Documentación Técnica: Fase de Estructura Base (Pre-Alpha 0.1.X)

Proyecto: ChemsTools

Versión: 0.1.0

Fecha: 9 de julio de 2025

1. Introducción

Esta documentación establece las guías, estándares y diagramas de componentes para la fase **Pre-Alpha 0.1.X**. El objetivo de esta fase es construir el framework básico de la aplicación, desarrollar los componentes centrales y realizar la integración inicial. Este documento servirá como manual para todos los desarrolladores del proyecto.

2. Manual de Desarrollo

Esta sección describe el flujo de trabajo estándar para desarrollar ChemsTools.

2.1. Configuración del Entorno

La configuración inicial del entorno de desarrollo está automatizada. Para configurar un nuevo entorno, sigue estos pasos:

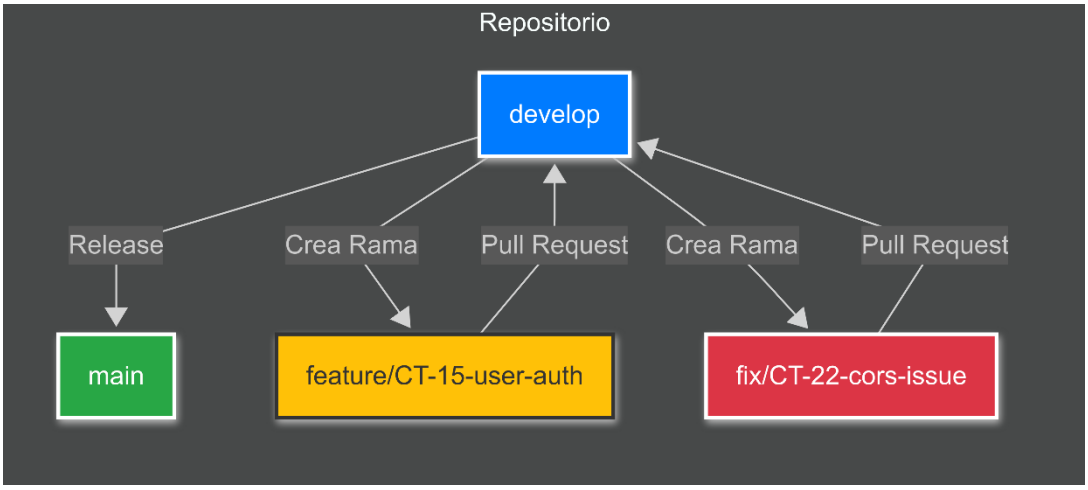
1. Clona el repositorio desde GitHub.
2. Ejecuta el script de Python sutup.py ubicado en la raíz del proyecto.
3. Sigue las instrucciones detalladas en el archivo setup_instructions.txt para la configuración manual final (variables de entorno y conexión a la base de datos).
4. Levanta todo el entorno con el comando:

```
docker-compose up –build
```

2.2. Flujo de Trabajo con Git (Git Flow)

Utilizamos un modelo de branching simplificado basado en Git Flow para mantener el repositorio organizado.

- **main**: Esta rama contiene el código de producción estable. Solo se fusiona desde develop al preparar un nuevo lanzamiento.
- **develop**: Es la rama principal de desarrollo. Contiene las últimas funcionalidades integradas y debe mantenerse estable.
- **Ramas de Funcionalidad (feature/...)**: Todo nuevo desarrollo se realiza en una rama de este tipo.
    - Formato: feature/ID-tarea-descripcion-corta (ej. feature/CT-15-user-authentication).
    - Se crean a partir de develop.
    - Una vez completadas, se fusionan de nuevo a develop a través de un Pull Request.
- **Ramas de Corrección (fix/...)**: Se usan para corregir errores urgentes.
    - Formato: fix/ID-tarea-descripcion-corta (ej. fix/CT-21-login-button-bug).
    - Se crean a partir de develop o main (si es un hotfix) y se fusionan de vuelta a ambas.



2.3. Proceso de Pull Request (PR)

1. **Crea tu rama** a partir de develop.
2. **Realiza tus commits** siguiendo los estándares de mensajes de commit.
3. **Actualiza tu rama** con los últimos cambios de develop (git pull origin develop) antes de crear el PR para resolver conflictos localmente.
4. **Crea el Pull Request** en GitHub apuntando a la rama develop.
5. **Asegúrate** de que todas las pruebas automatizadas (CI) pasen correctamente.
6. **Solicita una revisión** de al menos otro miembro del equipo.
7. Una vez aprobado y con las pruebas en verde, el PR puede ser **fusionado (squash and merge)** en develop.

3. Estándares de Código

Para garantizar la calidad, legibilidad y mantenibilidad del código, se deben seguir los siguientes estándares.

3.1. Principios Generales

- **DRY (Don't Repeat Yourself):** Evita la duplicación de código. Encapsula la lógica reutilizable en funciones o componentes.
- **KISS (Keep It Simple, Stupid):** Prefiere soluciones simples y claras sobre las complejas.

3.2. Backend (Python / Django)

- **Formato:** Se sigue el estándar **PEP 8**. El formateo es automático y obligatorio a través de Black. El linting se realiza con Flake8.
- **Nomenclatura:**
    - Variables y funciones: snake_case (ej. molecular_weight).
    - Clases: PascalCase (ej. MolecularWeightCalculatorView).
    - Constantes: UPPER_SNAKE_CASE (ej. MAX_RETRIES).
- **Tipado (Type Hints):** Todas las funciones y métodos deben incluir type hints para mejorar la claridad y permitir el análisis estático.
- **Estructura de Apps:** Cada funcionalidad principal de Django debe residir en su propia "app" (ej. users, molecules, calculations).

3.3. Frontend (TypeScript / React)

- **Formato:** Se sigue la guía de estilo de **Airbnb para JavaScript/React**, reforzada por ESLint y formateada automáticamente por Prettier.
- **Nomenclatura:**
    - Variables y funciones: camelCase (ej. molecularWeight).
    - Componentes y tipos: PascalCase (ej. MolecularWeightCalculator, type UserProfile).
    - Archivos de componentes: PascalCase (ej. HealthCheck.tsx).
- **Estructura de Componentes:** Seguimos una filosofía inspirada en **Atomic Design**:
    - src/components/common/: Átomos y Moléculas. Componentes de UI puros y reutilizables (ej. Button, Input, Card).
    - src/components/features/: Organismos. Componentes complejos que agrupan otros componentes y manejan lógica de una funcionalidad específica (ej. MolecularWeightCalculator).
    - src/app/: Páginas. Ensamblan los organismos para construir una vista completa.
- **Manejo de Estado:** El estado global se gestiona con **Redux Toolkit**. Se debe usar el patrón de slices para organizar la lógica del estado por funcionalidad.

3.4. Mensajes de Commit

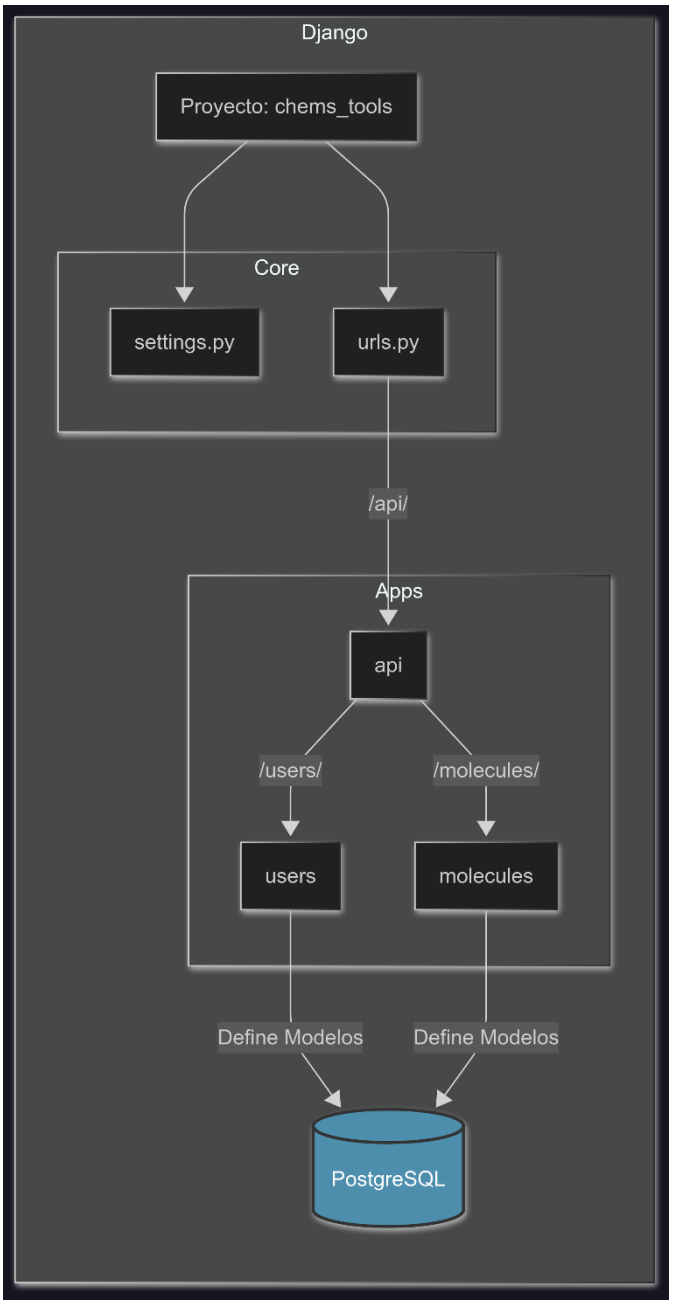
Usamos el estándar de **Conventional Commits**. Esto facilita la generación de changelogs y la comprensión del historial.

- **Formato:** tipo(ámbito): mensaje
- **Ejemplos:**
    - feat(api): add user registration endpoint
    - fix(frontend): correct molecular weight display format
    - docs(readme): update setup instructions
    - refactor(backend): simplify calculation service
- **Tipos comunes:** feat, fix, docs, style, refactor, test, chore.

4. Diagramas de Componentes (Fase 0.1.X)

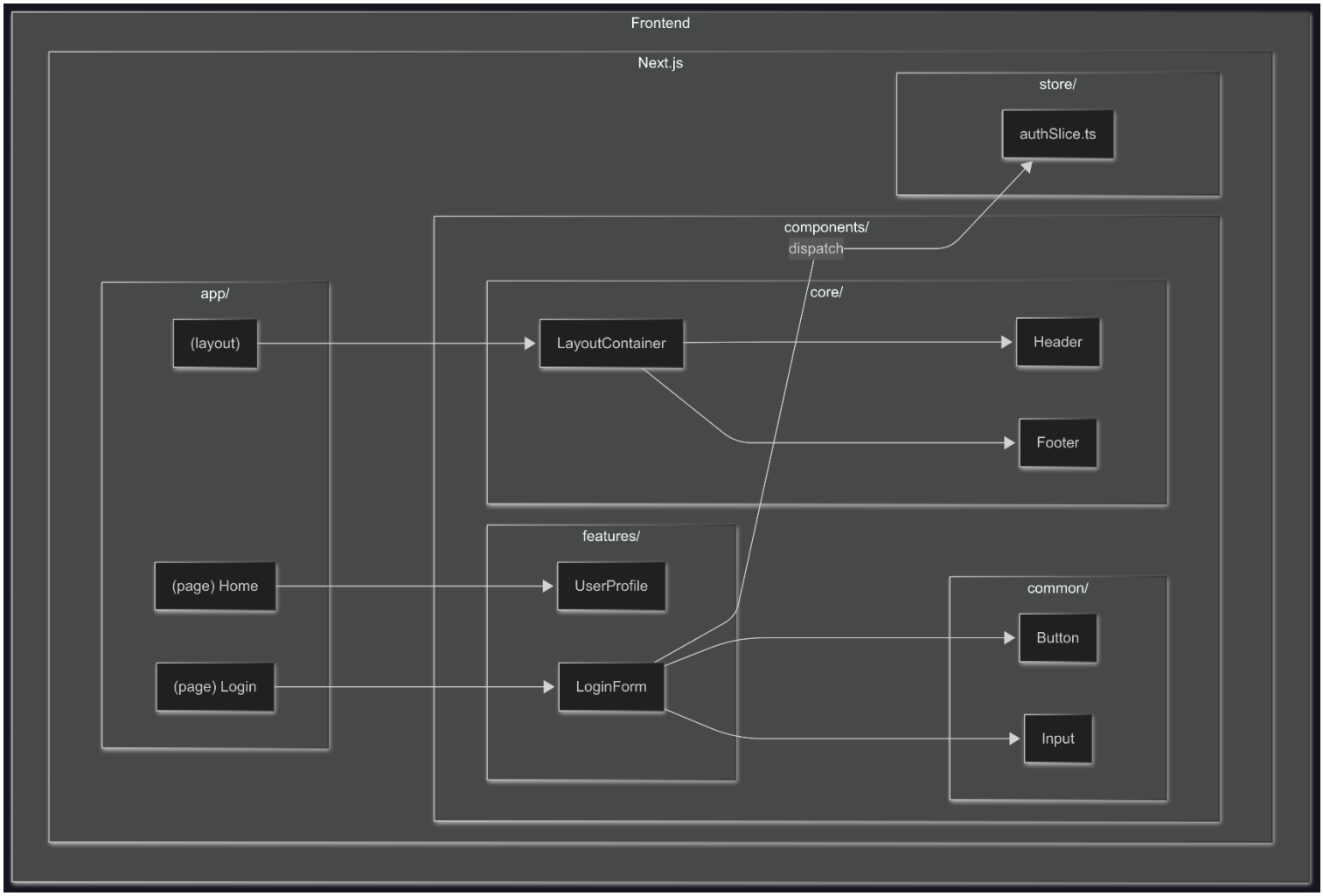
4.1. Arquitectura de Componentes del Backend

En esta fase, expandiremos el backend más allá de la simple app api. Crearemos apps dedicadas para gestionar usuarios y moléculas, estableciendo las bases para futuras funcionalidades.



4.2. Arquitectura de Componentes del Frontend

El frontend se estructurará para soportar la autenticación de usuarios y un layout principal consistente que pueda albergar las futuras herramientas químicas.



4.3. Diagrama de Secuencia de Autenticación de Usuario

Este diagrama muestra el flujo de interacciones para el inicio de sesión de un usuario, una de las funcionalidades clave a desarrollar en la fase 0.1.1.