UNIVERSIDAD NACIONAL DE SAN AGUSTÍN
FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

INFORME DEL PROYECTO: REPOSITORIO ELECTORAL UNSA GRUPO 07

Integrantes:

Ajra Huacso, Jeans Anthony
Choquehuanca Berna, William Herderson
Caceres Ruiz, Johann Andre
Garambel Marín Fernando Miguel
Luque Condori, Luis Guillermo
Noa Camino Yenaro, Joel

Curso:
Fundamentos de Sistemas de Información

Sección:
A

Profesor:
Mg. Maribel Molina Barriga

AREQUIPA-PERÚ

2025

ÍNDICE

1. Introducción

El presente informe describe el proyecto “Repositorio Electoral UNSA”, una plataforma web desarrollada para la gestión centralizada y transparente de información electoral universitaria incluyendo candidatos, propuestas, publicaciones y participación de votantes mediante una arquitectura moderna que integra un frontend desacoplado con Vite/React y un backend robusto basado en Django y Django REST Framework (DRF). Este proyecto surge como una reimplementación destinada a superar las limitaciones de la versión anterior, la cual operaba bajo un diseño monolítico con deficiencias en escalabilidad y mantenibilidad; por ello, la adopción del stack Vite + Django/DRF responde a la necesidad de modernizar la infraestructura, aplicar el enfoque API-first y asegurar una gestión de datos segura, modular y escalable para el crecimiento futuro del sistema (Humble & Farley, 2010). En este contexto, el presente documento abarca la descripción de la arquitectura del sistema, la metodología de desarrollo adoptada (Ágil Híbrida), la definición de los modelos de datos y la validación de los requisitos funcionales y no funcionales mediante las pruebas iniciales de verificación

2. Marco teórico

El diseño del sistema se sustenta en conceptos fundamentales del desarrollo moderno de software y en una arquitectura de microservicios desacoplada.

a. Arquitectura MVC y Principios de Desacoplamiento

Se adopta el patrón Modelo–Vista–Controlador (MVC), que permite una separación clara entre la lógica de negocio, la presentación y la persistencia (Gamma et al., 1995). Este patrón se acompaña de principios de microservicios, promoviendo:

- Separación clara entre lógica de negocio, presentación y persistencia.
- Escalabilidad horizontal y vertical.
- Independencia del cliente gracias a un CMS Headless.

Además, el sistema sigue un enfoque API-first (Django REST Framework, s. f.), donde el backend expone recursos REST que el frontend consume. Este desacoplamiento cliente/servidor permite que la capa de presentación (Vite/React) se despliegue de forma independiente de la capa de aplicación.

b. Patrones y principios

El sistema sigue un enfoque API-first (Django REST Framework, s. f.), donde el backend expone recursos REST que el frontend consume. El desacoplamiento cliente/servidor permite que el frontend en Vite se despliegue de forma independiente. También se aplican principios DevOps como la contenerización (Docker) y la integración y despliegue continuos (CI/CD), fundamentales para la entrega continua (Humble & Farley, 2010).

c. Backend: Django + Django REST Framework (DRF)

Django (Django Software Foundation, s. f.) fue seleccionado por su ORM potente, su sistema de gestión de autenticación y sus herramientas de migración. DRF fue elegido para exponer los recursos del sistema en formato JSON, permitiendo una serialización robusta y características clave como la paginación y la gestión de permisos. La autenticación del usuario se maneja mediante JWT (JSON Web Tokens).

d. Frontend: Vite

Vite (Vite, s. f.) es la herramienta central del entorno de desarrollo, proporcionando un bundling extremadamente rápido y optimización en producción. El desarrollo de la interfaz se realiza con React con TypeScript y TailwindCSS, comunicándose con el

backend mediante un cliente API para el manejo de la autenticación de tokens y el enrutamiento de vistas

3. Metodología

El proyecto adoptó un enfoque de Metodología Ágil Híbrida, organizada en *sprints* de dos semanas. Este enfoque se basó en los principios de la Guía Scrum (Schwaber & Sutherland, 2020), con roles definidos (Product Owner, Scrum Master, Development Team, DevOps) para garantizar la flexibilidad y la entrega incremental.

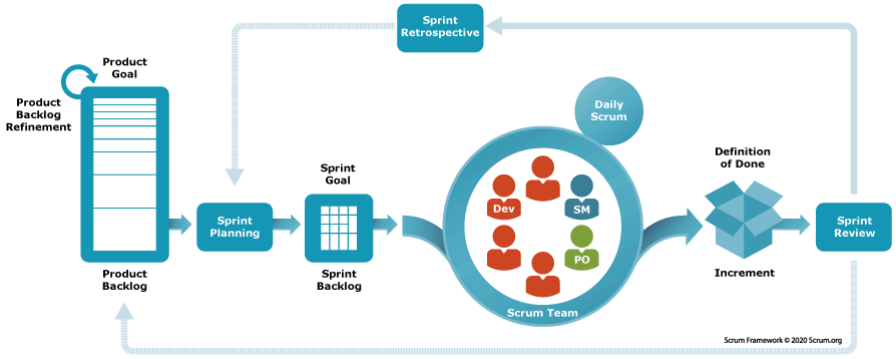


Figura 1. Ciclo de Trabajo en la Metodología Scrum. *Fuente: Adaptado de Scrum.org (Schwaber & Sutherland, 2020).*

a. Herramientas y prácticas

- **Contenerización:** Se utilizaron Dockerfiles y docker-compose para crear entornos consistentes entre el frontend y el backend, facilitando el desarrollo local y el despliegue.
- **Integración y Despliegue Continuos (CI/CD):** Se implementaron GitHub Actions para automatizar los procesos de tests, linters, build y despliegue a los entornos (staging/production).
- **Definición de Terminado (DoD):** Cada entrega se validó mediante un DoD riguroso que incluía la revisión de Pull Requests (PR), el pase de tests unitarios y E2E (End-to-End) para flujos críticos, y la revisión de seguridad y accesibilidad antes del despliegue.

4. Desarrollo / Resultados

a. Arquitectura del sistema

La plataforma se organiza en una arquitectura de tres capas principales que trabajan de forma conjunta para mantener un entorno modular y estable, reflejando el principio de desacoplamiento (Humble & Farley, 2010). El diseño conceptual de esta arquitectura se ilustra en el **Anexo 1** (Diagrama UML de la Arquitectura).

- **Capa cliente (Frontend)**

  Esta capa es el punto de interacción con el usuario. Utiliza Vite, React y TypeScript, enfocándose en la velocidad de desarrollo y rendimiento.

  - **Implementación:** Incluye componentes reutilizables, hooks personalizados y un estado global (Zustand o Redux) para la gestión de datos. Los estilos se manejan con TailwindCSS.
  - **Comunicación y Seguridad:** Se comunica con el backend a través de un cliente API. La autenticación se realiza mediante JWT (JSON Web Tokens), gestionando refresh tokens y almacenamiento seguro (en cookies seguras o almacenamiento protegido).

- **Capa de aplicación (Backend)**

  Basada en Django y Django REST Framework (DRF), esta capa es el núcleo de la lógica de negocio y la seguridad del sistema.

  - **Responsabilidades:** Maneja la Lógica de negocio para usuarios, candidatos, posts, comentarios, reacciones y categorías. Incluye Serializadores, vistas y validaciones por módulo.
  - **Administración:** Gestiona la autenticación con simplejwt y roles (votantes, candidatos, administradores), e integra el Panel administrativo de Django para la supervisión.

- **Capa de datos**

  Esta capa asegura la persistencia, integridad y escalabilidad de la información.

  - **Bases de Datos:** Se usa SQLite para el entorno de desarrollo por su simplicidad, y PostgreSQL para la producción, elegido por su robustez y escalabilidad en entornos de alta concurrencia.
  - **Mantenimiento:** Las Migraciones de Django se utilizan para mantener la consistencia estructural de la base de datos a lo largo del desarrollo.

- **Despliegue y CI/CD**

  El entorno de desarrollo y producción se basa en Contenedores con Docker y docker-compose. La gestión de calidad y despliegue se realiza mediante GitHub Actions para pruebas, análisis, build y despliegues automáticos.

b. UML de la Arquitectura

El diseño lógico del sistema, que define los elementos funcionales del repositorio, se detalla en el **Anexo 1** (Diagrama UML de la Arquitectura). Este diagrama ilustra las interconexiones entre las principales entidades del backend: Usuario, Candidato, Post, Comentario, Reacción y Categoría, incluyendo las asociaciones clave como la relación múltiple entre publicaciones y categorías.

c. Modelo de Base de Datos

La estructura de persistencia del sistema se basa en un diseño coherente de entidades interconectadas. Este modelo, detallado visualmente en el **Anexo 2** (Modelo Entidad-Relación), garantiza la integridad de los datos y la escalabilidad del repositorio electoral.

Las principales entidades y sus características funcionales son:

- **Usuario y Candidato**
  - El modelo base User de Django se utiliza y extiende para incluir el campo rol (votante, candidato, administrador).
  - La entidad Candidate extiende al usuario, añadiendo atributos específicos como biografía, facultad, programa, foto y campos JSON para permitir la adición flexible de atributos futuros.
- **Publicaciones (Post)**
  - Contienen el título, contenido, estado y visibilidad de los anuncios o propuestas.
  - Pueden vincularse a múltiples categorías mediante una relación muchos a muchos.
- **Categorías**
  - Definen la taxonomía del contenido mediante atributos de nombre, descripción e identificador legible, facilitando el filtrado y la organización.

- Los Comentarios están vinculados a un autor y a una publicación, con capacidad de jerarquía para implementar respuestas.
- Las Reacciones (como like, support, insight) son simples, pero están diseñadas con restricciones para evitar duplicados por usuario, asegurando que cada votante interactúe una sola vez por post.

El modelo se refuerza con la aplicación de índices en campos de consulta frecuente y reglas adecuadas en claves foráneas para controlar el comportamiento de los datos ante eliminaciones, optimizando el rendimiento y manteniendo la consistencia estructural.

d. Primeras Pruebas de Verificación

Las pruebas iniciales se realizaron de manera manual y se enfocaron en validar los flujos de usuario más críticos y la integridad de la Capa de Aplicación. La documentación estructurada de los casos de prueba ejecutados y sus resultados detallados se presenta en el **Anexo 3.**

- **Autenticación Exitosa:** Se confirmó el funcionamiento correcto del Inicio de Sesión con JWT y la emisión/actualización de tokens, validando las decisiones de seguridad de la arquitectura.
- **Gestión de Identidad:** El Registro de votantes y candidatos (incluyendo sus campos extendidos) y las validaciones de formularios funcionaron sin inconvenientes, demostrando la integridad del modelo de datos inicial.
- **Visualización Base:** Se verificó la visualización estable de publicaciones y de la lista de candidatos, asegurando que la Capa Cliente (Frontend) consume los endpoints de la API correctamente.

Se identificó la necesidad futura de optimizar los endpoints a medida que aumente la carga de datos y el número de usuarios. Esta observación subraya la importancia de implementar técnicas de caching y paginación avanzada para garantizar el cumplimiento continuo del Requerimiento No Funcional de rendimiento (< 2 segundos), especialmente en entornos de producción.

e. Requerimientos del Sistema

Los requerimientos que guiaron el desarrollo del sistema se categorizan y se resumen en la **Tabla 1**, la cual vincula las funcionalidades solicitadas con las especificaciones de calidad del sistema.

Tabla 1. Resumen de Requerimientos Funcionales y No Funcionales

Categoría	Requerimiento Funcional (RF)	Requerimiento No Funcional (RNF)
Gestión de Contenido	Creación y administración de publicaciones, categorías y el uso del panel administrativo.	**Seguridad:** Implementación de JWT, protección CSRF, validaciones y limitación de peticiones.
Interacción	Interacción por comentarios y reacciones.	**Rendimiento:** Tiempos de respuesta óptimos (**<2 segundos** en operaciones CRUD).
Identidad	Gestión completa de usuarios (registro, autenticación, roles, recuperación de contraseña).	**Usabilidad:** Diseño responsivo y accesible.
Infraestructura	Integración de servicios (API).	**Escalabilidad y Observabilidad:** Mediante contenedores (Docker), bases de datos robustas y métricas operativas.

f. Modernización de infraestructura y estrategia Serveless

Para cumplir con los estándares de escalabilidad y rendimiento global, se implementaron tres mejoras arquitectónicas críticas en el despliegue del sistema:

- **Almacenamiento Distribuido con CDN:** Se desacopló el almacenamiento de archivos estáticos y media del servidor principal, implementando una estrategia de Almacenamiento en la Nube integrada con una red de distribución de contenidos (CDN).

  La entrega de estos contenidos se realiza a través de una Red de Distribución de Contenidos (CDN), reduciendo la latencia y garantizando una alta

disponibilidad de los recursos visuales independientemente de la ubicación del usuario.

- **Transición a Arquitectura Serveless y Edge Computing:** Mediante el uso de adaptadores especializados para Django, se transformó la arquitectura monolítica tradicional en un modelo Serverless (sin servidor).

  Esta adaptación permite desplegar el backend como funciones ejecutables bajo demanda en plataformas líderes como AWS Lambda, Cloudflare Workers y Google Cloud Functions.

  Se migró el motor de ejecución de JavaScript de Node.js a un Edge Runtime. Este entorno es significativamente más ligero y está optimizado para ejecutarse en el borde de la red (Edge Computing), minimizando los tiempos de arranque en frío (cold starts) y mejorando la respuesta del sistema.
- **Automatización del ciclo de Vida (CI/CD):** Para agilizar la modernización del desarrollo, se integraron herramientas de Integración y Despliegue Continuo (CI/CD) de estándar industrial. Se utilizan GitHub Workflows y Jenkins para orquestar la creación de contenedores Docker, automatizando las pruebas y el despliegue inmediato de nuevas versiones a la infraestructura Serverless. La representación gráfica de este flujo de despliegue y arquitectura de nube se detalla en el **Anexo 5.**

g. Planificación Ágil (Sprints)

El desarrollo del proyecto se organizó en cinco ciclos iterativos, cada uno con una duración de dos semanas, siguiendo la metodología Ágil Híbrida definida previamente. La **Tabla 2** detalla el enfoque y los entregables clave de cada *sprint*.

Tabla 2. Planificación de Sprints del Proyecto.

Sprint	Duración	Objetivos y Entregables Clave

Sprint 1	2 semanas	Infraestructura base (Docker, CI/CD) y diseño inicial de la base de datos.
Sprint 2	2 semanas	Identidad del usuario, autenticación (JWT) y roles (votante/candidato).
Sprint 3	2 semanas	Módulos de Publicaciones (*Posts*) y Categorías (Endpoints REST para CRUD).
Sprint 4	2 semanas	Interacciones (Comentarios y Reacciones) y desarrollo del Panel Administrativo.
Sprint 5	2 semanas	Pruebas funcionales E2E, mejoras de usabilidad y optimización inicial de rendimiento.

5. Discusión

La implementación del proyecto Repositorio Electoral UNSA mediante la combinación de Vite + Django demostró ser viable y un desacoplamiento eficiente que valida el patrón API-first definido en el Marco Teórico. Django proporcionó la solidez necesaria en la gestión de datos y seguridad (Django Software Foundation, s. f.), mientras que Vite aceleró la productividad y la experiencia de desarrollo del frontend.

- **Interpretación de Resultados:** Las pruebas de verificación iniciales (Anexo 3) resultaron exitosas para los flujos críticos. El éxito en la autenticación JWT valida directamente las decisiones de arquitectura de seguridad. La separación de responsabilidades facilitada por la arquitectura de tres capas permitió el desarrollo simultáneo de cliente y servidor, cumpliendo con el objetivo de independencia del cliente.
- **Retos:** Los principales retos se concentraron en la configuración de seguridad y despliegue, específicamente en el ajuste de CORS y la protección contra

ataques comunes. Estos desafíos se abordaron exitosamente mediante la aplicación de medidas de seguridad recomendadas por DRF, incluyendo el uso de cookies seguras y la limitación de peticiones, lo cual refuerza el requisito no funcional de Seguridad establecido en la Tabla 1.

- **Conexión con Teoría:** El modelo de desarrollo Ágil Híbrido, con ciclos cortos (*sprints*), permitió una rápida adaptación ante la complejidad de la configuración inicial de infraestructura (Docker/CI/CD), cumpliendo con el principio de respuesta al cambio inherente a la agilidad (Schwaber & Sutherland, 2020).
- **Preparación para el Crecimiento:** Aunque las pruebas iniciales fueron satisfactorias, la observación de la necesidad de optimizar los endpoints a futuro subraya que, si bien la arquitectura es escalable, el rendimiento bajo alta carga aún requiere validación y la implementación de técnicas de caching y paginación avanzadas para sostener el crecimiento.

6. Conclusiones

Los principales hallazgos y la relevancia del proyecto Repositorio Electoral UNSA se sintetizan en los siguientes puntos:

- **Viabilidad Tecnológica:** La adopción del stack Vite + Django resultó ser viable y beneficiosa. Django facilita una gestión robusta de datos y usuarios, mientras que Vite mejoró significativamente la productividad y la experiencia de desarrollo del cliente.
- **Funcionamiento Comprobado:** Las pruebas iniciales comprobaron el funcionamiento correcto de los módulos clave del sistema, especialmente los de autenticación (JWT) y navegación, validando las decisiones de seguridad y la correcta interacción entre las capas.
- **Potencial de Crecimiento:** La arquitectura desacoplada propuesta ofrece bases sólidas para escalar y extender el sistema en futuras etapas, lo cual permite responder a las necesidades de crecimiento del Repositorio Electoral.

7. Recomendaciones

Las siguientes recomendaciones se proponen para mejorar el desempeño, completar la validación y proyectar la continuidad y aplicación futura del Repositorio Electoral UNSA. En primer lugar, es crucial abordar la observación crítica sobre el rendimiento mediante la implementación de mecanismos de caching (utilizando Redis o Memcached) y paginación avanzada en la Capa de Aplicación (Django REST Framework). Esto es fundamental para garantizar el cumplimiento continuo del Requerimiento No Funcional de rendimiento (< 2 segundos) en entornos de producción.En segundo lugar, se debe aumentar el rigor en las pruebas y el proceso CI/CD. Se recomienda formalizar las pruebas de carga (Load Testing) para obtener datos cuantitativos y validar la escalabilidad bajo condiciones de tráfico real. Adicionalmente, se debe extender la cobertura de GitHub Actions para incluir tests E2E (End-to-End) automáticos que cubran los flujos críticos del sistema, mejorando la confiabilidad de la entrega continua.Finalmente, la arquitectura actual ofrece bases sólidas para la extensión funcional. Se recomienda priorizar la implementación del módulo de Votación Electrónica Segura, lo cual permitirá aplicar el sistema directamente a los procesos electorales de la universidad. Esta extensión requerirá la investigación e integración de tecnologías de seguridad especializadas para asegurar la transparencia, el anonimato y la no manipulación del voto universitario.

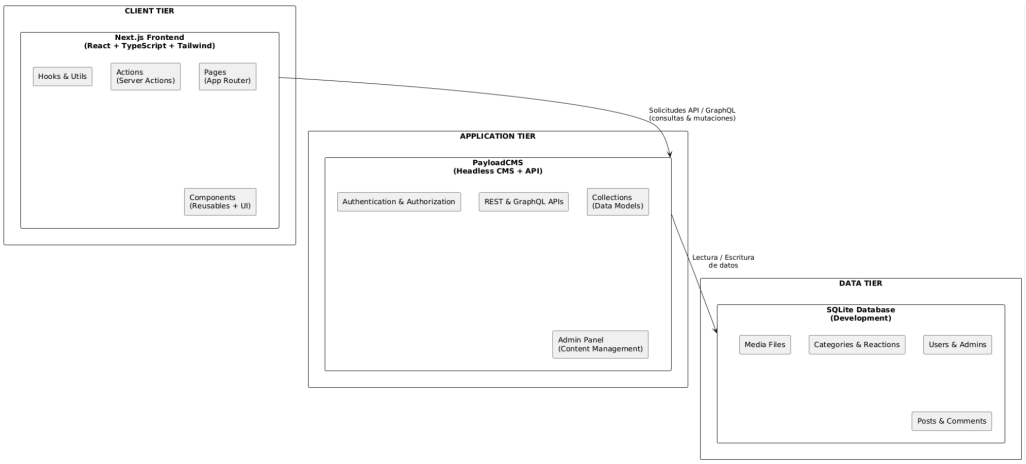
8. Bibliografía

- Schwaber, K., & Sutherland, J. (2020). *La Guía de Scrum*. Scrum.org.
- **Django Software Foundation.** (s. f.). *The Web framework for perfectionists with deadlines*. Recuperado de https://www.djangoproject.com/
- **Humble, J., & Farley, D.** (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional.
- **Gamma, E., Helm, R., Johnson, R., & Vlissides, J.** (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.
- **React.** (s. f.). *The library for web and native user interfaces*. Recuperado de https://react.dev/
- **Vite.** (s. f.). *Next Generation Frontend Tooling*. Recuperado de https://vitejs.dev/

9. Anexos

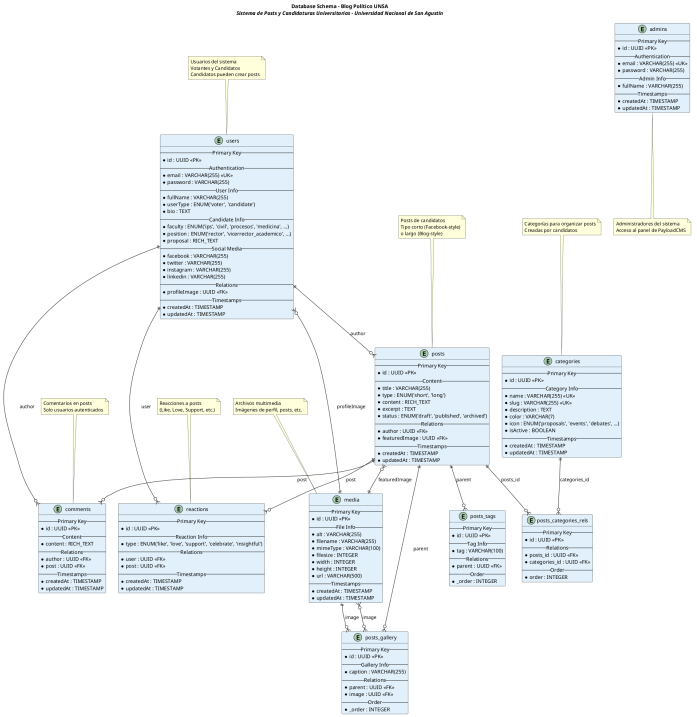
Anexo 1: Diagrama UML de la Arquitectura del Sistema

Descripción: Este diagrama ilustra la arquitectura lógica del sistema, mostrando la interacción entre las principales entidades del *backend* (Usuario, Candidato, Post, Comentario, Reacción y Categoría) y su relación con las capas Cliente y Aplicación.



Anexo 2: Modelo Entidad-Relación (Base de Datos)

Descripción: Este diagrama ilustra el diseño detallado de la base de datos (PostgreSQL), presentando las entidades principales (User, Candidate, Post, Comentarios, Reacciones, etc.) y sus interconexiones. El modelo refleja la estructura de persistencia del sistema, garantizando la integridad de los datos y la eficiencia de las consultas.



Anexo 3: Documentación de Pruebas de Verificación Iniciales

Descripción: Este anexo presenta la tabla de resultados de la ejecución de los casos de prueba manuales para la fase inicial del proyecto. Los resultados detallan la validación del cumplimiento de los requerimientos funcionales críticos (RF) del sistema, sirviendo como evidencia de la correcta implementación y funcionalidad de la Capa de Aplicación y la Capa Cliente.

ID Caso	Funcionalidad	Descripción de la Prueba	Resultado Obtenido	Evidencia
CP-001	Inicio de sesión	Permite acceder al sistema mediante correo institucional y contraseña válidos.	Correcto. Se valida el acceso y la recepción de JWT.	**Ver Anexo 4, Fig. 1**
CP-002	Registro de	Permite crear nuevas cuentas seleccionando el tipo de usuario (votante o	Correcto. Usuario creado y rol asignado	**Ver Anexo**

	cuenta	candidato).	en la base de datos.	**4, Fig. 2**
CP-003	Form. Extendido Candidatos	Permite a los candidatos ingresar campos adicionales: facultad, propuesta electoral y enlaces.	Correcto. Los datos específicos del candidato se persisten sin errores de validación.	**Ver Anexo 4, Fig. 3**
CP-004	Página principal	Presentación de la cabecera con secciones principales (Inicio, Candidatos, Posts, Buscar).	Correcto. Interfaz estable y navegación de secciones disponible.	**Ver Anexo 4, Fig. 4**
CP-005	Visualización de Candidatos	Acceso a la vista de candidatos registrados.	Correcto. La vista carga sin errores, lista para recibir datos de la API.	**Ver Anexo 4, Fig. 5**

Anexo 4: Vistas de Interfaz y Evidencia de Pruebas

Descripción: Este anexo contiene las capturas de pantalla de la interfaz de usuario del sistema (Mockups de Figma o capturas de la implementación inicial), que sirven como evidencia visual del correcto funcionamiento de las funcionalidades validadas en la tabla del Anexo 3.

Figura 1. Pantalla de inicio de sesión funcional.

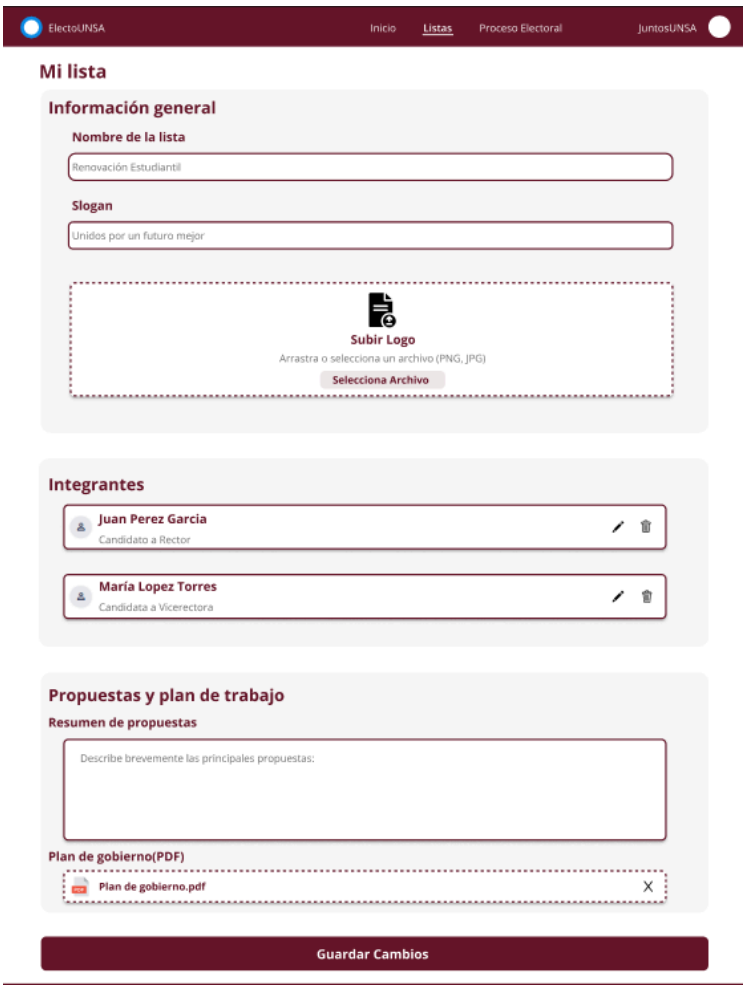


Figura 2. Formulario de registro funcional.

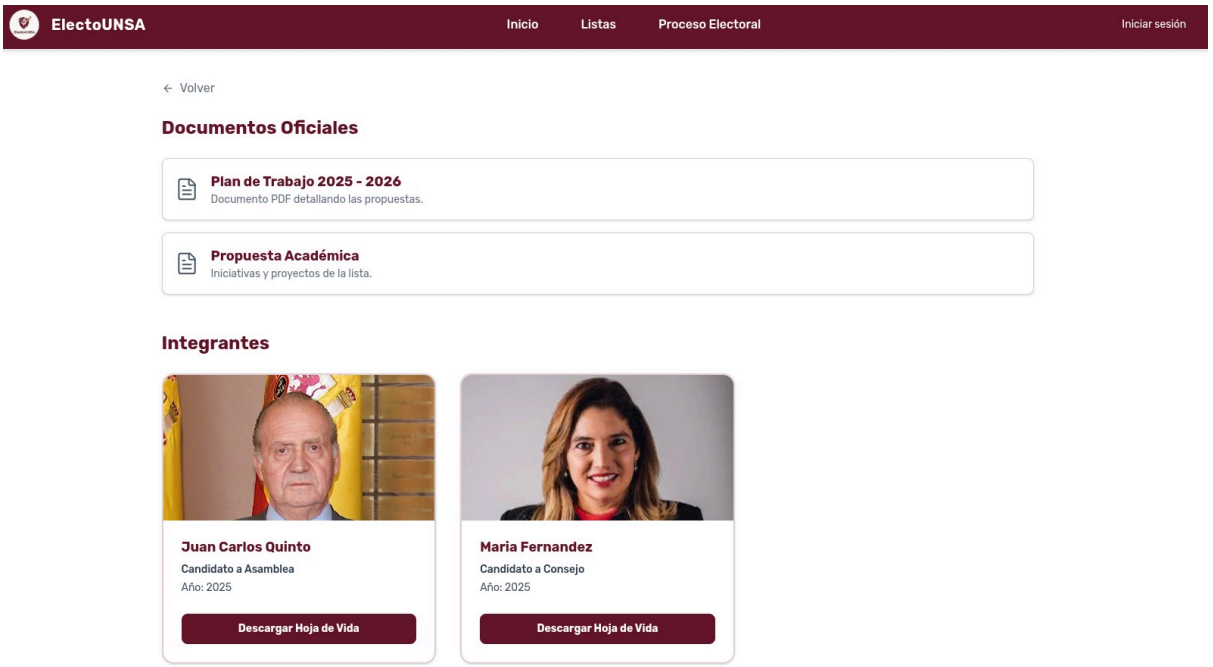


Figura 3. Formulario de registro con campos específicos para candidatos.

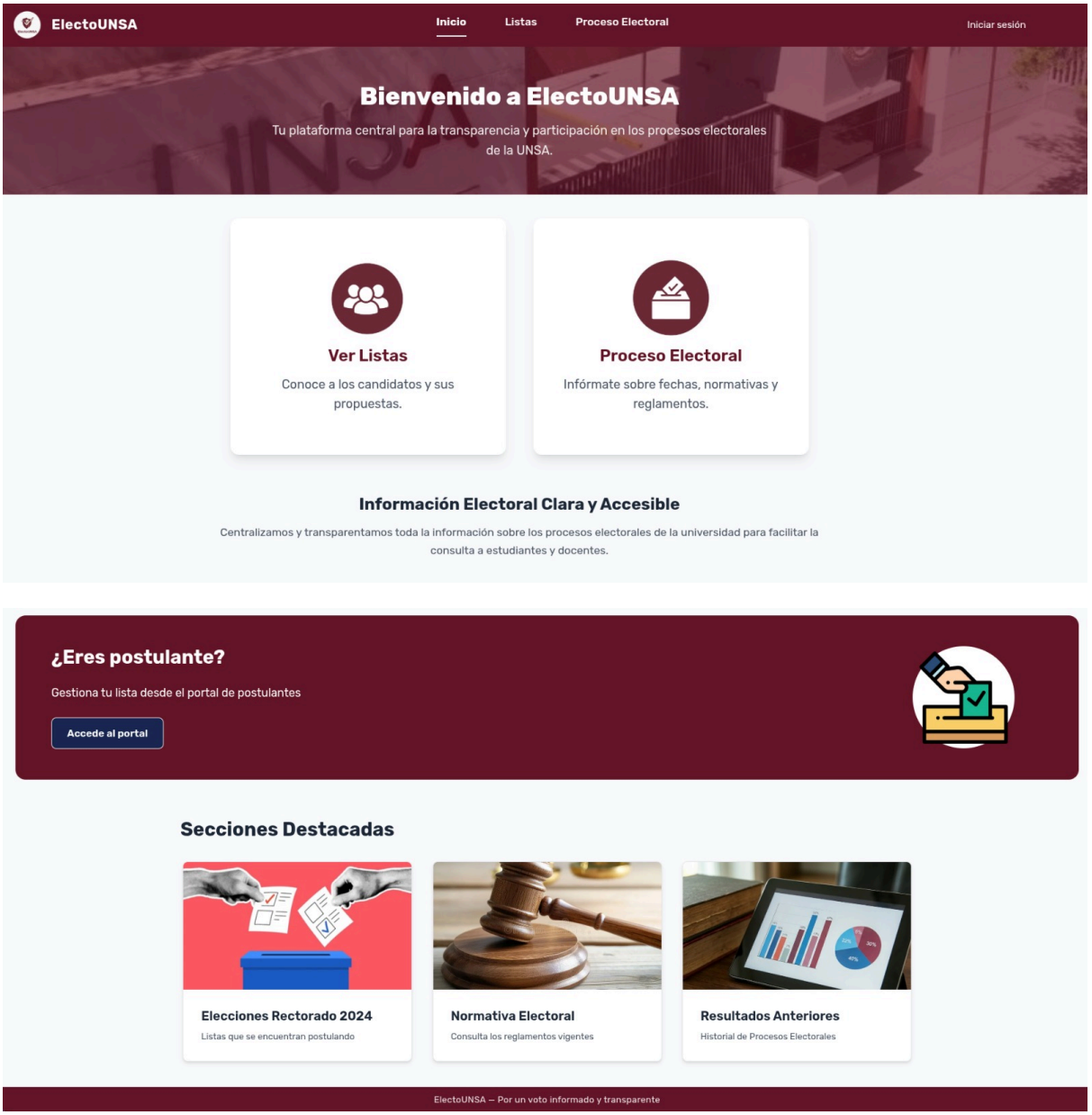


Figura 4. Página Principal.

Anexo 5: Diagrama de Infraestructura Serverless y Flujo CI/CD

Descripción: Esquema técnico que ilustra la arquitectura de despliegue moderna del Repositorio Electoral. El diagrama detalla el flujo desde la confirmación del código (GitHub/Jenkins), pasando por la contenerización (Docker), hasta la ejecución en entornos Serverless (AWS/Cloudflare) y la distribución de contenido estático mediante CDN y almacenamiento en la nube.