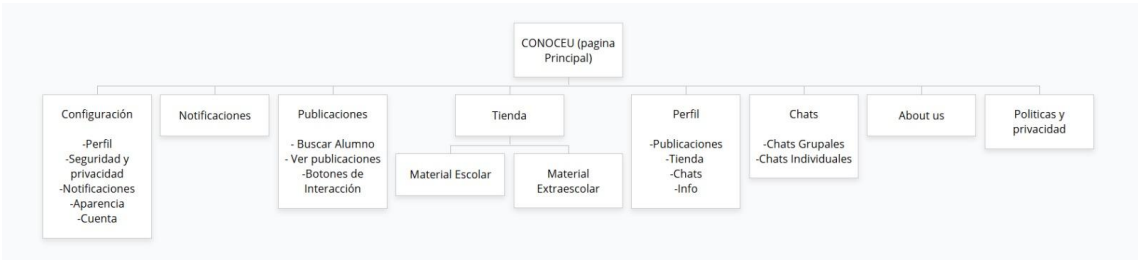


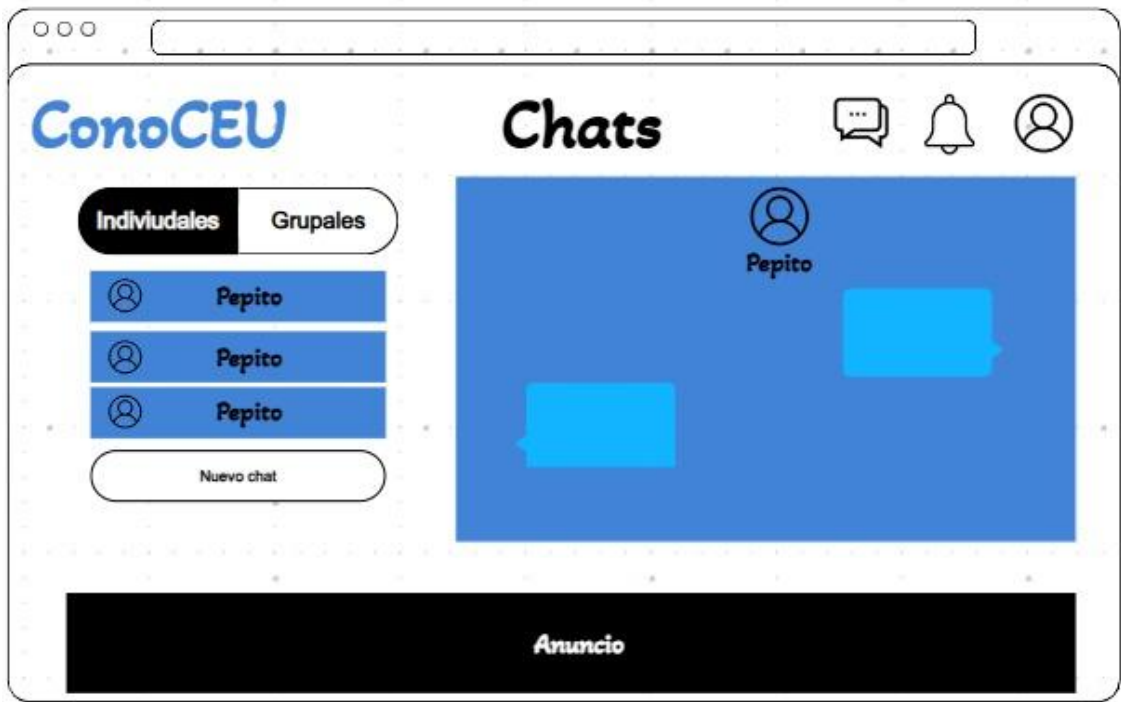


Diseño

SITEMAP



WIREFRAMES



ConoCEU

Configuración

Perfil

Seguridad y privacidad

Notificaciones

Apariencia

Cuenta

Foto de perfil

Cambiar imagen

Nombre y apellidos

Nombre Apellidos

Carrera / Curso

Ing. Sistemas - 3º

País / Intercambio

España / Erasmus

Biografía

Cuéntanos algo sobre ti

ConoCEU

CEU

PERFIL

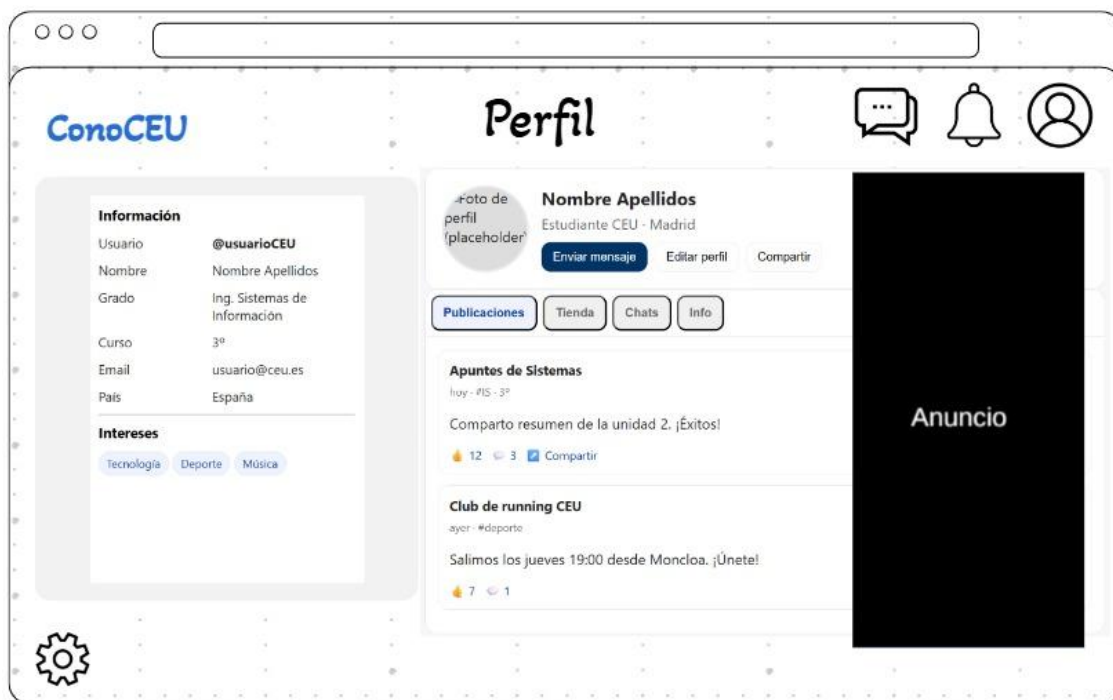
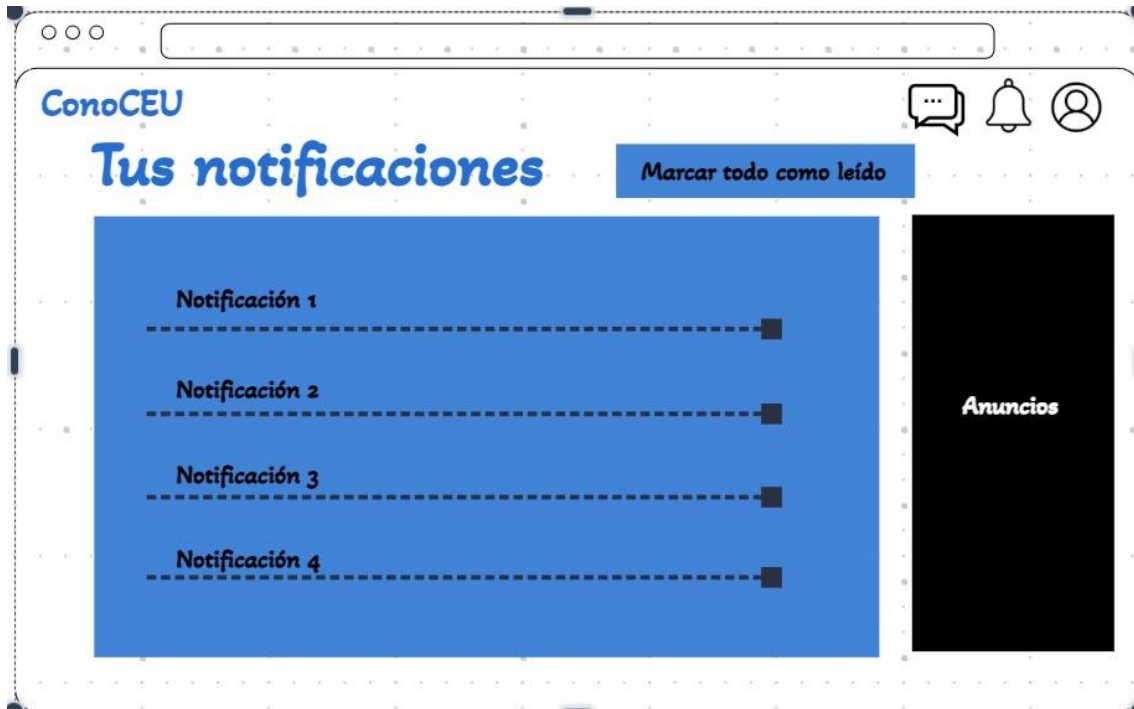
PUBLICACIONES

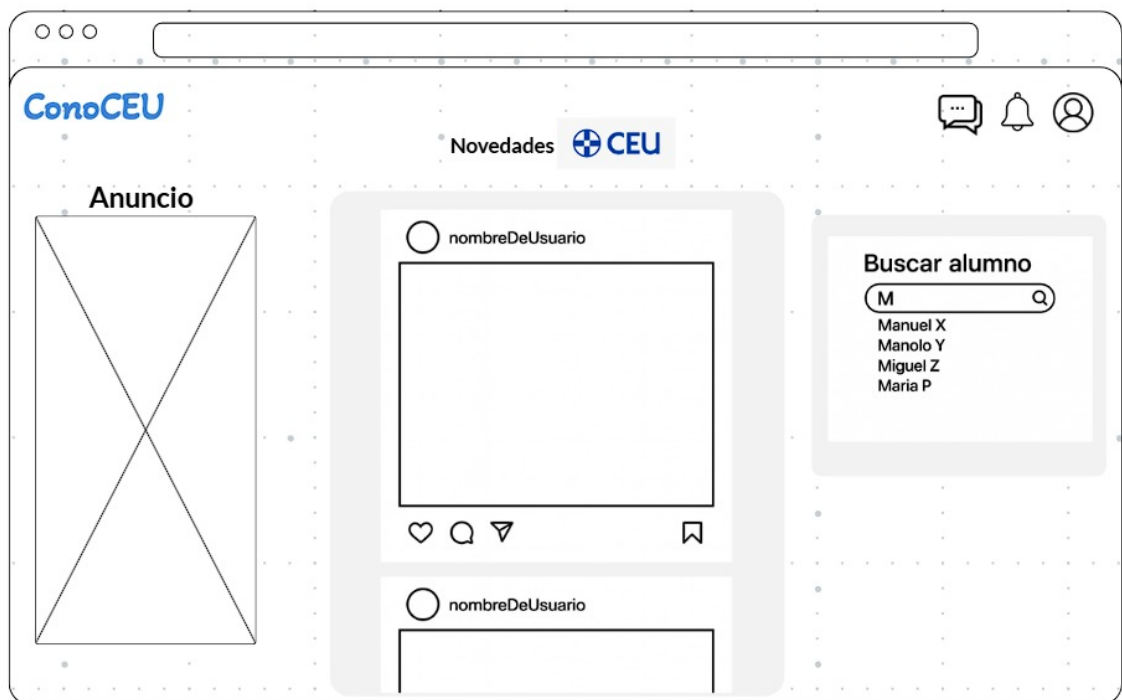
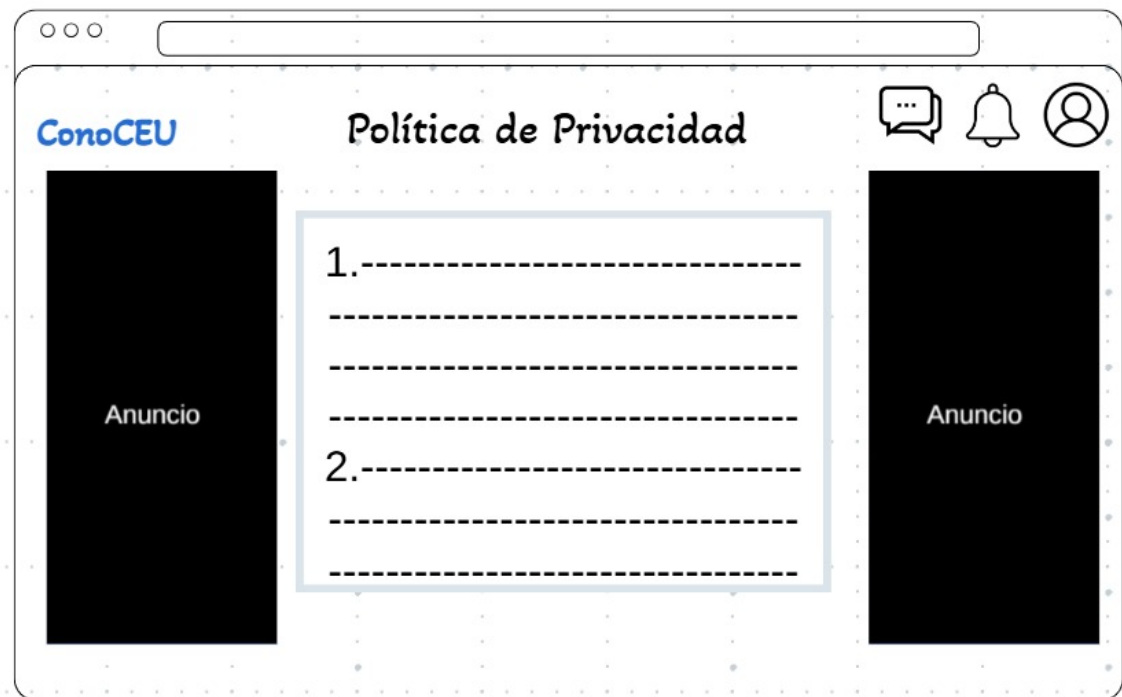
CHATS

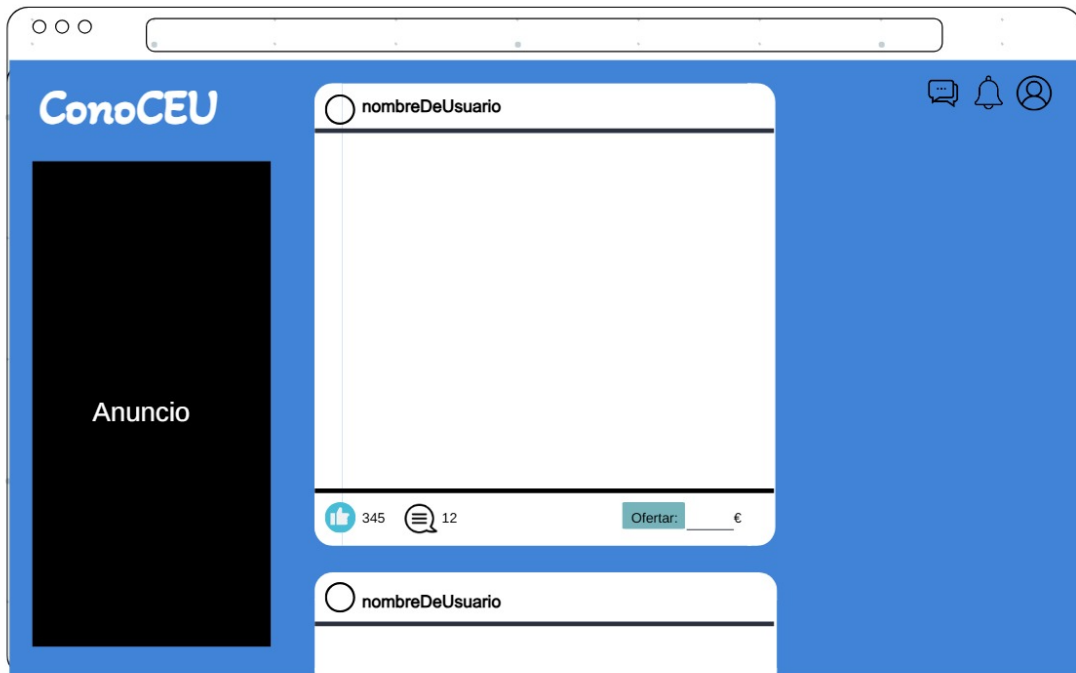
TIENDA

Política de Privacidad

About us









Arquitectura del sistema

La arquitectura de ConoCEU sigue un modelo **cliente-servidor** clásico:

- Una **capa cliente** formada por una aplicación web estática (HTML/CSS/JavaScript), desplegada como frontend.
- Una **capa servidor** implementada en **Node.js** con **Express**, que expone una API REST y gestiona la lógica de negocio y la administración de la plataforma.
- Una **capa de datos** basada en **MySQL**, accesible desde el servidor mediante un pool de conexiones.
- Un canal de **comunicación en tiempo real** basado en **Socket.IO**, utilizado principalmente para la funcionalidad de chat y la gestión de salas.

Esta separación permite que el frontend consuma los servicios del backend a través de peticiones HTTP y sockets, manteniendo desacopladas la interfaz de usuario y la lógica de negocio.

Capa servidor (backend)

El backend se encuentra en la carpeta `server/` y está construido sobre **Node.js** y **Express**, tal y como se especifica en `package.json`.

Los elementos principales son:

- `index.js`: fichero principal del servidor.
- `db.js`: módulo de conexión a la base de datos.

En `index.js` se realiza:

- Carga de variables de entorno mediante `dotenv`.
- Creación de la aplicación Express (`const app = express();`).
- Creación de un servidor HTTP a partir de la app (`http.createServer(app)`), necesario para integrar Socket.IO.
- Configuración de middlewares:
 - `cors()` para habilitar peticiones desde el frontend.
 - `express.json()` para parsear cuerpos JSON.
- Definición de **middleware de autenticación**:
 - `auth`: verifica la presencia de un token JWT en la cabecera `Authorization`, lo valida con `jsonwebtoken` y adjunta los datos del usuario (`req.user`) si el token es correcto.
 - `isAdmin`: comprueba el rol de usuario y restringe el acceso a determinadas rutas solo a administradores.

Sobre estos middlewares se definen varias rutas REST bajo el prefijo `/api/admin/...`, por ejemplo:

- `GET /api/admin/users`: lista usuarios con filtros de búsqueda.
- `DELETE /api/admin/users/:id`: elimina usuarios.
- `PUT /api/admin/users/:id/role`: cambia el rol de un usuario.
- `GET /api/admin/publicaciones`: obtiene todas las publicaciones con información del autor.
- `DELETE /api/admin/publicaciones/:id`: elimina publicaciones.
- `GET /api/admin/productos`: combina productos de `tienda_escolar` y `tienda_extraescolar`.
- `DELETE /api/admin/tienda-escolar/:id` y `DELETE /api/admin/tienda-extraescolar/:id`: eliminan productos de cada tienda.
- `GET /api/admin/stats`: devuelve estadísticas agregadas (número de usuarios, publicaciones y chats).

Todas estas operaciones acceden a la base de datos usando el módulo `db` (pool MySQL).

Socket.IO

En el mismo `index.js` se configura **Socket.IO** a partir del servidor HTTP:

- Se crea una instancia de `Server` de Socket.IO, con configuración CORS que permite el origen del frontend (dominio de Vercel y `localhost` para desarrollo).
- Se añade un middleware de autenticación a nivel de socket, reutilizando el token JWT enviado en `socket.handshake.auth.token`.
- En el evento `connection` se registra cada usuario conectado y se habilitan, entre otros, los eventos:
 - `chat:join`: el usuario abandona salas anteriores y se une a la sala del chat indicado (`chat-{chatId}`).
 - `disconnect`: registra la desconexión del usuario.

Esta capa de sockets proporciona soporte para salas de chat en tiempo real sobre la misma infraestructura que el servidor Express.

Capa de persistencia (base de datos)

El acceso a la base de datos se centraliza en el fichero `server/db.js`.

En este módulo:

- Se importa `mysql2/promise`.
- Se crea un **pool de conexiones** (`mysql.createPool`) usando parámetros de configuración obtenidos de variables de entorno:
 - `DB_HOST`
 - `DB_PORT`
 - `DB_USER`
 - `DB_PASSWORD`
 - `DB_NAME`
- Se exporta el pool para que pueda ser utilizado por el resto del servidor (por ejemplo, en `index.js`).

Todas las consultas SQL del backend se realizan a través de este pool, lo que permite una gestión eficiente de conexiones concurrentes y simplifica el acceso a datos desde las rutas de administración y futuras rutas de negocio.

Capa cliente (frontend)

En la raíz del repositorio se encuentra el archivo `index.html`, que representa la **interfaz principal** del sistema ConoCEU. Desde esta entrada se accede a las distintas secciones de la aplicación (perfil, publicaciones, chats, tienda, política de privacidad, about us) mediante la barra de navegación.

El frontend se despliega como **aplicación web estática**, consumiendo:

- La API REST expuesta por el servidor Node.js para funcionalidades de negocio (usuarios, publicaciones, tienda, administración).
- El canal de Socket.IO para las funcionalidades de chat en tiempo real.

Adicionalmente, el repositorio está asociado a un despliegue en Vercel, lo que indica que el frontend se sirve desde un hosting estático, mientras que el backend se ejecuta de forma separada como servicio Node.js.