ChemTools Pro: Documentación de Implementación Alpha-1.0

Introducción

Este documento detalla el proceso de implementación de la versión Alpha-1.0 de ChemTools Pro, una plataforma educativa y práctica para el estudio y análisis químico. Esta versión constituye el "esqueleto" del proyecto, sentando las bases arquitectónicas y técnicas sobre las cuales se desarrollarán las funcionalidades avanzadas en versiones posteriores.

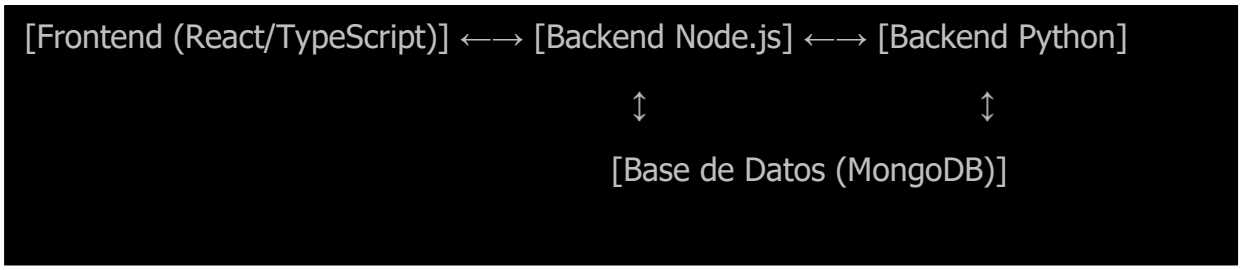
Objetivos de la Versión Alpha-1.0

La versión Alpha-1.0 tiene como objetivo principal establecer la estructura fundamental del proyecto, incluyendo:

1. Configuración del entorno de desarrollo
2. Implementación de la arquitectura base
3. Establecimiento de conexiones entre componentes
4. Creación de la infraestructura de despliegue

Arquitectura General

La arquitectura de ChemTools Pro se basa en un enfoque de microservicios con un backend híbrido:



Componentes Principales

- Frontend: Aplicación React con TypeScript
- Backend Node.js: Servidor Express para APIs generales y gestión de usuarios
- Backend Python: Servidor Flask para cálculos químicos
- Base de Datos: MongoDB para almacenamiento persistente
- Infraestructura: Docker para contenerización

Plan de Implementación Alpha-1.0

Fase 1: Configuración del Entorno de Desarrollo

1.1 Repositorio y Control de Versiones

- Crear repositorio Git en GitHub/GitLab
- Establecer estructura de ramas:
  - main: Versiones estables
  - develop: Integración de características
  - feature/*: Características específicas
- Configurar .gitignore para archivos de entorno y dependencias

1.2 Configuración de Entorno de Desarrollo Local

- Instalar Node.js (v16+), Python (v3.9+), Docker
- Configurar gestor de paquetes y dependencias:
  - npm/yarn para JavaScript
  - pip/venv para Python
- Establecer linters y formateadores de código:
  - ESLint y Prettier para JavaScript/TypeScript
  - Black y Flake8 para Python

Fase 2: Implementación del Frontend

2.1 Estructura Base

```
frontend/
├── public/
├── src/
│   ├── assets/
│   ├── components/
│   │   └── common/
│   ├── config/
│   ├── hooks/
│   ├── pages/
│   ├── services/
│   ├── types/
│   ├── utils/
│   ├── App.tsx
│   └── index.tsx
├── package.json
├── tsconfig.json
└── README.md
```

2.2 Configuración Inicial

- Inicializar proyecto React con TypeScript:
- ```
  npx create-react-app frontend --template typescript
  ```
- Instalar dependencias básicas:
- ```
  npm install react-router-dom axios @mui/material @emotion/react @emotion/styled
  ```
- Configurar TypeScript (tsconfig.json)
- Implementar enrutamiento básico con React Router

2.3 Componentes Básicos

- Crear estructura para componentes comunes:
  - Layout.tsx (estructura general de la aplicación)
  - Navbar.tsx (navegación principal)
  - Footer.tsx (información de pie de página)
  - Loading.tsx (indicador de carga)
  - ErrorBoundary.tsx (manejo de errores)

2.4 Servicios API

- Crear cliente HTTP con Axios:

```
// src/services/api.ts
import axios from 'axios';

const API_URL = process.env.REACT_APP_API_URL || 'http://localhost:4000/api';

const apiClient = axios.create({
  baseURL: API_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});

export default apiClient;
```

Fase 3: Implementación del Backend Node.js

3.1 Estructura Base

```
backend-node/
├── src/
│   ├── config/
│   ├── controllers/
│   ├── middleware/
│   ├── models/
│   ├── routes/
│   ├── services/
│   ├── utils/
│   └── app.js
├── package.json
└── README.md
```

3.2 Configuración Inicial

- Inicializar proyecto Node.js:
- ```
  mkdir backend-node && cd backend-node
  npm init -y
  npm install express mongoose cors helmet dotenv
  npm install -D nodemon typescript ts-node @types/express @types/node
  ```
- Configurar TypeScript (tsconfig.json)
  ```

- Implementar servidor Express básico:
-

```
// src/app.ts
import express from 'express';
import cors from 'cors';
import helmet from 'helmet';
import dotenv from 'dotenv';

dotenv.config();

const app = express();
const PORT = process.env.PORT || 4000;

// Middleware
app.use(cors());
app.use(helmet());
app.use(express.json());

// Routes
app.get('/api/health', (req, res) => {
  res.status(200).json({ status: 'OK', message: 'Node.js API running' });
});

// Error handling middleware
app.use((err: any, req: express.Request, res: express.Response, next
express.NextFunction) => {
  console.error(err.stack);
  res.status(500).json({ error: 'Server error' });
});

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});

export default app;
```

3.3 Conexión a MongoDB

- Configurar conexión a MongoDB:
- ```
  // src/config/database.ts
  import mongoose from 'mongoose';

  const MONGO_URI = process.env.MONGO_URI || 'mongodb://localhost:27017/chemtools';

  const connectDB = async (): Promise<void> => {
    try {
      await mongoose.connect(MONGO_URI);
      console.log('MongoDB connected');
    } catch (error) {
      console.error('MongoDB connection error:', error);
      process.exit(1);
    }
  };
  export default connectDB;
  ```
  ```

3.4 Modelos Básicos

- Implementar modelo básico de usuario:
-
```
// src/models/User.ts
import mongoose, { Schema, Document } from 'mongoose';

export interface IUser extends Document {
  username: string;
  email: string;
  password: string;
  createdAt: Date;
  updatedAt: Date;
}

const UserSchema: Schema = new Schema(
  {
    username: { type: String, required: true, unique: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
  },
```

Fase 4: Implementación del Backend Python

4.1 Estructura Base

```
backend-python/
├── app/
│   ├── __init__.py
│   ├── routes/
│   ├── services/
│   ├── utils/
│   └── config.py
├── requirements.txt
├── Dockerfile
└── README.md
```

4.2 Configuración Inicial

- Configurar entorno virtual de Python:
- ```
  python -m venv venv
  source venv/bin/activate  # En Windows: venv\Scripts\activate
  ```
- Instalar dependencias básicas:
- ```
  pip install flask flask-cors pymongo python-dotenv
  pip freeze > requirements.txt
  ```

- Implementar servidor Flask básico:
-
```
# app/__init__.py
from flask import Flask
from flask_cors import CORS
from dotenv import load_dotenv
import os

load_dotenv()
def create_app():
    app = Flask(__name__)
    CORS(app)
    @app.route('/api/health', methods=['GET'])
    def health_check():
        return {'status': 'OK', 'message': 'Python API running'}
    @app.route('/api/calculate/simple', methods=['POST'])
    def simple_calculation():
        return {'result': 'Calculation placeholder'}
    return app

if __name__ == '__main__':
    app = create_app()
    app.run(host='0.0.0.0', port=int(os.environ.get('PORT', 5000)))  { timestamps: true }
);
export default mongoose.model<IUser>('User', UserSchema);
```

4.3 Conexión a MongoDB

- Configurar conexión a MongoDB:
-
```
# app/config.py
import os
from pymongo import MongoClient

MONGO_URI = os.environ.get('MONGO_URI', 'mongodb://localhost:27017/chemtools')

def get_db_connection():
    client = MongoClient(MONGO_URI)
    db = client.get_database()
    return db
```

Fase 5: Contenedorización con Docker

5.1 Dockerfile para Frontend

```
# frontend/Dockerfile

FROM node:16-alpine as build


WORKDIR /app


COPY package*.json ./

RUN npm install


COPY . .

RUN npm run build


FROM nginx:alpine

COPY --from=build /app/build /usr/share/nginx/html

COPY nginx.conf /etc/nginx/conf.d/default.conf


EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

5.2 Dockerfile para Backend Node.js

```
# backend-node/Dockerfile

FROM node:16-alpine


WORKDIR /app


COPY package*.json ./

RUN npm install


COPY . .


EXPOSE 4000

CMD ["npm", "start"]
```

5.3 Dockerfile para Backend Python

```
# backend-python/Dockerfile

FROM python:3.9-slim



WORKDIR /app



COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt



COPY . .



EXPOSE 5000

CMD ["python", "-m", "app"]
```

5.4 Docker Compose

```
# docker-compose.yml

version: '3.8'


services:

  frontend:

    build: ./frontend

    ports:

      - "80:80"

    depends_on:

      - backend-node

    environment:

      -
REACT_APP_API_URL=http://localhost:4000/api


  backend-node:

    build: ./backend-node

    ports:

      - "4000:4000"

    depends_on:

      - mongodb

      - backend-python

    environment:

      - PORT=4000

      -
MONGO_URI=mongodb://mongodb:27017/chemtools

      - PYTHON_API_URL=http://backend-
python:5000/api
```

```
  backend-python:

    build: ./backend-python

    ports:

      - "5000:5000"

    depends_on:

      - mongodb

    environment:

      - PORT=5000

      -
MONGO_URI=mongodb://mongodb:27017/chemtools


  mongodb:

    image: mongo:latest

    ports:

      - "27017:27017"

    volumes:

      - mongodb_data:/data/db


volumes:

  mongodb_data:
```

1. Pruebas de Componentes Individuales
    - Verificar que cada servicio inicie correctamente
    - Confirmar que los endpoints de salud respondan adecuadamente
    - Validar la conexión a la base de datos
2. Pruebas de Integración
    - Comprobar comunicación entre Frontend y Backend Node.js
    - Verificar comunicación entre Backend Node.js y Backend Python
    - Asegurar que todas las conexiones a MongoDB funcionen correctamente
3. Prueba de Despliegue
    - Construir y ejecutar todos los contenedores Docker
    - Verificar que la aplicación sea accesible en el navegador
    - Confirmar que los logs no muestren errores críticos

Plan para la Versión Alpha-1.1

En la siguiente versión (Alpha-1.1: Articulaciones del Proyecto), se implementarán las siguientes mejoras:

Frontend

- Desarrollo de componentes UI avanzados:
  - Tabla periódica básica (visual sin funcionalidad)
  - Formularios para entrada de datos químicos
  - Layout responsivo para diferentes dispositivos
- Implementación de navegación completa
- Diseño visual preliminar con Material-UI

Backend Node.js

- Desarrollo de endpoints funcionales:
  - Registro y autenticación básica de usuarios
  - Endpoints CRUD para compuestos químicos básicos
  - Integración de validación de datos
- Implementación de middleware de seguridad

Backend Python

- Implementación de endpoints básicos para cálculos:
  - Conversión de unidades químicas
  - Cálculos de masa molar
  - Validación de fórmulas químicas
- Integración de librerías científicas básicas (NumPy, SymPy)

Base de Datos

- Implementación completa de esquemas:
  - Usuarios (perfil, preferencias)
  - Compuestos químicos básicos
  - Historial de cálculos
- Poblado de datos iniciales para pruebas

Conclusión: La versión Alpha-1.0 establece los cimientos técnicos de ChemTools Pro, permitiendo el desarrollo progresivo de funcionalidades en versiones posteriores. Su objetivo principal es proporcionar una base sólida y bien estructurada para la evolución del proyecto.