuación, debemos llamar a la [database.js](#) que hemos creado en la carpeta config:

```
app.get("/api/usuarios", async (req, res) => {
  const { pool } = require("./config/database");
  try {
    const [usuarios] = await pool.query("SELECT * FROM usuario");
    //Envía los datos como respuesta JSON al cliente (React)=>
    res.json(usuarios);
    res.json(usuarios);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```