TÉCNICO SUPERIOR EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

(IFCS02)

MÓDULO: PROYECTO DE DESARROLLO DE APLICACIONES MULTIPLATAFORMA (0492)

CURSO: 2024-2025

CONVOCATORIA: Ordinaria

RED SOCIAL DE DEBATE POLÍTICO: ÁGORA

(BORRADOR)

MODALIDAD DE PROYECTO: B

PROFESOR-COORDINADOR: Manuel Pablo Cantizano González

APELLIDOS Y NOMBRE: Carlos Sánchez Pellón

DNI: 50248243M

APELLIDOS Y NOMBRE: Miguel Ceada Fernández

DNI: no aprueba

Madrid, 07 de junio de 2025

Resumen del proyecto

Ágora es una red social especializada en el debate político, diseñada para acumular y centralizar las discusiones sobre actualidad y opinión pública. El objetivo de Ágora es ofrecer un entorno dedicado donde usuarios interesados en la política puedan acceder en tiempo real a un feed de noticias relacionadas con el tema, publicar sus propias opiniones y participar en hilos de discusión. Gracias a su sistema de autenticación, perfiles editables y sección de comentarios vinculados a cada publicación, Ágora facilita el diálogo constructivo y la transparencia informativa. De este modo, los participantes podrán seguir de cerca los eventos políticos, compartir análisis y mejorar la calidad del debate colectivo.

Palabras clave

Red social
Política
Discusión
Noticias
Publicaciones
Comentarios
Igualdad

1. DESCRIPCIÓN DEL PROYECTO

Ágora es una red social de tipo vertical enfocada en la política. En esta aplicación para móviles los usuarios pueden ponerse al día de la actualidad política, a la vez que comparten y debaten sus opiniones entre ellos. Con esta aplicación se pretende que aquellas personas con ferviente interés por la política, geopolítica, economía de Estados, relaciones internacionales, etc, se reúnan en una misma aplicación desarrollada específicamente para sus intereses.

La aplicación cuenta con dos secciones principales: Noticias y Foro. La primera contiene las últimas noticias relacionadas con la política, pudiendo filtrarse mediante una interfaz con diversas opciones. En la sección del Foro los usuarios pueden compartir y discutir sus opiniones sin ningún tipo de censura. No son completamente independientes, sino que las noticias de la primera incluyen la opción de crear un comentario referenciandolas.

A nivel técnico, nuestra aplicación se ha diseñado siguiendo el modelo de tres capas, o modelo Vista-Controlador, por lo que incluye un frontend, un backend y una base de datos, elementos detallados más adelante. Adicionalmente, ciertas funcionalidades de la aplicación se basan en el modelo CRUD. Los usuarios pueden crear, leer, actualizar y eliminar usuarios, publicaciones, perfiles, etc.

2. JUSTIFICACIÓN DEL PROYECTO

Necesidad de un espacio especializado

Las redes sociales generalistas (Facebook, X (antes Twitter), Instagram, etc.) concentran un volumen masivo de contenidos de muy diversos tipos, en los que los debates políticos suelen verse diluidos por dinámicas de viralidad ajenas al fondo del discurso. Esta dispersión reduce la calidad y profundidad del intercambio de ideas, favorece la polarización y dificulta la formación de opinión basada en datos y argumentos sólidos. Ágora nace para cubrir esa carencia: un entorno dedicado al análisis y la discusión política, donde el contenido y la estructura favorezcan la continuidad del tema principal.

Fomento del diálogo constructivo

Al proporcionar herramientas de moderación y un sistema de hilos de discusión jerarquizados, la plataforma facilita que los usuarios centren sus aportaciones en argumentos y evidencias, en lugar de en dinámicas de confrontación espontánea o desinformación. Esto contribuye a elevar el nivel del debate público, poniendo el énfasis en la calidad de las contribuciones más que en su viralidad.

Énfasis en la igualdad de las opiniones

Con la ausencia de censura directa o indirectamente impuesta por algoritmos de visualización, verificadores de contenido, moderación excesiva… Ágora evita convertirse en un instrumento de manipulación y promueve la igualdad de validez de todas las opiniones.

Viabilidad técnica y escalabilidad

El uso de tecnologías maduras como Flutter, Node.js y PostgreSQL facilita un desarrollo ágil, un mantenimiento sencillo y una arquitectura escalable. Esto reduce riesgos de implantación y permite futuras ampliaciones (mensajería privada, encuestas, geolocalización) sin reestructuraciones drásticas.

Alineación con objetivos de la titulación DAM

Desde la perspectiva académica, este proyecto integra los ámbitos de desarrollo de aplicaciones móviles, diseño de back-end y gestión de bases de datos, así

como la integración de servicios externos (APIs). Permite demostrar competencias de análisis de requisitos, diseño de arquitectura, implementación y pruebas, cumpliendo con los objetivos formativos del ciclo de grado medio de Desarrollo de Aplicaciones Multiplataforma.

3. OBJETIVOS

3.1. *Objetivos funcionales*

Registro y autenticación de usuarios

- Permitir el alta de nuevos usuarios mediante correo electrónico y contraseña.
- Ofrecer inicio de sesión seguro.

Gestión de perfiles

- Mostrar y editar datos de perfil (nombre de usuario, país…).
- Listar el historial de publicaciones y comentarios de cada usuario.

Feed de noticias en tiempo real

- Conectar a APIs de noticias políticas para obtener actualizaciones continuas.
- Mostrar titulares, resúmenes y enlaces a fuentes originales.
- Ofrecer filtros para las noticias (país, idioma, tema específico…)

Publicación de contenido

- Posibilitar la creación de publicaciones de texto libre.
- Vincular artículos de noticias directamente a una publicación.

Sistema de comentarios y hilos de discusión

- Permitir comentar en cualquier publicación.
- Organizar los comentarios en hilos anidados para facilitar el debate.

Foro de publicaciones

- Mostrar las publicaciones de los distintos usuarios ordenadas por fecha y hora de publicación.
- Filtrar las publicaciones mostradas por distintas características y/o categorías.

3.2. *Objetivos técnicos*

Arquitectura escalable y modular

- Diseñar el backend con una estructura de capas (controladores, servicios, repositorios) para facilitar ampliaciones y mantenimiento.

- Implementar módulos independientes para autenticación, gestión de noticias y gestión de publicaciones.

Rendimiento y eficiencia

- Optimizar consultas a PostgreSQL mediante índices adecuados y sentencias parametrizadas.
- Configurar un sistema de caché para contenidos de alto acceso (por ejemplo, noticias más recientes).

Seguridad y protección de datos

- Utilizar JWT con expiración controlada para la autenticación y autorización de usuarios.
- Encriptar contraseñas con bcrypt y manejar adecuadamente las variables de entorno (dotenv) para claves y credenciales.

Integración de API de noticias

- Obtener noticias en formato JSON de una API web a través de solicitudes http.

Desarrollo multiplataforma y consistencia UI

- Utilizar Flutter para compilar una única base de código que funcione en Android e iOS.
- Definir estilos y componentes reutilizables para mantener coherencia visual y optimizar tiempos de desarrollo.

Pruebas y calidad de software

- Escribir tests unitarios (Jest/Mocha para backend, Flutter Test para frontend) cubriendo al menos el 70 % del código crítico.
- Implementar pruebas de integración en endpoints clave (autenticación, feed, comentarios).

4. RECURSOS

En el desarrollo de Ágora se han utilizado los siguientes recursos:

4.1. *Hardware*

Ordenador de desarrollo: con al menos 8 GB de RAM, sistema operativo compatible con Android/iOS SDK.
Conexión a Internet estable: para pruebas en tiempo real, consumo de APIs y colaboración online.
Dispositivo móvil Android: para pruebas de versiones.

Para la presentación de Ágora es necesario el empleo de los siguientes elementos:

- **Dispositivo móvil Android**: instalación y uso de la aplicación móvil.
- **Portátil**: despliegue del backend y la base de datos.
- **Cable USB - Tipo C / Dispositivo móvil**: conexión por red de ambos elementos anteriores.

4.2. *Software*

Frontend móvil

- **Tecnología**: Flutter
- **IDE**: Visual Studio Code
- **Lenguaje**: Dart
- **Dispositivos de prueba**: emuladores Android

Backend

- **Tecnología**: Node.js con Express.
- **IDE**: Nano, Visual Studio Code
- **Lenguaje**: JavaScript
- **Librerías y herramientas**: JWT para autenticación, bcrypt para encriptado, dotenv para configuración
- **Pruebas**: Git Bash

Base de datos

- **Sistema de gestión**: PostgreSQL
- **Lenguaje**: SQL

APIs externas

- **Fuentes de noticias políticas**: Real-Time News Data

Recursos software adicionales

- **Control de versiones**: Git + GitHub
- **Pruebas**: Emulador de Android Studio.
- **Diseño de sistema**: Lucidchart, Draw.io

4.3. *Humanos*

Desarrolladores principales: encargados del análisis, diseño, programación y pruebas de todo el sistema (frontend y backend).

Usuarios de prueba: colaboradores que ayudarán a realizar pruebas de usabilidad y funcionalidad en las fases de validación.

5. DESARROLLO DEL PROYECTO

5.1. *Diseño*

El diseño de Ágora consta de tres secciones generales: Frontend, Backend y Base de Datos.

5.1.1 *Frontend*

El diseño del frontend de Ágora está enfocado en ofrecer una experiencia de usuario clara, directa y centrada en el contenido político. A continuación, se detallan las principales pantallas, componentes y principios de diseño aplicados:

Principios de diseño

- **Simplicidad visual**: se prioriza la legibilidad del texto, los contrastes claros y una jerarquía visual basada en titulares, subtítulos y cuerpo.
- **Facilidad de navegación**: los usuarios deben acceder rápidamente a noticias, publicaciones y perfiles con la menor cantidad de pasos.
- **Coherencia**: se utilizan estilos y componentes reutilizables, manteniendo una identidad visual homogénea.
- **Diseño responsive**: adaptable a diferentes resoluciones y formatos de pantalla móviles.

Pantallas principales

- **Inicio / Feed de publicaciones**
    - Scroll de los títulos de las publicaciones.
    - Opción de actualizar pantalla.

- **Publicación**
    - Texto completo de la publicación.
    - Sección de comentarios anidados con opción de comentar.
    - Cada tarjeta de comentario muestra:
        - Nombre del autor
        - Texto
- **Feed de noticias**
    - Lista vertical de noticias políticas.
    - Cada tarjeta de noticia muestra:
        - Titular
        - Fragmento
        - Fuente
        - Fecha/hora
        - Botón para publicar
    - Opción de actualizar pantalla.
    - Filtro por parámetros y buscador por palabras.

Nueva publicación

- Editor simple de texto con:
    - Campo para escribir opinión
    - Botón para publicar
- Validación de texto mínimo y máximo

Perfil de usuario

- Visualización del perfil con:
    - Nombre de usuario
    - Intereses políticos
    - Listado de publicaciones hechas
- Botón para editar el perfil (en perfiles propios).

Inicio de sesión / Registro

- Imagen
- Formulario con validación para:
    - Nombre de usuario (registro)

    - Correo electrónico
    - Contraseña
    - Confirmación de contraseña (registro)
    - Selector de país con country_code_picker (registro)
- Acceso directo al registro (inicio de sesión) / inicio de sesión (registro).
- Botón de confirmación

Botón flotante común

- Botones de acceso a:
    - Mi perfil
    - Preferencias (filtros, orden)
    - Publicar
    - Cerrar sesión

5.1.2 *Backend*

El backend es el encargado de gestionar las peticiones que le haga el backend y establecer conexión entre este y la base de datos. Gestiona las peticiones mediante un token que se entrega al usuario al iniciar sesión. Cada petición que el frontend hace al backend, irá con ese token dentro, lo que permite que el backend compruebe la autenticidad del token, habilitando o no la operación deseada.

Está desarrollado siguiendo el modelo Vista-Controlador. Los ficheros se guardan en distintas carpetas siguiendo la lógica de dicho modelo:

- server.js
- /controllers
    - userController.js
    - postController.js
    - newsController.js
- /routes
    - userRoutes.js
    - postRoutes.js
    - newsRoutes.js

- /config
    - db.js
- /middlewares
    - auth.js

El backend consta de un fichero server.js que es quien levanta el servicio y el pilar del backend, además de un fichero db.js que gestiona el pool de conexiones con la base de datos. La carpeta routes indica los controladores que se utilizarán para cada petición del frontend y el fichero auth.js comprueba que el token que envía la petición es correcto autorizando así la operación.

5.1.3. *Base de Datos*

La base de datos ha sido creada y gestionada utilizando PostgreSQL. El backend mediante el módulo pg se conecta a la base de datos y permite al usuario interactuar con ella (siempre que la autenticación por token haya sido correcta) y la base le proporciona los datos que este le pide, o al contrario, inserta o actualiza los datos que el backend le manda.

Hemos creado la base a partir de un fichero .sql, a continuación, se muestra el diagrama el diagrama Entidad-Relación de la base de datos.

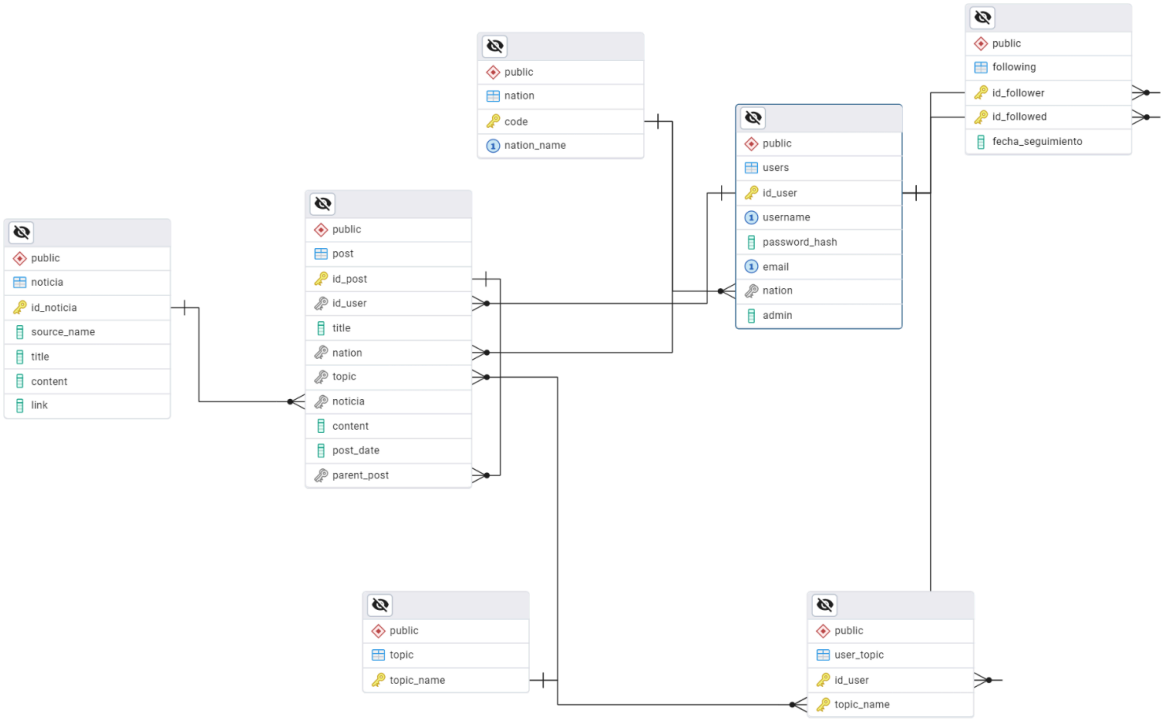

Diagrama Entidad-Relación de la Base de datos

5.2. Codificación y documentación

Frontend

Una muestra del código de una pantalla del frontend es el fichero login.dart (Ver anexos 1, 2, 3 y 4).

Backend

Una muestra del código de nuestro backend son los ficheros server.js (Ver Anexo 5), db.js (Ver Anexo 6) y userRoutes.js (Ver Anexo 7).

6. CONCLUSIÓN

Ágora nace como una propuesta innovadora para centralizar el debate político en un entorno específico, libre y participativo. Frente a las redes sociales generalistas, esta aplicación ofrece un espacio orientado exclusivamente a la expresión y el intercambio de ideas políticas.

7. BIBLIOGRAFÍA Y REFERENCIAS

Documentación oficial y técnica

- Flutter Documentation. *https://docs.flutter.dev/*
- Dart Language Tour. *https://dart.dev/guides/language/language-tour*
- Flutter country_code_picker. https://pub.dev/packages/country_code_picker
- Node.js Documentation. *https://nodejs.org/en/docs/*
- Express.js Guide. *https://expressjs.com/*
- PostgreSQL Documentation. *https://www.postgresql.org/docs/*
- JWT (JSON Web Tokens). *https://jwt.io/introduction/*
- bcrypt for Node.js. *https://github.com/kelektiv/node.bcrypt.js*

APIs y servicios externos

- NewsAPI – Real-time news data. *https://rapidapi.com/letscrape-6bRBa3QguO5/api/real-time-news-data*

Foros y comunidades

- *Stack Overflow. https://stackoverflow.com/*

8. AGRADECIMIENTOS

Edu, compañero en Indra durante FCTs

9. ANEXOS

9.1. Anexo 1

```dart
// Pantalla de inicio de sesión de la aplicación Ágora
documentation:
class LoginScreen extends StatefulWidget {
  const LoginScreen({super.key});

  @override
  LoginScreenState createState() => LoginScreenState();
}

class LoginScreenState extends State<LoginScreen> {
  // Controlador para gestionar el texto ingresado como correo electrónico
  final TextEditingController _emailController = TextEditingController();
  // Controlador para gestionar el texto ingresado como contraseña
  final TextEditingController _passwordController = TextEditingController();
  // Clave global para acceder y validar el estado del formulario
  final GlobalKey<FormState> _formKey = GlobalKey<FormState>();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      // Área de contenido central con padding para evitar estar al borde de la pantalla
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Center(
          child: Form(
            key: _formKey, // Asocia el formulario a la clave global
            child: Column(
              mainAxisSize: MainAxisSize.min,
              children: [
                // Espacio superior
                SizedBox(height: 20),
```

login_screen.dart 1

9.2. Anexo 2

```
38                  // Mensaje de bienvenida con el nombre de la app desde AppData
39                  Text(
40                    'Bienvenido a \${AppData.name}',
41                    style: TextStyle(
42                      fontSize: 20,
43                    ),
44                  ),
45
46                  SizedBox(height: 20),
47
48                  // Título de la sección de inicio de sesión
49                  Text(
50                    'Iniciar sesión',
51                    style: TextStyle(
52                      fontSize: 28,
53                      fontWeight: FontWeight.bold,
54                    ),
55                  ),
56
57                  SizedBox(height: 10),
58
59                  // Campo de texto para ingresar el correo electrónico
60                  TextFormField(
61                    controller: _emailController,
62                    decoration: InputDecoration(labelText: 'Correo electrónico'),
63                    keyboardType: TextInputType.emailAddress,
64                    validator: (value) {
65                      // Validación: no debe estar vacío
66                      if (value == null || value.isEmpty) {
67                        return 'Ingrese su correo electrónico';
68                      }
69                      return null;
```

login_screen.dart 2

9.3. Anexo 3

```
73                  SizedBox(height: 10),
74
75                  // Campo de texto para ingresar la contraseña
76                  TextFormField(
77                    controller: _passwordController,
78                    decoration: InputDecoration(labelText: 'Contraseña'),
79                    obscureText: true, // Oculta el texto para mayor privacidad
80                    validator: (value) {
81                      if (value == null || value.isEmpty) {
82                        return 'Ingrese su contraseña';
83                      }
84                      return null;
85                    },
86                  ),
87
88                  SizedBox(height: 20),
89
90                  // Botón para enviar el formulario y procesar el inicio de sesión
91                  ElevatedButton(
92                    onPressed: () {
93                      // Si el formulario es válido, llama al servicio de autenticación
94                      if (_formKey.currentState!.validate()) {
95                        loginUser(
96                          context: context,
97                          email: _emailController.text,
98                          password: _passwordController.text,
99                        );
100                     }
101                   },
102                   child: Text('Iniciar sesión'),
103                 ),
104
105                 SizedBox(height: 20),
```

login_screen.dart 3

9.4. Anexo 4

```
107                 // Enlace para navegar a la pantalla de registro de nuevos usuarios
108                 TextButton(
109                   onPressed: () {
110                     Navigator.pushNamed(context, AppRoutes.singUpScreen);
111                   },
112                   child: Text(
113                     '¿No tienes una cuenta? Regístrate aquí',
114                     style: TextStyle(
115                       color: Colors.blue,
116                       decoration: TextDecoration.underline,
117                     ),
118                   ),
119                 ),
120
121                 SizedBox(height: 20),
122
123                 // Botón de acceso rápido (login de prueba) sin validación de formulario
124                 ElevatedButton(
125                   onPressed: () {
126                     loginUser(
127                       context: context,
128                       email: 'paco@prueba.com',
129                       password: '123456',
130                     );
131                   },
132                   child: Text('Inicio rápido'),
133                 ),
134               ],
135             ),
136           ),
137         ),
138       ),
139     );
140   }
```

login_screen.dart 4

9.5. Anexo 5

```
// Fichero server.js

// Importar funciones y frameworks/librerías.
const express = require('express');
const cors = require('cors');
const userRoutes = require('./routes/userRoutes');
const postRoutes = require('./routes/postRoutes');
const newsRoutes = require('./routes/newsRoutes');
require('dotenv').config();

// Inicializa Express
const app = express();

// Define el puerto den el que se levantará el servidor
const PORT = process.env.SERVER_PORT || 5000;

// Configura que el backend pueda recibir peticiones con json como body
app.use(express.json({ type: 'application/json', charset: 'utf-8' }));

// Habilita que el backend pueda recibir peticiones desde diferentes dominios.
app.use(cors());

// Configuración de las rutas que han de seguir las peticiones HTTP
app.use('/users', userRoutes);
app.use('/posts', postRoutes);
app.use('/news', newsRoutes);

// Puerto en el que está levantado el servidor
app.listen(PORT, () => {
  console.log(`Servidor corriendo en http://localhost:${PORT}`);
});

```

server.js

9.6. Anexo 6

```
// Importamos el módulo pg de PostgreSQL
const { Pool } = require('pg');
// Cargamos las variables del fichero .env
require('dotenv').config();

// Configuración de la conexión
const pool = new Pool({
  user: process.env.DB_USER,
  host: process.env.DB_HOST,
  database: process.env.DB_NAME,
  password: process.env.DB_PASSWORD,
  port: process.env.DB_PORT,
  client_encoding: 'UTF8',
});

// Ejecutamos la conexión
pool.on('connect', (client) => {
  client.query('SET client_encoding = "UTF8";')
    .then(() => console.log('Codificación del cliente establecida en UTF-8'))
    .catch((err) => console.error('Error al configurar client_encoding:', err));
});

// Mostramos que la conexión ha sido exitosa
pool.on('connect', () => {
  console.log('Conectado a la base de datos PostgreSQL');
});

module.exports = pool;
```

db.js

9.7. Anexo 7

```
// Importar funciones y frameworks/librerías.
const express = require('express');
const { registerUser, loginUser, editProfileUser, userPosts } = require('../controllers/userController');
const { authenticateUser } = require('../middlewares/auth.js');

// Creamos un enrutador para manejar rutas de manera modular, fuera del fichero server.js
const router = express.Router();

// Rutas para el controlador de usuarios
router.post('/register', registerUser);
router.post('/login', loginUser);
router.put('/editProfile/', authenticateUser , editProfileUser);
router.post('/userposts/', authenticateUser , userPosts);

// Exportamos el enrutador para ser utilizado en otros ficheros.
module.exports = router;

```

userRoutes.js