Documentación: ChemsTools Pre-Alpha 0.0.3 - Prototipo Conceptual

Versión: 0.0.3

Fecha: 9 de julio de 2025

1. Resumen de Objetivos

Esta fase se centra en materializar los conceptos definidos en las versiones 0.0.1 y 0.0.2. El objetivo principal es desarrollar un prototipo funcional mínimo que permita validar la arquitectura, probar las tecnologías seleccionadas en casos de uso críticos y establecer el flujo de trabajo de desarrollo y construcción del proyecto.

---

2. Implementación del Prototipo Mínimo

El prototipo se enfocará en implementar un esqueleto funcional de la aplicación, conectando el frontend con el backend a través de las interfaces API definidas. Las funcionalidades por desarrollar son seleccionadas para abordar los riesgos técnicos más altos y validar los componentes centrales de la arquitectura.

2.1. Alcance del Prototipo:

- **Sistema de Autenticación (RF1):**
    - **Objetivo:** Validar el flujo de JWT entre frontend y backend.
    - **Implementación:** Crear endpoints POST /auth/register y POST /auth/login en el backend de Django. En el frontend de React, desarrollar componentes básicos para un formulario de registro e inicio de sesión que interactúen con estos endpoints y almacenen el token en el estado global (Redux).
- **Editor de Moléculas 2D (RF2):**
    - **Objetivo:** Probar la integración y rendimiento de la biblioteca Kekule.js.
    - **Implementación:** Crear un componente React que renderice el editor molecular. La funcionalidad se limitará a dibujar y validar estructuras simples. No se requiere guardado en base de datos en esta etapa.
- **Visualizador 3D (RF3):**
    - **Objetivo:** Abordar el riesgo técnico **RT1** (Rendimiento insuficiente del visualizador 3D).
    - **Implementación:** Desarrollar un componente React que utilice Three.js para renderizar un modelo molecular 3D a partir de datos estáticos (hardcodeados). Se probarán la rotación y el zoom interactivos.

- **Cálculo de Propiedades Básicas (RF4):**
    - **Objetivo:** Validar la integración del núcleo químico (**RDKit**) y los cálculos en segundo plano para mitigar el riesgo **RT3**.
    - **Implementación:** Crear un endpoint POST /calculations/properties/basic que acepte datos estructurales, los procese con RDKit en el backend de Python y devuelva la masa molecular y la fórmula.

---

3. Prueba de Conceptos Técnicos Críticos

Se realizarán pruebas de concepto (PoC) específicas para validar las decisiones tecnológicas y mitigar los riesgos identificados.

- **PoC 1: Rendimiento del Renderizado 3D (Mitigación RT1)**
    - **Prueba:** Renderizar moléculas de complejidad variable (de 10 a 500 átomos) en el componente del visualizador 3D.
    - **Métrica de Éxito:** Mantener un mínimo de 30 FPS en los navegadores objetivo (Chrome, Firefox) durante la manipulación interactiva.
- **PoC 2: Integración de API Externa (Mitigación RT2)**
    - **Prueba:** Crear un script en el backend que realice una búsqueda simple en la API de PubChem por nombre de compuesto y procese la respuesta.
    - **Métrica de Éxito:** Conexión exitosa y correcta serialización de los datos recibidos. Se validará el sistema de caché básico con Redis para evitar llamadas repetidas.
- **PoC 3: Cálculo Asíncrono (Mitigación RT3)**
    - **Prueba:** Implementar el endpoint de cálculo de propiedades utilizando un worker o una tarea asíncrona en Django para no bloquear el hilo principal.
    - **Métrica de Éxito:** El servidor debe seguir aceptando otras peticiones mientras se realiza un cálculo, y la respuesta del cálculo debe ser inferior a 2 segundos para moléculas de tamaño medio.

---

4. Validación de la Arquitectura Propuesta

El desarrollo del prototipo servirá como la validación práctica de la arquitectura definida en

Pre-Alpha 0.0.2.

- **Validación del Flujo de Datos Cliente-Servidor:** Se verificará el flujo de datos principal: Interfaz de Usuario (React) → Acción (Redux) → Petición a API Gateway (Django REST) → Servicio Backend → Respuesta → Actualización de estado (Redux) → Actualización de UI.
- **Validación de Patrones de Diseño:**
    - **Frontend:** Se aplicará **Atomic Design** para los componentes de UI (ej. botones, inputs) y el patrón

Container/Presentational para separar la lógica (ej. MoleculeViewerContainer) del renderizado.

    - **Backend:** Se implementará el **Repository Pattern** para abstraer las consultas a la base de datos de usuarios y el

Factory Method para la creación de objetos de cálculo.

- **Validación de la Estructura del Repositorio:** El código del prototipo se organizará siguiendo la estructura de directorios definida en el documento 0.0.1, confirmando su idoneidad.

---

5. Integración de Herramientas de Desarrollo

Se configurará y verificará el ecosistema de herramientas para asegurar un entorno de desarrollo consistente y de alta calidad.

- **Contenerización:** Se crearán los archivos docker-compose.yml y los Dockerfiles para los servicios de frontend, backend y base de datos (PostgreSQL). El entorno de desarrollo local se levantará con un solo comando (

docker-compose up).

- **Linters y Formateadores:**
    - **Frontend:** Se configurará ESLint con las reglas de Airbnb y Prettier para garantizar un estilo de código TypeScript/React uniforme.
    - **Backend:** Se integrarán Black y Flake8 en el entorno de desarrollo de Python para el formateo y linting automáticos.
- **Hooks de Pre-commit:** Se utilizará husky o una herramienta similar para ejecutar los linters y formateadores antes de cada commit, previniendo la introducción de código que no cumpla los estándares.

---

6. Configuración Inicial del Sistema de Build

Se establecerán los flujos de trabajo iniciales de Integración Continua (CI) para automatizar la construcción y las pruebas del prototipo.

- **Plataforma de CI/CD:** Se configurará **GitHub Actions** en el repositorio del proyecto.
- **Workflow de Integración Continua:**
    - **Disparador:** El workflow se ejecutará en cada push a la rama develop y en la creación de Pull Requests hacia develop.
    - **Pasos del Workflow:**
        1. **Checkout:** Clonar el código del repositorio.
        2. **Setup Environment:** Configurar los entornos de Node.js y Python.
        3. **Install Dependencies:** Ejecutar npm install para el frontend y pip install para el backend.
        4. **Lint & Format Check:** Ejecutar ESLint, Prettier, Black y Flake8 para verificar los estándares de código.
        5. **Build:** Construir la aplicación de producción de Next.js (npm run build) para asegurar que no hay errores de compilación.
        6. **Run Tests (Platzhalter):** Se incluirá un paso para ejecutar pruebas (aún sin tests implementados), que pasará por defecto. La implementación real de las pruebas unitarias comenzará en la fase 0.1.X.

---

7. Próximos Pasos

Al finalizar la fase 0.0.3, el equipo habrá logrado un prototipo tangible y validado las suposiciones técnicas más críticas. El siguiente paso será avanzar a la

Pre-Alpha 0.1.X - Fase de Estructura Base, donde se comenzará a construir el framework básico del sistema sobre los cimientos validados en esta etapa.