A continuación se presenta la documentación para la versión **Alpha-1.1 (Articulaciones)** de ChemTools Pro, donde se define el flujo de trabajo, las tareas a realizar y se ofrece una visión general mediante pseudocódigo. Además, se incluye un apartado que resume lo que se abordará en la siguiente versión (Alpha-2.0). La documentación se estructura en secciones para facilitar la comprensión e implementación.

---

1. Introducción

La versión Alpha-1.1 tiene como objetivo articular los cimientos establecidos en Alpha-1.0, agregando funcionalidades básicas de interacción y ampliando la infraestructura para que los módulos del frontend, backend (Node.js y Python) y la base de datos trabajen en conjunto. Esta etapa se centra en implementar componentes UI más avanzados, endpoints funcionales y la creación y poblamiento de esquemas en la base de datos.

---

2. Objetivos de la Versión Alpha-1.1

- **Frontend:**
    - Desarrollar componentes UI avanzados (ej. navbar, footer, formularios).
    - Crear páginas visuales para la tabla periódica básica y formularios de entrada de datos.
    - Implementar navegación completa y diseño responsivo usando Material-UI o Bootstrap.
- **Backend Node.js:**
    - Desarrollar endpoints funcionales para:
        - Registro y autenticación básica de usuarios.
        - Operaciones CRUD para compuestos químicos.
        - Validación de datos mediante middleware de seguridad.
- **Backend Python:**
    - Implementar endpoints para cálculos químicos básicos:
        - Conversión de unidades.
        - Cálculo de masa molar.
        - Validación de fórmulas químicas.
    - Integrar librerías científicas (NumPy, SymPy) para los cálculos.
- **Base de Datos:**
    - Crear e implementar esquemas completos para:
        - Usuarios (perfil, preferencias).
        - Compuestos químicos básicos.
        - Historial de cálculos.
    - Poblado de datos iniciales para pruebas.

---

3. Flujo de Trabajo General

1. **Planificación y Definición de Requerimientos:**
    - Revisión de las funcionalidades descritas en el resumen del proyecto.
    - Definición de tareas específicas para cada módulo (frontend, backend Node.js, backend Python, base de datos).
2. **Diseño de la Arquitectura y Esquemas:**
    - Elaboración de diagramas de flujo y mockups de la interfaz.
    - Diseño de los esquemas para la base de datos.
3. **Desarrollo Paralelo:**
    - **Frontend:** Creación de componentes y páginas de interacción.
    - **Backend Node.js:** Desarrollo de endpoints y middleware de seguridad.
    - **Backend Python:** Implementación de endpoints para cálculos.
    - **Base de Datos:** Configuración de MongoDB y creación de esquemas.
4. **Integración y Pruebas:**
    - Pruebas unitarias e integración entre módulos.
    - Verificación de comunicación entre servicios y validación de datos.
5. **Contenedorización y Despliegue:**
    - Actualización de Dockerfiles y docker-compose para integrar los cambios.
    - Despliegue en entorno de pruebas (local o nube).

---

4. Descripción de Tareas y Pseudocódigo

4.1. Frontend

Tareas:

- Crear y estructurar componentes (Navbar, Footer, Formularios).
- Diseñar la página de la tabla periódica básica (visual, sin funcionalidad completa).
- Implementar la navegación y diseño responsivo.

Pseudocódigo:

```
Función RenderMainLayout():
    Mostrar(Navbar)
    Mostrar(Contenido basado en la Ruta Actual)
    Mostrar(Footer)


Función RenderPaginaPeriodicTable():
    Cargar(EstructuraVisualTablaPeriodica)
    Por cada elemento en listaElementos:
        Mostrar(Elemento de la tabla en formato gráfico)


Función NavegarA(pagina):
    Si pagina existe:
        Redirigir a pagina
    Sino:
        Mostrar("Página no encontrada")
```

4.2. Backend Node.js

Tareas:

- Crear endpoints básicos:
    - /api/health para comprobar el estado del servidor.
    - /api/users para registro y autenticación.
    - /api/compounds para operaciones CRUD en compuestos químicos.
- Implementar middleware de validación y seguridad.

Pseudocódigo:

```
Función Endpoint_Health(request):
    Devolver { status: "OK", message: "Node.js API running" }


Función Endpoint_RegistroUsuario(request):
    datosUsuario = ExtraerDatos(request)
    Si Validar(datosUsuario) es True:
        Crear nuevo usuario en BaseDeDatos
        Devolver { success: True, message: "Usuario registrado" }
    Sino:
        Devolver { success: False, message: "Error en la validación" }


Función Middleware_Validacion(request, next):
    token = ExtraerToken(request)
    Si token es válido:
        Ejecutar next()
    Sino:
        Devolver error "Acceso no autorizado"
```

4.3. Backend Python

Tareas:

- Desarrollar endpoints para:
    - Conversión de unidades químicas.
    - Cálculo de masa molar.
    - Validación de fórmulas químicas.
- Integrar librerías científicas (NumPy, SymPy).

Pseudocódigo:

```
Función Endpoint_CalcularMasaMolar(request):
    fórmula = ExtraerFormula(request)
    Si ValidarFormula(fórmula):
        masaMolar = CalcularMasa(fórmula)  // Usar datos de tabla periódica
        Devolver { result: masaMolar }
    Sino:
        Devolver { error: "Fórmula inválida" }


Función Endpoint_ConversionUnidades(request):
    valor, unidadOrigen, unidadDestino = ExtraerDatos(request)
    valorConvertido = Convertir(valor, unidadOrigen, unidadDestino)
    Devolver { result: valorConvertido }
```

4.4. Base de Datos

Tareas:

- Crear esquemas para:
    - **Usuario:** username, email, password, fecha de creación.
    - **Compuesto:** nombre, fórmula, propiedades.
    - **Historial de Cálculos:** id_usuario, detalle del cálculo, fecha.
- Poblar la base de datos con datos iniciales para pruebas.

Pseudocódigo:

```
Definir Esquema_Usuario:
    username: String
    email: String
    password: String
    fechaCreacion: Date

Definir Esquema_Compuesto:
    nombre: String
    fórmula: String
    propiedades: Objeto

Definir Esquema_HistorialCalculos:
    id_usuario: Referencia a Usuario
    detalleCalculo: Objeto
    fecha: Date

Función InicializarDatos():
    Insertar datos de prueba en Esquema_Compuesto
    Insertar datos de prueba en Esquema_Usuario
```

---

5. Integración y Pruebas

Tareas:

- Realizar pruebas unitarias para cada endpoint.
- Ejecutar pruebas de integración entre:
    - Frontend y Backend Node.js.
    - Backend Node.js y Backend Python.
    - Conexión con MongoDB.

Pseudocódigo para pruebas:

```
Función Test_EndpointHealth():
    respuesta = EnviarPeticion(GET, "/api/health")
    Verificar(respuesta.status == "OK")

Función Test_RegistroUsuario():
    datosPrueba = { username, email, password }
    respuesta = EnviarPeticion(POST, "/api/users", datosPrueba)
    Verificar(UsuarioCreadoEnBD(datosPrueba))
```

---

6. Contenedorización y Despliegue

Tareas:

- Actualizar y probar Dockerfiles para cada servicio.
- Configurar docker-compose para iniciar el entorno completo (frontend, backend-node, backend-python, mongodb).

Pseudocódigo para despliegue:

Definir servicios en docker-compose.yml:

servicio_frontend: construir desde ./frontend, exponer puerto 80

servicio_backend_node: construir desde ./backend-node, exponer puerto 4000

servicio_backend_python: construir desde ./backend-python, exponer puerto 5000

servicio_mongodb: imagen de mongo, exponer puerto 27017

Ejecutar "docker-compose up" para iniciar todos los servicios

Verificar logs y respuestas de endpoints

---

7. Plan para la Siguiente Versión: Alpha-2.0

En la versión **Alpha-2.0 (Intermedio Bajo – Órganos)** se implementarán funcionalidades centrales que ampliarán la interactividad del proyecto:

- **Frontend:**
    - Desarrollo de una página interactiva para la tabla periódica, con datos dinámicos y elementos gráficos.
    - Implementación de la página para la calculadora de disoluciones.
- **Backend:**
    - Lógica de negocio para la tabla periódica (carga de datos estáticos o mediante API externa).
    - Desarrollo de la calculadora de disoluciones en Python, aplicando fórmulas químicas.
- **Base de Datos:**
    - Almacenamiento de datos ampliado de compuestos y usuarios, optimizando la estructura para consultas más complejas.

Pseudocódigo para Alpha-2.0:

```
Función RenderPaginaTablaPeriodica():
    datosElementos = ObtenerDatosElementos()  // Desde BD o API externa
    Por cada elemento en datosElementos:
        Mostrar(Elemento con detalles interactivos)
    Permitir(Filtro y Selección de Elementos)


Función CalcularDisolucion():
    entrada = ExtraerDatosFormulario()
    resultado = AplicarFormulaDisolucion(entrada)
    Devolver(resultado)
```

---

8. Conclusiones

La versión Alpha-1.1 sienta las bases para la interacción y articulación de los módulos del proyecto ChemTools Pro. Se han definido tareas claras para el desarrollo en cada capa (frontend, backend Node.js, backend Python y base de datos) mediante pseudocódigo, facilitando la comprensión del flujo de trabajo y la integración de componentes. Posteriormente, la versión Alpha-2.0 se enfocará en transformar los elementos visuales y funcionales básicos en módulos interactivos y operativos que impulsarán el crecimiento y la complejidad del proyecto.

Esta documentación servirá de guía para el equipo de desarrollo, asegurando que cada componente se integre de manera coherente y que las futuras mejoras se implementen de forma escalable y organizada.
