Etapa 0 – Setup base (esqueleto mínimo de la SPA)

Crear la estructura de carpetas

En la raíz del frontend:

```
frontend/
 index.html

 assets/
  css/
   bootstrap.min.css   (opcional si no usas CDN)
   styles.css

 js/
  core/
   component.js
   view.js

  views/
   HomeView.js

  app.js
```

- assets/css será solo para Bootstrap + CSS mínimo propio.
- Todo el comportamiento irá dentro de js/, organizado por "núcleo" (core) y vistas (views).

index.html – HTML base con Bootstrap y contenedor #app

Archivo: frontend/index.html

```
<!doctype html>
<html lang="es">
<head>
 <meta charset="utf-8" />
```

```
 <title>Nexo - SPA Frontend</title>
 <meta name="viewport" content="width=device-width, initial-scale=1" />

 <!-- Bootstrap CSS (CDN) -->
 <link
  href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
  rel="stylesheet"
  integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
  crossorigin="anonymous"
 >

 <!-- CSS propio mínimo -->
 <link rel="stylesheet" href="assets/css/styles.css">
</head>
<body class="bg-light">

 <!-- Navbar vacía simple con Bootstrap -->
 <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
  <div class="container">
   <a class="navbar-brand" href="#">Nexo</a>
   <!-- Más adelante aquí irán los links, user, logout, etc. -->
  </div>
 </nav>

 <main class="container py-4">
  <!-- Aquí se montará la SPA -->
  <div id="app"></div>
 </main>
```

```
 <!-- Bootstrap JS (opcional por ahora, solo para componentes que lo requieran) -->
 <script
  src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
  integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"
  crossorigin="anonymous"
 ></script>

 <!-- Punto de entrada JS -->
 <script type="module" src="js/app.js"></script>
</body>
</html>
```

- Un solo para toda la SPA.
- Nada de HTML “suelto” de vistas aquí; todo lo que se vea en pantalla vendrá desde JS (componentes).

CSS mínimo

Archivo: frontend/assets/css/styles.css

```
/* Ajustes mínimos por encima de Bootstrap */

body {
  /* Simplemente para diferenciar un poco el entorno */
  background-color: #f5f5f5;
}

/* Más adelante puedes definir variables de marca, etc. */
```

En esta etapa basta con algo muy simple para enfatizar que casi todo vendrá de Bootstrap.

Clase base Component (estilo React-like)

Archivo: frontend/js/core/component.js

```
// js/core/component.js

export class Component {
  constructor(props = {}) {
    this.props = props;
    this.root = null;  // nodo DOM donde se monta el componente
  }

  /**
   * Devuelve el HTML del componente como string.
   * Similar a "return JSX" en React.
   */
  template() {
    return "";
  }

  /**
   * Monta el componente en el elemento root.
   * Similar a ReactDOM.render.
   */
  mount(rootElement) {
    this.root = rootElement;
    this.root.innerHTML = this.template();
    this.afterRender();
  }

  /**
```

```
  * Se ejecuta después de pintar el HTML.
  * Aquí se enganchan eventos, listeners, etc.
  * Similar a useEffect/componentDidMount.
  */
 afterRender() {
   // Por defecto no hace nada.
 }

 /**
  * Limpia el contenido del root.
  */
 unmount() {
   if (this.root) {
     this.root.innerHTML = "";
   }
 }

 // NUEVO: volver a pintar el componente en el mismo root
 rerender() {
   if (!this.root) return;
   this.root.innerHTML = this.template();
   this.afterRender();
 }
}
```

La idea que puedes remarcar:

- Contrato de componente:
  - Recibe props.
  - Tiene template() → devuelve HTML string.
  - Tiene afterRender() → engancha eventos.

Helper renderView para montar vistas en #app

Archivo: frontend/js/core/view.js

```
// js/core/view.js


import { Component } from "./component.js";


/**
 * Renderiza una vista (clase que extiende Component)
 * dentro del contenedor principal #app.
 */
export function renderView(ViewClass, props = {}) {
  const root = document.getElementById("app");
  if (!root) {
    throw new Error("No se encontró el elemento #app en el DOM");
  }

  // Por ahora no guardamos la instancia globalmente;
  // más adelante, con router, podremos manejar mount/unmount.
  const view = new ViewClass(props);
  view.mount(root);
}
```

Esto estandariza cómo se “pinta” cualquier vista principal. Más adelante el router llamará siempre a renderView.

Primera vista: HomeView

Archivo: frontend/js/views/HomeView.js

```
// js/views/HomeView.js


import { Component } from "../core/component.js";
```

```
export class HomeView extends Component {
 template() {
  return `
   <section class="text-center">
    <h1 class="display-5 fw-bold mb-3">Hola Nexo</h1>
    <p class="lead">
     Esta es la primera vista de la SPA construida con HTML, Bootstrap y JS modular.
    </p>
    <p class="text-muted">
     Más adelante aquí irá el login, el feed y todo el resto de la aplicación.
    </p>
   </section>
  `;
 }
}
```

En esta etapa no necesitas lógica en afterRender(). Solo mostrar algo en pantalla para validar el flujo.

Punto de entrada app.js

Archivo: frontend/js/app.js

```
// js/app.js

import { renderView } from "./core/view.js";
import { HomeView } from "./views/HomeView.js";

// En etapas posteriores aquí se inicializará el router.
// Por ahora solo pintamos HomeView.
document.addEventListener("DOMContentLoaded", () => {
```

```
  renderView(HomeView);
});
```

Esta es la mínima “aplicación”:

- Espera al DOM cargado.
- Llama a renderView(HomeView) que:
    - Busca #app.
    - Instancia HomeView.
    - Monta el HTML.

Cómo probar la Etapa 0

1. Ubícate en la carpeta frontend/.
2. Sirve los archivos estáticos con algún servidor simple, por ejemplo:
    - Extensión Live Server de VS Code, o
    - python -m http.server 5500 (en esa carpeta) y abrir http://localhost:5500/.
3. Verifica que:
    - Ves la navbar oscura con el texto “Nexo”.
    - Dentro del contenedor aparece el título “Hola Nexo” y el texto explicativo.

Etapa 1 – Router SPA y Layout con Bootstrap

Ajuste de index.html para tener contenedor de Navbar

Vamos a separar el espacio de la navbar del espacio de la vista:

frontend/index.html (solo muestro la parte del <body>):

```
<body class="bg-light">

  <!-- Contenedor para la Navbar de la SPA -->
  <div id="navbar-root"></div>

  <main class="container py-4">
    <!-- Aquí se montará la vista actual -->
    <div id="app"></div>
  </main>

  <!-- Bootstrap JS (opcional) -->
  <script
    src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"
    crossorigin="anonymous"
  ></script>

  <!-- Punto de entrada JS -->
  <script type="module" src="js/app.js"></script>
</body>
```

La navbar ya no está “hardcodeada” en HTML, sino que se montará como componente en #navbar-root.

Crear las vistas estáticas iniciales

LoginView

frontend/js/views/LoginView.js:

```
// js/views/LoginView.js
import { Component } from "../core/component.js";

export class LoginView extends Component {
 template() {
  return `
   <section class="row justify-content-center">
    <div class="col-md-6 col-lg-4">
     <h1 class="h3 mb-3 text-center">Iniciar sesión</h1>
     <form>
      <div class="mb-3">
       <label class="form-label">Email</label>
       <input type="email" class="form-control" placeholder="tucorreo@nexo.com" />
      </div>
      <div class="mb-3">
       <label class="form-label">Contraseña</label>
       <input type="password" class="form-control" placeholder="********" />
      </div>
      <button type="submit" class="btn btn-primary w-100">Ingresar</button>
     </form>
     <p class="mt-3 text-center">
      ¿No tienes cuenta?
      <a href="#/register">Crear cuenta</a>
     </p>
    </div>
   </section>
```

```
    `;
  }

  afterRender() {
    // Más adelante aquí se enganchará el submit real hacia el backend.
  }
}
```

RegisterView

frontend/js/views/RegisterView.js:

```
// js/views/RegisterView.js
import { Component } from "../core/component.js";

export class RegisterView extends Component {
  template() {
    return `
      <section class="row justify-content-center">
        <div class="col-md-6 col-lg-5">
          <h1 class="h3 mb-3 text-center">Crear cuenta</h1>
          <form>
            <div class="row">
              <div class="col-md-6 mb-3">
                <label class="form-label">Nombre</label>
                <input type="text" class="form-control" />
              </div>
              <div class="col-md-6 mb-3">
                <label class="form-label">Apellido</label>
                <input type="text" class="form-control" />
              </div>
```

```
        </div>
        <div class="mb-3">
          <label class="form-label">Nombre de usuario</label>
          <input type="text" class="form-control" />
        </div>
        <div class="mb-3">
          <label class="form-label">Email</label>
          <input type="email" class="form-control" />
        </div>
        <div class="mb-3">
          <label class="form-label">Fecha de nacimiento</label>
          <input type="date" class="form-control" />
        </div>
        <div class="mb-3">
          <label class="form-label">Contraseña</label>
          <input type="password" class="form-control" />
        </div>
        <button type="submit" class="btn btn-primary w-100">Registrarme</button>
      </form>
      <p class="mt-3 text-center">
        ¿Ya tienes cuenta?
        <a href="#/login">Iniciar sesión</a>
      </p>
    </div>
  </section>
 `;
}

afterRender() {
```

```
    // Más adelante se conectará con /auth/register.
  }
}
```

FeedView

frontend/js/views/FeedView.js:

```
// js/views/FeedView.js
import { Component } from "../core/component.js";


export class FeedView extends Component {
  template() {
    return `
      <section>
        <h1 class="h3 mb-4">Feed global</h1>
        <div class="alert alert-info">
          Aquí aparecerán los posts cuando conectemos con el backend.
        </div>
        <p class="text-muted">
          Esta vista es estática por ahora; sirve para probar el router.
        </p>
      </section>
    `;
  }
}
```

NotFoundView (opcional pero recomendable)

frontend/js/views/NotFoundView.js:

```
// js/views/NotFoundView.js
import { Component } from "../core/component.js";

export class NotFoundView extends Component {
 template() {
  return `
   <section class="text-center">
    <h1 class="display-6 mb-3">404</h1>
    <p class="lead">La página que buscas no existe.</p>
    <a href="#/login" class="btn btn-outline-primary mt-3">Ir al login</a>
   </section>
  `;
 }
}
```

Crear el router por hash

frontend/js/core/router.js:

```
// js/core/router.js

import { renderView } from "./view.js";
import { LoginView } from "../views/LoginView.js";
import { RegisterView } from "../views/RegisterView.js";
import { FeedView } from "../views/FeedView.js";
import { NotFoundView } from "../views/NotFoundView.js";

// Tabla de rutas: hash -> clase de vista
const routes = {
```

```
  "#/login": LoginView,
  "#/register": RegisterView,
  "#/feed": FeedView,
};

/**
 * Lógica para decidir qué vista renderizar según location.hash
 */
function navigate() {
  const hash = window.location.hash || "#/login";
  const ViewClass = routes[hash] || NotFoundView;
  renderView(ViewClass);
}

/**
 * Inicializa el router: escucha cambios en el hash
 * y hace el primer render.
 */
export function initRouter() {
  window.addEventListener("hashchange", navigate);
  navigate();
}
```

- Este router es equivalente, en espíritu, a algo como React Router:
    - #/login, #/register, #/feed son las “rutas”.
- El hash se puede cambiar con <a href="#/feed"> sin recargar la página.

Navbar como componente

frontend/js/components/Navbar.js:

```
// js/components/Navbar.js
import { Component } from "../core/component.js";


export class Navbar extends Component {
  template() {
    return `
      <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <div class="container">
          <a class="navbar-brand" href="#/feed">Nexo</a>


          <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarMain">
            <span class="navbar-toggler-icon"></span>
          </button>


          <div class="collapse navbar-collapse" id="navbarMain">
            <ul class="navbar-nav me-auto mb-2 mb-lg-0">
              <li class="nav-item">
                <a class="nav-link" href="#/feed">Feed</a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="#/login">Login</a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="#/register">Registro</a>
              </li>
            </ul>
```

```
          <span class="navbar-text">
            <!-- Más adelante aquí irán datos del usuario / botón logout -->
            Invitado
          </span>
        </div>
      </div>
    </nav>
  `;
 }

 afterRender() {
  // Más adelante se podrá actualizar según si hay usuario logueado o no.
 }
}
```

En esta etapa la navbar es estática, sin integración con el estado de autenticación.

Actualizar app.js para montar Navbar + inicializar router

frontend/js/app.js:

```
// js/app.js

import { renderView } from "./core/view.js";
import { HomeView } from "./views/HomeView.js";  // opcional si aún la quieres usar
import { initRouter } from "./core/router.js";
import { Navbar } from "./components/Navbar.js";

document.addEventListener("DOMContentLoaded", () => {
 // 1. Montar la Navbar en su contenedor
 const navbarRoot = document.getElementById("navbar-root");
```

```
 if (navbarRoot) {
   const navbar = new Navbar();
   navbar.mount(navbarRoot);
 }

 // 2. Inicializar el router (esto pintará la vista según el hash actual)
 initRouter();

 // Si quieres que HomeView desaparezca, puedes eliminar esta línea.
 // renderView(HomeView);
});
```

A partir de ahora, lo que controla qué vista se ve es el router.

HomeView puedes dejarla solo como ejemplo o eliminarla para simplificar.

Prueba de la Etapa 1

1. Levanta tu servidor estático en frontend/.
2. Abre:
    - http://localhost:5500/#/login → deberías ver el formulario de login.
    - http://localhost:5500/#/register → formulario de registro.
    - http://localhost:5500/#/feed → feed estático.
3. Usa los enlaces de la navbar:
    - Al hacer clic en “Feed” debe cambiar la vista.
    - “Login” y “Registro” navegan entre vistas sin recargar la página.
4. Prueba una ruta inexistente, por ejemplo #/otra-cosa:
    - Debe aparecer la NotFoundView.

Etapa 2 – Autenticación y estado global

Estado global: store.js

Archivo: frontend/js/core/store.js

```
// js/core/store.js

const KEY = "nexo_auth";

// Estado global sencillo
export const store = {
  user: null,   // { user_uuid, nombre, apellido, email, username }
  token: null,  // access_token (JWT)
};

const listeners = new Set();

// Cargar desde localStorage al iniciar
function load() {
  try {
    const raw = localStorage.getItem(KEY);
    if (!raw) return;
    const data = JSON.parse(raw);
    store.user = data.user || null;
    store.token = data.token || null;
  } catch (e) {
    console.error("Error cargando estado de auth:", e);
  }
}

// Guardar en localStorage
```

```
function save() {
  try {
    const data = {
      user: store.user,
      token: store.token,
    };
    localStorage.setItem(KEY, JSON.stringify(data));
  } catch (e) {
    console.error("Error guardando estado de auth:", e);
  }
}

function notify() {
  for (const fn of listeners) {
    try {
      fn(store);
    } catch (e) {
      console.error("Error en listener de auth:", e);
    }
  }
}

// Suscripción a cambios de auth
export function subscribeAuth(listener) {
  listeners.add(listener);
  // Devolvemos función para desuscribirse si alguna vez la necesitas
  return () => listeners.delete(listener);
}
```

```
// Establecer sesión
export function setAuth(user, token) {
  store.user = user;
  store.token = token;
  save();
  notify();
}

// Limpiar sesión
export function clearAuth() {
  store.user = null;
  store.token = null;
  save();
  notify();
}

// Helper opcional
export function isAuthenticated() {
  return !!store.token;
}

// Ejecutar carga inicial una sola vez
load();
```

Con esto, al refrescar el navegador se mantiene la sesión.

Cliente HTTP genérico: api/client.js

Archivo: frontend/js/api/client.js

```
// js/api/client.js
```

```
import { store, clearAuth } from "../core/store.js";

const BASE_URL = "http://localhost:5000"; // ajusta si tu backend corre en otro host/puerto

export async function apiFetch(path, options = {}) {
 const url = path.startsWith("http") ? path : BASE_URL + path;

 const opts = { ...options };
 opts.headers = opts.headers ? { ...opts.headers } : {};

 // Si el body no es FormData, asumimos JSON
 if (opts.body && !(opts.body instanceof FormData)) {
  opts.headers["Content-Type"] = "application/json";
 }

 // Adjuntar Authorization si hay token
 if (store.token) {
  opts.headers["Authorization"] = `Bearer ${store.token}`;
 }

 let response;

 try {
  response = await fetch(url, opts);
 } catch (err) {
  throw new Error("Error de red o servidor no disponible");
 }
 // Manejo de 401: limpiar sesión y redirigir al login
 if (response.status === 401) {
```

```
    clearAuth();
    window.location.hash = "#/login";
    throw new Error("No autorizado, por favor inicia sesión nuevamente.");
  }

  let data;
  try {
    data = await response.json();
  } catch (err) {
    throw new Error("Respuesta inválida del servidor");
  }

  if (!response.ok) {
    // El backend usa { error: "mensaje" }
    const msg = data && data.error ? data.error : "Error en la petición";
    throw new Error(msg);
  }

  return data;
}

export const API_BASE_URL = BASE_URL;
```

Con apiFetch estandarizas:

- URL base.
- Headers comunes.
- Manejo de JSON y errores.
- Manejo automático de 401.

Servicios específicos de auth: api/auth.js

Archivo: frontend/js/api/auth.js

```
// js/api/auth.js


import { apiFetch } from "./client.js";


export async function loginApi({ email, password }) {
  return apiFetch("/auth/login", {
    method: "POST",
    body: JSON.stringify({ email, password }),
  });
}


export async function registerApi(payload) {
  // payload: { nombre, apellido, email, fecha_nacimiento, username, password }
  return apiFetch("/auth/register", {
    method: "POST",
    body: JSON.stringify(payload),
  });
}
```

Conectar LoginView con el backend y el store

Actualizamos LoginView para:

- Leer los valores del form.
- Llamar a loginApi.
- Guardar en store con setAuth.
- Redirigir a #/feed.
- Mostrar mensajes de error.

frontend/js/views/LoginView.js:

```
// js/views/LoginView.js
import { Component } from "../core/component.js";
import { loginApi } from "../api/auth.js";
import { setAuth } from "../core/store.js";

export class LoginView extends Component {
  template() {
    return `
      <section class="row justify-content-center">
        <div class="col-md-6 col-lg-4">
          <h1 class="h3 mb-3 text-center">Iniciar sesión</h1>

          <div id="login-error" class="alert alert-danger d-none"></div>

          <form id="login-form" novalidate>
            <div class="mb-3">
              <label class="form-label">Email</label>
              <input name="email" type="email" class="form-control"
placeholder="tucorreo@nexo.com" required />
            </div>
            <div class="mb-3">
              <label class="form-label">Contraseña</label>
              <input name="password" type="password" class="form-control" placeholder="********"
required />
            </div>
            <button type="submit" class="btn btn-primary w-100">Ingresar</button>
          </form>

          <p class="mt-3 text-center">
```

```
        ¿No tienes cuenta?
        <a href="#/register">Crear cuenta</a>
       </p>
     </div>
   </section>
  `;
}

afterRender() {
  const form = this.root.querySelector("#login-form");
  const errorBox = this.root.querySelector("#login-error");
  if (!form) return;

  form.addEventListener("submit", async (ev) => {
    ev.preventDefault();
    errorBox.classList.add("d-none");
    errorBox.textContent = "";

    const formData = new FormData(form);
    const email = (formData.get("email") || "").toString().trim().toLowerCase();
    const password = (formData.get("password") || "").toString();

    if (!email || !password) {
      errorBox.textContent = "Email y contraseña son requeridos.";
      errorBox.classList.remove("d-none");
      return;
    }

    try {
```

```
      const data = await loginApi({ email, password });
      // backend: { access_token, user }
      setAuth(data.user, data.access_token);
      window.location.hash = "#/feed";
    } catch (err) {
      errorBox.textContent = err.message || "Error al iniciar sesión.";
      errorBox.classList.remove("d-none");
    }
  });
 }
}
```

Conectar RegisterView con el backend

Actualizamos RegisterView para:

- Tomar todos los campos.
- Llamar registerApi.
- Tras registro exitoso, redirigir a login (o incluso podrías loguear automáticamente, pero para el curso es didáctico enviar al login).

frontend/js/views/RegisterView.js:

```
// js/views/RegisterView.js
import { Component } from "../core/component.js";
import { registerApi } from "../api/auth.js";

export class RegisterView extends Component {
 template() {
  return `
   <section class="row justify-content-center">
    <div class="col-md-6 col-lg-5">
     <h1 class="h3 mb-3 text-center">Crear cuenta</h1>
```

```
<div id="register-error" class="alert alert-danger d-none"></div>
<div id="register-success" class="alert alert-success d-none"></div>

<form id="register-form" novalidate>
 <div class="row">
  <div class="col-md-6 mb-3">
   <label class="form-label">Nombre</label>
   <input name="nombre" type="text" class="form-control" required />
  </div>
  <div class="col-md-6 mb-3">
   <label class="form-label">Apellido</label>
   <input name="apellido" type="text" class="form-control" required />
  </div>
 </div>
 <div class="mb-3">
  <label class="form-label">Nombre de usuario</label>
  <input name="username" type="text" class="form-control" required />
 </div>
 <div class="mb-3">
  <label class="form-label">Email</label>
  <input name="email" type="email" class="form-control" required />
 </div>
 <div class="mb-3">
  <label class="form-label">Fecha de nacimiento</label>
  <input name="fecha_nacimiento" type="date" class="form-control" required />
 </div>
 <div class="mb-3">
  <label class="form-label">Contraseña</label>
```

```
          <input name="password" type="password" class="form-control" required />
          <div class="form-text">
            Debe tener mayúscula, minúscula, número y símbolo.
          </div>
        </div>
        <button type="submit" class="btn btn-primary w-100">Registrarme</button>
      </form>

      <p class="mt-3 text-center">
        ¿Ya tienes cuenta?
        <a href="#/login">Iniciar sesión</a>
      </p>
    </div>
  </section>
  `;
}

afterRender() {
  const form = this.root.querySelector("#register-form");
  const errorBox = this.root.querySelector("#register-error");
  const successBox = this.root.querySelector("#register-success");
  if (!form) return;

  form.addEventListener("submit", async (ev) => {
    ev.preventDefault();
    errorBox.classList.add("d-none");
    successBox.classList.add("d-none");
    errorBox.textContent = "";
    successBox.textContent = "";
```

```
    const fd = new FormData(form);

    const payload = {
      nombre: (fd.get("nombre") || "").toString().trim(),
      apellido: (fd.get("apellido") || "").toString().trim(),
      username: (fd.get("username") || "").toString().trim().toLowerCase(),
      email: (fd.get("email") || "").toString().trim().toLowerCase(),
      fecha_nacimiento: (fd.get("fecha_nacimiento") || "").toString(),
      password: (fd.get("password") || "").toString(),
    };

    if (!payload.nombre || !payload.apellido || !payload.username || !payload.email ||
!payload.fecha_nacimiento || !payload.password) {
      errorBox.textContent = "Todos los campos son obligatorios.";
      errorBox.classList.remove("d-none");
      return;
    }

    try {
      await registerApi(payload);
      successBox.textContent = "Registro exitoso. Ahora puedes iniciar sesión.";
      successBox.classList.remove("d-none");
      // Opcional: redirigir automáticamente después de un tiempo
      setTimeout(() => {
        window.location.hash = "#/login";
      }, 1500);
    } catch (err) {
      errorBox.textContent = err.message || "Error al registrar usuario.";
```

```
      errorBox.classList.remove("d-none");
    }
  });
 }
}
```

Proteger rutas privadas en el router

Actualizamos frontend/js/core/router.js para:

- Definir qué rutas son públicas y cuáles requieren sesión.
- Redirigir a #/login si se intenta acceder a una ruta privada sin token.
- Opcional: si ya está autenticado y va a #/login o #/register, redirigir a #/feed.

```
// js/core/router.js

import { renderView } from "./view.js";
import { LoginView } from "../views/LoginView.js";
import { RegisterView } from "../views/RegisterView.js";
import { FeedView } from "../views/FeedView.js";
import { NotFoundView } from "../views/NotFoundView.js";
import { isAuthenticated } from "./store.js";

const routes = {
  "#/login": LoginView,
  "#/register": RegisterView,
  "#/feed": FeedView,
};

// Rutas públicas (no requieren sesión)
const publicRoutes = new Set(["#/login", "#/register"]);
```

```
function navigate() {
  let hash = window.location.hash || "#/login";

  const authenticated = isAuthenticated();

  // Si la ruta es privada y no hay sesión, ir a login
  if (!publicRoutes.has(hash) && !authenticated) {
    hash = "#/login";
    window.location.hash = hash;
  }

  // Si está autenticado e intenta ir a login/register, redirigir a feed
  if (authenticated && (hash === "#/login" || hash === "#/register")) {
    hash = "#/feed";
    window.location.hash = hash;
  }

  const ViewClass = routes[hash] || NotFoundView;
  renderView(ViewClass);
}

export function initRouter() {
  window.addEventListener("hashchange", navigate);
  navigate();
}
```

Más adelante, cuando tengas más vistas (Profile, Friends, Interactions), simplemente las agregas a routes y las tratas como privadas por defecto (no las incluyes en publicRoutes).

Navbar reaccionando al estado de autenticación

Actualizamos frontend/js/components/Navbar.js para:

- Mostrar menú diferente si hay sesión o no.
- Incluir botón de Logout que:
    - clearAuth().
    - Redirige a #/login.

```
// js/components/Navbar.js

import { Component } from "../core/component.js";

import { store, clearAuth, isAuthenticated, subscribeAuth } from "../core/store.js";


export class Navbar extends Component {

 template() {

  const loggedIn = isAuthenticated();

  const user = store.user;


  return `

   <nav class="navbar navbar-expand-lg navbar-dark bg-dark">

    <div class="container">

     <a class="navbar-brand" href="#/feed">Nexo</a>


     <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarMain">

      <span class="navbar-toggler-icon"></span>

     </button>


     <div class="collapse navbar-collapse" id="navbarMain">

      <ul class="navbar-nav me-auto mb-2 mb-lg-0">

       ${

        loggedIn

         ? `
```

```
            <li class="nav-item">
              <a class="nav-link" href="#/feed">Feed</a>
            </li>
          `
          : ""
        }
      </ul>

      <div class="d-flex align-items-center gap-2">
        ${
          loggedIn
            ? `
              <span class="navbar-text text-nowrap">
                ${user?.nombre || ""} ${user?.apellido || ""}
              </span>
              <button id="btn-logout" class="btn btn-outline-light btn-sm">
                Salir
              </button>
            `
            : `
              <a class="btn btn-outline-light btn-sm" href="#/login">Ingresar</a>
              <a class="btn btn-primary btn-sm" href="#/register">Registrarme</a>
            `
        }
      </div>
    </div>
  </div>
</nav>
`;
```

```
}

afterRender() {
  // Botón logout
  const btnLogout = this.root.querySelector("#btn-logout");
  if (btnLogout) {
    btnLogout.addEventListener("click", () => {
      clearAuth();
      window.location.hash = "#/login";
    });
  }

  // Suscribirse UNA sola vez a cambios de auth
  if (!this._unsubscribe) {
    this._unsubscribe = subscribeAuth(() => {
      // Cuando cambie el auth, volvemos a pintar la navbar
      this.rerender();
    });
  }
}

// Opcional: si algún día desmontas la navbar
unmount() {
  if (this._unsubscribe) {
    this._unsubscribe();
    this._unsubscribe = null;
  }
  super.unmount();
}
```

```
}
```

Si recargas la página con una sesión válida en localStorage, la Navbar mostrará nombre/apellido y el botón “Salir”. Si no hay sesión, mostrará los botones “Ingresar” y “Registrarme”.

Prueba de la Etapa 2

1. Backend levantado en http://localhost:5000 con tu código Flask.
2. Frontend servido (p. ej. http://localhost:5500).
3. Flujo:
    - Ve a #/register, crea un usuario válido (siguiendo las reglas del backend).
    - Tras el éxito, te debería redirigir (o permitir ir) a #/login.
    - Haz login:
        - Debe llamar a /auth/login.
        - Guardar JWT y usuario en localStorage.
        - Redirigir a #/feed.
    - Navbar debe cambiar a modo “logueado”.
    - Si cambias manualmente location.hash a #/feed sin estar logueado, el router te manda a #/login.
    - Si estando logueado vas a #/login, el router te manda a #/feed.

Etapa 3: Feed y Posts por componentes

Módulo de API para posts

Crea frontend/js/api/posts.js:

```
// js/api/posts.js
import { apiFetch } from "./client.js";

export async function listPosts({ limit = 10, offset = 0 } = {}) {
  const params = new URLSearchParams({
    limit: String(limit),
    offset: String(offset),
  });

  return apiFetch(`/posts?${params.toString()}`);
}

export async function createPost(formData) {
  // formData debe ser FormData con campos: texto, (opcional) imagen
  return apiFetch("/posts", {
    method: "POST",
    body: formData,
  });
}

export async function likePost(postId) {
  return apiFetch(`/posts/${postId}/like`, {
    method: "POST",
  });
}
```

```
export async function unlikePost(postId) {
  return apiFetch(`/posts/${postId}/like`, {
    method: "DELETE",
  });
}
```

Componente PostForm (crear nuevo post)

Archivo: frontend/js/components/PostForm.js

```
// js/components/PostForm.js
import { Component } from "../core/component.js";
import { createPost } from "../api/posts.js";

export class PostForm extends Component {
  template() {
    return `
      <div class="card mb-4">
        <div class="card-body">
          <h2 class="h5 mb-3">Crear publicación</h2>

          <div id="postform-error" class="alert alert-danger d-none"></div>

          <form id="postform-form" novalidate>
            <div class="mb-3">
              <textarea
                name="texto"
                class="form-control"
                rows="3"
                placeholder="¿Qué estás pensando?"
                required
```

```
          ></textarea>
        </div>

        <div class="mb-3">
          <label class="form-label">Imagen (opcional)</label>
          <input
            type="file"
            name="imagen"
            class="form-control"
            accept=".jpg,.jpeg,.png,.gif,.webp,image/*"
          />
          <div class="form-text">
            Tamaño máximo según configuración del servidor.
          </div>
        </div>

        <div class="d-flex justify-content-end">
          <button type="submit" class="btn btn-primary">
            Publicar
          </button>
        </div>
      </form>
    </div>
  </div>
  `;
}

afterRender() {
  const form = this.root.querySelector("#postform-form");
```

```
const errorBox = this.root.querySelector("#postform-error");
if (!form) return;

form.addEventListener("submit", async (ev) => {
  ev.preventDefault();
  errorBox.classList.add("d-none");
  errorBox.textContent = "";

  const fd = new FormData(form);
  const texto = (fd.get("texto") || "").toString().trim();

  if (!texto) {
    errorBox.textContent = "El texto es obligatorio.";
    errorBox.classList.remove("d-none");
    return;
  }

  try {
    const post = await createPost(fd);
    // Limpia solo si fue bien
    form.reset();

    // Notifica a la vista que se creó un post
    if (typeof this.props.onPostCreated === "function") {
      this.props.onPostCreated(post);
    }
  } catch (err) {
    errorBox.textContent = err.message || "Error al crear la publicación.";
    errorBox.classList.remove("d-none");
```

```
    }
  });
 }
}
```

La vista (FeedView) recibirá el post creado y decidirá dónde insertarlo (normalmente al inicio del feed).

Componente PostCard (mostrar cada post)

Archivo: frontend/js/components/PostCard.js

```
// js/components/PostCard.js
import { Component } from "../core/component.js";
import { likePost, unlikePost } from "../api/posts.js";
import { API_BASE_URL } from "../api/client.js";

export class PostCard extends Component {
 template() {
  const p = this.props.post;
  const author = p.author || {};
  const counts = p.counts || { likes: 0, comments: 0, shares: 0 };
  const liked = !!p.liked_by_me;

  const avatarUrl = author.avatar_url
   ? API_BASE_URL + author.avatar_url
   : "https://placehold.co/40x40";

  const createdAt = p.created_at
   ? new Date(p.created_at).toLocaleString()
   : "";
```

```
const imagenUrl = p.imagen_url ? API_BASE_URL + p.imagen_url : null;


return `
  <article class="card mb-3" data-post-id="${p.id}">
    <div class="card-body">
      <div class="d-flex align-items-center mb-2">
        <img
          src="${avatarUrl}"
          alt="Avatar"
          class="rounded-circle me-2"
          width="40"
          height="40"
        />
        <div>
          <div class="fw-semibold">
            ${author.nombre || ""} ${author.apellido || ""}
            <span class="text-muted">@${author.username || ""}</span>
          </div>
          <div class="text-muted small">${createdAt}</div>
        </div>
      </div>

      <p class="mb-2">${this._escapeHtml(p.texto || "")}</p>

      ${
        p.imagen_url
          ? `
        <div class="mb-2">
          <img src="${imagenUrl}" class="post-image rounded" alt="Imagen del post" />
```

```
        </div>
          `
          : ""
        }

        <div class="d-flex align-items-center justify-content-between mt-2">
          <div class="d-flex gap-3 small text-muted">
            <span><span class="post-likes-count">${counts.likes}</span> Me gusta</span>
            <span>${counts.comments} comentarios</span>
            <span>${counts.shares} compartidos</span>
          </div>

          <button
            type="button"
            class="btn btn-sm ${liked ? "btn-primary" : "btn-outline-primary"} post-like-btn"
          >
            ${liked ? "Te gusta" : "Me gusta"}
          </button>
        </div>
      </div>
    </article>
  `;
}

afterRender() {
  const btnLike = this.root.querySelector(".post-like-btn");
  if (!btnLike) return;

  btnLike.addEventListener("click", async () => {
```

```
    await this.handleLikeClick(btnLike);
  });
}

async handleLikeClick(btnLike) {
  const post = this.props.post;
  const liked = !!post.liked_by_me;
  const likesCountEl = this.root.querySelector(".post-likes-count");

  try {
    if (!liked) {
      await likePost(post.id);
      post.liked_by_me = true;
      if (typeof post.counts.likes === "number") {
        post.counts.likes += 1;
      }
    } else {
      await unlikePost(post.id);
      post.liked_by_me = false;
      if (typeof post.counts.likes === "number" && post.counts.likes > 0) {
        post.counts.likes -= 1;
      }
    }

    // Actualizar UI (texto, estilo botón y contador)
    if (post.liked_by_me) {
      btnLike.classList.remove("btn-outline-primary");
      btnLike.classList.add("btn-primary");
      btnLike.textContent = "Te gusta";
```

```
    } else {
      btnLike.classList.remove("btn-primary");
      btnLike.classList.add("btn-outline-primary");
      btnLike.textContent = "Me gusta";
    }

    if (likesCountEl) {
      likesCountEl.textContent = String(post.counts.likes);
    }
  } catch (err) {
    alert(err.message || "No se pudo actualizar el Me gusta");
  }
 }

 _escapeHtml(text) {
  const map = {
    "&": "&amp;",
    "<": "&lt;",
    ">": "&gt;",
    '"': "&quot;",
    "'": "&#039;",
  };
  return text.replace(/[&<>"']/g, (m) => map[m]);
 }
}
```

- El propio PostCard llama al backend de likes.
- Actualiza su props.post local y la UI.
- Más adelante puedes refactorizar para que el FeedView controle el estado global de posts.

assets/css/styles.css

```
.post-image {
  width: 100%;         /* que siga siendo responsive */
  max-height: 320px;    /* ajusta a la altura que quieras */
  object-fit: cover;    /* recorta en lugar de deformar */
}
```

Actualizar FeedView para usar PostForm + PostCard + paginación

Archivo: frontend/js/views/FeedView.js

```
// js/views/FeedView.js
import { Component } from "../core/component.js";
import { PostForm } from "../components/PostForm.js";
import { PostCard } from "../components/PostCard.js";
import { listPosts } from "../api/posts.js";

export class FeedView extends Component {
  constructor(props) {
    super(props);
    this.posts = [];
    this.limit = 5;
    this.offset = 0;
    this.hasMore = true;
    this.loading = false;
  }

  template() {
    return `
      <section>
        <h1 class="h3 mb-4">Feed global</h1>
```

```
    <div id="feed-error" class="alert alert-danger d-none"></div>

    <div id="post-form"></div>

    <div id="post-list"></div>

    <div class="d-grid mt-3">
      <button
        id="btn-load-more"
        type="button"
        class="btn btn-outline-secondary"
      >
        Cargar más
      </button>
    </div>
  </section>
 `;
}

afterRender() {
 this.errorBox = this.root.querySelector("#feed-error");
 this.listContainer = this.root.querySelector("#post-list");
 this.loadMoreBtn = this.root.querySelector("#btn-load-more");

 // Montar el formulario de posts
 const formRoot = this.root.querySelector("#post-form");
 if (formRoot) {
  const form = new PostForm({
```

```
      onPostCreated: (post) => this.handlePostCreated(post),
    });
    form.mount(formRoot);
  }

  if (this.loadMoreBtn) {
    this.loadMoreBtn.addEventListener("click", () => this.loadMore());
  }

  // Cargar primera página
  this.loadMore();
}

async loadMore() {
  if (this.loading || !this.hasMore) return;
  this.setLoading(true);

  try {
    const data = await listPosts({
      limit: this.limit,
      offset: this.offset,
    });

    const items = data.items || [];
    const paging = data.paging || {};

    this.offset = paging.next_cursor ?? this.offset;
    this.hasMore = paging.next_cursor != null;
```

```
    this.appendPosts(items);
  } catch (err) {
    this.showError(err.message || "Error al cargar el feed.");
  } finally {
    this.setLoading(false);
  }
}

appendPosts(items) {
  for (const post of items) {
    this.posts.push(post);

    const wrapper = document.createElement("div");
    this.listContainer.appendChild(wrapper);

    const card = new PostCard({ post });
    card.mount(wrapper);
  }

  if (!this.hasMore && this.loadMoreBtn) {
    this.loadMoreBtn.classList.add("d-none");
  }
}

handlePostCreated(post) {
  // Insertar el nuevo post al inicio del listado
  this.posts.unshift(post);

  const wrapper = document.createElement("div");
```

```
    if (this.listContainer.firstChild) {
      this.listContainer.insertBefore(wrapper, this.listContainer.firstChild);
    } else {
      this.listContainer.appendChild(wrapper);
    }

    const card = new PostCard({ post });
    card.mount(wrapper);
  }

  setLoading(isLoading) {
    this.loading = isLoading;
    if (this.loadMoreBtn) {
      this.loadMoreBtn.disabled = isLoading || !this.hasMore;
      this.loadMoreBtn.textContent = isLoading ? "Cargando..." : "Cargar más";
    }
  }

  showError(message) {
    if (!this.errorBox) return;
    this.errorBox.textContent = message;
    this.errorBox.classList.remove("d-none");
  }
}
```

Asegúrate de que FeedView sigue registrado igual:

```
js/core/router.js:
import { FeedView } from "../views/FeedView.js";
// ...
```

```
const routes = {
  "#/login": LoginView,
  "#/register": RegisterView,
  "#/feed": FeedView,
};
```

Y que el router solo permite #/feed si hay sesión (como definimos en Etapa 2).

Pruebas para Etapa 3

1. Loguéate con un usuario válido.
2. Ve a #/feed (se debe redirigir automáticamente tras login).
3. Verifica:
    - El formulario de crear post aparece arriba.
    - Debajo, los posts existentes del backend (si hay).
    - El botón “Cargar más” trae más posts si existen.
4. Crea un post nuevo:
    - Debe aparecer inmediatamente al inicio del feed.
5. Prueba el botón “Me gusta”:
    - Cambia de “Me gusta” (outline) a “Te gusta” (relleno).
    - El contador de likes sube/baja correctamente.
    - En el backend se actualizan likes y liked_by_me.

Etapa 4 — Perfil, Amigos e Interacciones

API para Users, Friends e Interactions

api/users.js

```
// js/api/users.js
import { apiFetch } from "./client.js";

export function getMyProfile() {
  return apiFetch("/users/me");
}

export function updateMyProfile(payload) {
  return apiFetch("/users/me", {
    method: "PATCH",
    body: JSON.stringify(payload),
  });
}

export function updateMyAvatar(formData) {
  return apiFetch("/users/me/avatar", {
    method: "PATCH",
    body: formData,
  });
}

export function getUserPublic(user_uuid) {
  return apiFetch(`/users/${user_uuid}`);
}

export function listUserPosts(user_uuid) {
```

```
  return apiFetch(`/users/${user_uuid}/posts`);
}

export function searchUsers(query, limit = 20, offset = 0) {
  const params = new URLSearchParams({
    q: query,
    limit,
    offset,
  });
  return apiFetch(`/users/search?${params.toString()}`);
}
```

api/friends.js

```
// js/api/friends.js
import { apiFetch } from "./client.js";

export function listFriends(status = "") {
  const qs = status ? `?status=${status}` : "";
  return apiFetch(`/friends${qs}`);
}

export function requestFriend(to_user_uuid) {
  return apiFetch("/friends/request", {
    method: "POST",
    body: JSON.stringify({ to_user_uuid }),
  });
}

export function acceptFriend(user_uuid) {
```

```
  return apiFetch("/friends/accept", {
    method: "POST",
    body: JSON.stringify({ user_uuid }),
  });
}

export function rejectFriend(user_uuid) {
  return apiFetch("/friends/reject", {
    method: "POST",
    body: JSON.stringify({ user_uuid }),
  });
}

export function unfriend(user_uuid) {
  return apiFetch("/friends/unfriend", {
    method: "POST",
    body: JSON.stringify({ user_uuid }),
  });
}
```

api/interactions.js

```
// js/api/interactions.js
import { apiFetch } from "./client.js";

export function listMyInteractions({ user_search = "", limit = 20, offset = 0 }) {
  const params = new URLSearchParams({
    user_search,
    limit,
    offset,
```

```
  });
  return apiFetch(`/me/interactions?${params.toString()}`);
}
```

Vista de Perfil (ProfileView)

Debe soportar dos modos:

- #/profile/me → perfil propio (GET /users/me)
- #/profile/:uuid → perfil público de otro usuario (GET /users/:uuid)

En el router

Agrega rutas:

```
"#/profile/me": ProfileView,
"#/profile": ProfileView, // para evaluar hash dinámico
```

y en navigate() detecta si #/profile/<uuid>.

Perfil – archivo views/ProfileView.js

```
// js/views/ProfileView.js
import { Component } from "../core/component.js";
import { getMyProfile, getUserPublic } from "../api/users.js";
import { listUserPosts } from "../api/users.js";
import { PostCard } from "../components/PostCard.js";
import { API_BASE_URL } from "../api/client.js";
import { store } from "../core/store.js";

export class ProfileView extends Component {
  constructor(props) {
    super(props);
    this.user = null;
    this.posts = [];
```

```
  this.loading = false;
}

template() {
  return `
    <section>
      <h1 class="h4 mb-3">Perfil</h1>

      <div id="profile-error" class="alert alert-danger d-none"></div>

      <div id="profile-info" class="mb-4"></div>

      <h2 class="h5">Publicaciones</h2>
      <div id="profile-posts"></div>
    </section>
  `;
}

async afterRender() {
  this.errorBox = this.root.querySelector("#profile-error");
  this.infoBox  = this.root.querySelector("#profile-info");
  this.postsBox = this.root.querySelector("#profile-posts");

  await this.loadProfile();
}

async loadProfile() {
  this.loading = true;
  const uuid = this.props.user_uuid || "me";
```

```
  try {
    this.user =
      uuid === "me"
        ? await getMyProfile()
        : await getUserPublic(uuid);

    const postsData = await listUserPosts(uuid);
    this.posts = postsData.items || [];

    this.renderProfileInfo();
    this.renderPosts();
  } catch (err) {
    this.errorBox.textContent = err.message;
    this.errorBox.classList.remove("d-none");
  }

  this.loading = false;
}

renderProfileInfo() {
  const u = this.user;
  const avatar = u.avatar_url
    ? API_BASE_URL + u.avatar_url
    : "https://placehold.co/40x40”;

  this.infoBox.innerHTML = `
    <div class="d-flex align-items-center gap-3">
      <img src="${avatar}" class="rounded-circle" width="80" height="80" />
```

```
      <div>
        <div class="fw-bold">${u.nombre} ${u.apellido}</div>
        <div class="text-muted">@${u.username}</div>
      </div>
    </div>
  `;
 }

 renderPosts() {
  this.postsBox.innerHTML = "";

  for (const post of this.posts) {
    const wrapper = document.createElement("div");
    this.postsBox.appendChild(wrapper);

    const card = new PostCard({ post });
    card.mount(wrapper);
  }
 }
}
```

Con esto ya se muestra el perfil + sus posts.

Vista de Amigos (FriendsView)

Debe mostrar:

- Lista de amigos
- Lista de pendientes
- Acciones: pedir amistad, aceptar, rechazar, eliminar amigo
- Buscador de usuarios

Componente FriendItem

Archivo components/FriendItem.js:

```
// js/components/FriendItem.js
import { Component } from "../core/component.js";
import { API_BASE_URL } from "../api/client.js";
import { acceptFriend, rejectFriend, unfriend } from "../api/friends.js";

export class FriendItem extends Component {
  template() {
    const u = this.props.user;
    const status = this.props.status;
    const requested_by_me = this.props.requested_by_me;

    const avatar = u.avatar_url
      ? API_BASE_URL + u.avatar_url
      : "https://placehold.co/80x80”;

    return `
      <div class="d-flex align-items-center justify-content-between border rounded p-2 mb-2">
        <div class="d-flex align-items-center gap-2">
          <img src="${avatar}" width="40" height="40" class="rounded-circle" />
          <div>
            <div class="fw-semibold">${u.nombre} ${u.apellido}</div>
            <div class="text-muted small">@${u.username}</div>
          </div>
        </div>

        <div class="d-flex gap-2">
          ${this.renderButtons(status, requested_by_me)}
```

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

 renderButtons(status, requested_by_me) {
  if (status === "pending") {
    if (requested_by_me) {
      return `<span class="text-muted small">Solicitud enviada</span>`;
    } else {
      return `
        <button class="btn btn-sm btn-success" data-action="accept">Aceptar</button>
        <button class="btn btn-sm btn-danger" data-action="reject">Rechazar</button>
      `;
    }
  }

  if (status === "accepted") {
    return `<button class="btn btn-sm btn-outline-danger" data-
action="remove">Eliminar</button>`;
  }

  return `<span class="text-muted small">${status}</span>`;
 }

 afterRender() {
  const el = this.root;

  el.addEventListener("click", async (ev) => {
```

```
    const btn = ev.target.closest("[data-action]");
    if (!btn) return;

    const action = btn.dataset.action;
    const u = this.props.user;

    try {
      if (action === "accept") await acceptFriend(u.user_uuid);
      if (action === "reject") await rejectFriend(u.user_uuid);
      if (action === "remove") await unfriend(u.user_uuid);

      if (typeof this.props.onChange === "function") {
        this.props.onChange();
      }
    } catch (err) {
      alert(err.message);
    }
   });
  }
}
```

FriendsView

Archivo views/FriendsView.js:

```
// js/views/FriendsView.js
import { Component } from "../core/component.js";
import { listFriends, requestFriend } from "../api/friends.js";
import { searchUsers } from "../api/users.js";
import { FriendItem } from "../components/FriendItem.js";
```

```
export class FriendsView extends Component {
 constructor(props) {
  super(props);
  this.items = [];
  this.searchResults = [];
 }

 template() {
  return `
   <section>
    <h1 class="h4 mb-3">Amigos</h1>

    <div id="friends-error" class="alert alert-danger d-none"></div>

    <h2 class="h5 mt-4">Mis relaciones</h2>
    <div id="friends-list"></div>

    <h2 class="h5 mt-4">Buscar usuarios</h2>
    <input id="friends-search" type="text" class="form-control mb-2" placeholder="Buscar por nombre, usuario, email" />
    <div id="friends-results"></div>
   </section>
  `;
 }

 async afterRender() {
  this.errorBox = this.root.querySelector("#friends-error");
  this.listBox = this.root.querySelector("#friends-list");
  this.resultsBox = this.root.querySelector("#friends-results");
```

```
  await this.loadFriends();

  const searchInput = this.root.querySelector("#friends-search");
  searchInput.addEventListener("input", () => {
    const q = searchInput.value.trim();
    this.search(q);
  });
}

async loadFriends() {
  try {
    const data = await listFriends();
    this.items = data.items || [];
    this.renderFriends();
  } catch (err) {
    this.errorBox.textContent = err.message;
    this.errorBox.classList.remove("d-none");
  }
}

renderFriends() {
  this.listBox.innerHTML = "";
  for (const f of this.items) {
    const wrapper = document.createElement("div");
    this.listBox.appendChild(wrapper);

    const card = new FriendItem({
      user: f,
```

```
      status: f.status,
      requested_by_me: f.requested_by_me,
      onChange: () => this.loadFriends(),
    });

    card.mount(wrapper);
  }
}

async search(q) {
  if (!q) {
    this.resultsBox.innerHTML = "";
    return;
  }

  try {
    const data = await searchUsers(q, 10, 0);
    this.searchResults = data.items || [];
    this.renderSearchResults();
  } catch (err) {
    this.resultsBox.innerHTML = `<div class="text-danger">${err.message}</div>`;
  }
}

renderSearchResults() {
  this.resultsBox.innerHTML = "";

  for (const u of this.searchResults) {
    const wrapper = document.createElement("div");
```

```
    wrapper.className = "d-flex align-items-center justify-content-between border p-2 mb-2";

    wrapper.innerHTML = `
      <div>
        <div class="fw-semibold">${u.nombre} ${u.apellido}</div>
        <div class="text-muted small">@${u.username}</div>
      </div>
      <button class="btn btn-sm btn-primary" data-uuid="${u.user_uuid}">
        Enviar solicitud
      </button>
    `;

    wrapper.querySelector("button").addEventListener("click", async () => {
      try {
        await requestFriend(u.user_uuid);
        await this.loadFriends();
        alert("Solicitud enviada");
      } catch (err) {
        alert(err.message);
      }
    });

    this.resultsBox.appendChild(wrapper);
   }
  }
}
```

Vista de Interacciones (InteractionsView)

Archivo views/InteractionsView.js:

```
// js/views/InteractionsView.js
import { Component } from "../core/component.js";
import { listMyInteractions } from "../api/interactions.js";
import { API_BASE_URL } from "../api/client.js";

export class InteractionsView extends Component {
 constructor(props) {
  super(props);
  this.items = [];
  this.offset = 0;
  this.limit = 20;
  this.hasMore = true;
  this.loading = false;
  this.query = "";
 }

 template() {
  return `
   <section>
    <h1 class="h4 mb-3">Mis interacciones</h1>

    <input id="filter-users" type="text" class="form-control mb-3"
     placeholder="Filtrar por usuario (nombre, username o email)" />

    <div id="interactions-list"></div>

    <div class="d-grid mt-3">
```

```
      <button id="btn-load-more" class="btn btn-outline-secondary">Cargar más</button>
    </div>
  </section>
 `;
}

afterRender() {
 this.listBox = this.root.querySelector("#interactions-list");
 this.loadMoreBtn = this.root.querySelector("#btn-load-more");

 const filter = this.root.querySelector("#filter-users");
 filter.addEventListener("input", () => {
  this.query = filter.value.trim().toLowerCase();
  this.refresh();
 });

 this.loadMoreBtn.addEventListener("click", () => this.loadMore());
 this.refresh();
}

async refresh() {
 this.items = [];
 this.offset = 0;
 this.hasMore = true;
 this.listBox.innerHTML = "";
 await this.loadMore();
}

async loadMore() {
```

```
  if (this.loading || !this.hasMore) return;

  this.loading = true;
  try {
    const data = await listMyInteractions({
      user_search: this.query,
      limit: this.limit,
      offset: this.offset,
    });

    const items = data.items || [];
    const paging = data.paging || {};

    this.offset = paging.next_cursor ?? this.offset;
    this.hasMore = paging.next_cursor != null;

    this.appendItems(items);
  } catch (err) {
    this.listBox.innerHTML += `
      <div class="text-danger">${err.message}</div>
    `;
  }
  this.loading = false;
}

appendItems(items) {
  for (const ev of items) {
    const wrapper = document.createElement("div");
    wrapper.className = "border rounded p-2 mb-2";
```

```
  const other = ev.other_user;
  const avatar = other.avatar_url
    ? API_BASE_URL + other.avatar_url
    : "https://via.placeholder.com/40?text=?";

  wrapper.innerHTML = `
    <div class="d-flex gap-2 mb-2">
      <img src="${avatar}" width="40" height="40" class="rounded-circle" />
      <div>
        <div class="fw-bold">${other.nombre} ${other.apellido}</div>
        <div class="small text-muted">@${other.username}</div>
      </div>
    </div>

    <div class="small text-muted mb-1">${ev.created_at}</div>

    <div><strong>${ev.type}</strong> en post #${ev.post.id}</div>
    ${
      ev.type === "comment"
        ? `<div class="mt-1 border-start ps-2">${ev.comment.texto}</div>`
        : ""
    }
  `;

  this.listBox.appendChild(wrapper);
}

if (!this.hasMore) {
```

```
    this.loadMoreBtn.classList.add("d-none");
   }
  }
}
```

Registrar nuevas vistas en el router

En router.js:

```
import { ProfileView } from "../views/ProfileView.js";
import { FriendsView } from "../views/FriendsView.js";
import { InteractionsView } from "../views/InteractionsView.js";


const routes = {
  "#/login": LoginView,
  "#/register": RegisterView,
  "#/feed": FeedView,
  "#/friends": FriendsView,
  "#/interactions": InteractionsView,
  "#/profile/me": ProfileView,
};
```

Navbar: agregar enlaces

Actualiza Navbar:

```
<li class="nav-item"><a class="nav-link" href="#/feed">Feed</a></li>
<li class="nav-item"><a class="nav-link" href="#/friends">Amigos</a></li>
<li class="nav-item"><a class="nav-link" href="#/interactions">Interacciones</a></li>
<li class="nav-item"><a class="nav-link" href="#/profile/me">Perfil</a></li>
```

Pruebas finales de Etapa 4

1. Perfil
    - Entra a #/profile/me: muestra datos reales del backend.
    - Entra a #/profile/<uuid> de otro usuario desde búsqueda o amigos.
2. Amigos
    - Buscar usuario → enviar solicitud.
    - Ver estado pendiente.
    - Aceptar/rechazar solicitudes.
    - Eliminar a un amigo.
3. Interacciones
    - Dar likes, comentar, compartir → deben aparecer en timeline.
    - Filtrar por nombre/username/email.