DIAGRAMA DE SOLUCIÓN

1. Requisitos y Funcionalidades

- **Requisitos Funcionales:**

Registro de usuarios con nombre, correo electrónico, contraseña y detalles del teléfono.

Validaciones de datos de entrada (formato de correo electrónico, longitud de contraseña, etc.).

Almacenamiento de información de usuario en base de datos en memoria en este caso H2.

- **Requisitos No Funcionales:**

Seguridad: Manejo seguro de contraseñas.

Escalabilidad: Capacidad de manejar un crecimiento en el número de usuarios.

Mantenibilidad: Facilidad de mantenimiento y actualización del servicio.

2. Diseño de la Arquitectura

- **Capa de Presentación (Controlador)**

Manejo de solicitudes HTTP.

Validación de datos de entrada.

Invocación de servicios de negocio.

- **Capa de Negocio (Servicio)**

Validación de lógica de negocio.

Interacción con la capa de persistencia para almacenar o recuperar datos.

- **Capa de Persistencia (Base de Datos)**

Almacenamiento de datos de usuario en H2.

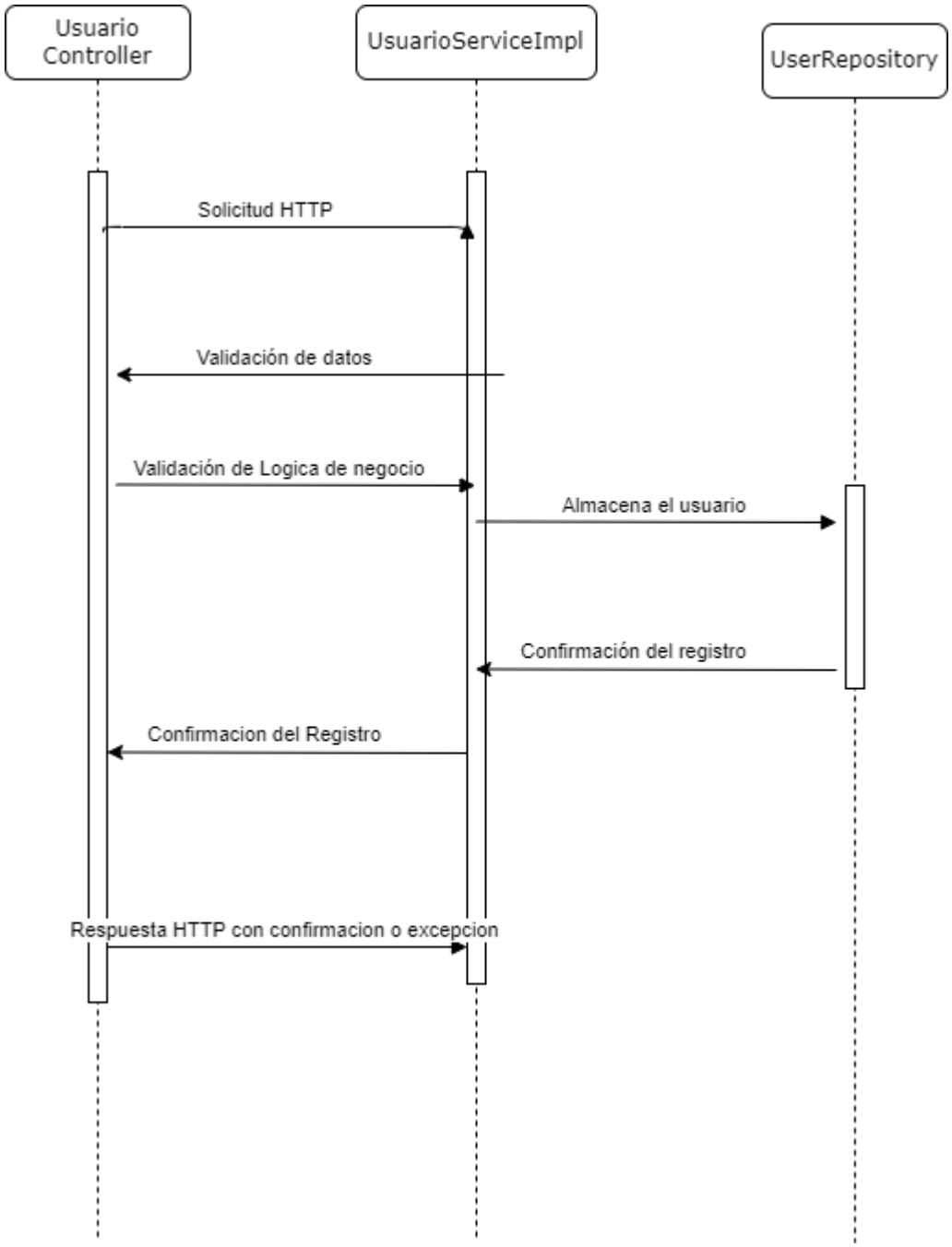
1.1 Objetivo del Servicio

El objetivo de este servicio es recibir y registrar la información de usuarios y los teléfonos asociados a este.

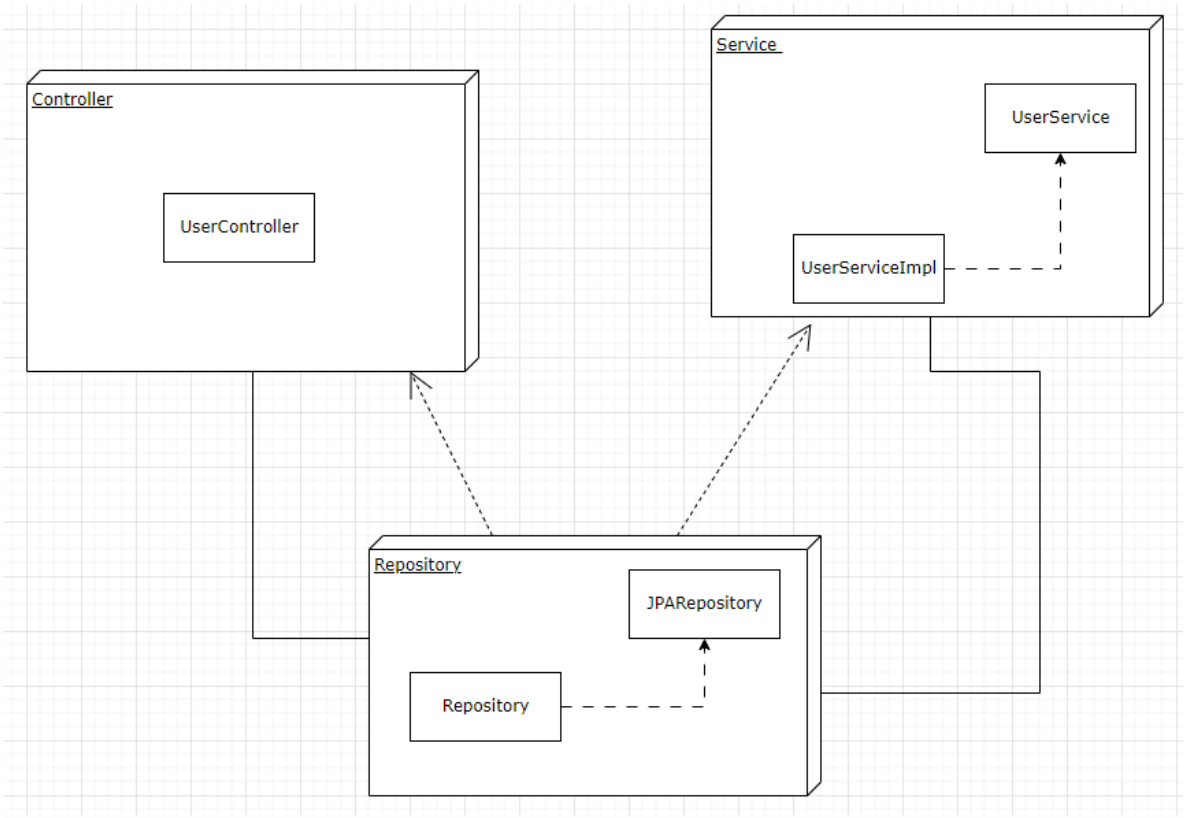
1.2 Alcance del Servicio

Este servicio debe permitir registrar la información de usuarios y los teléfonos.

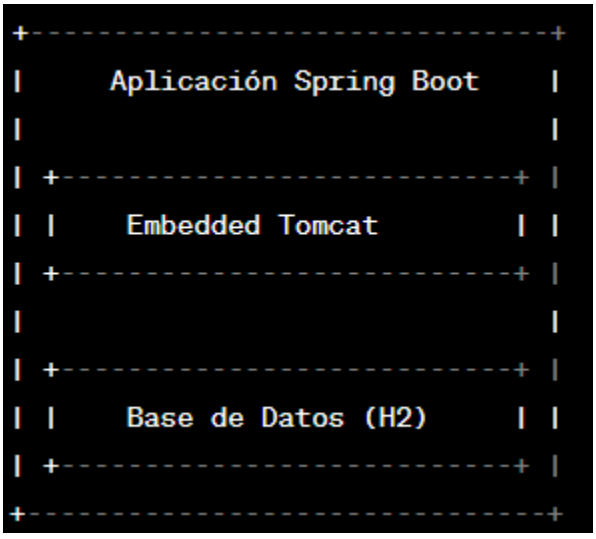
3. Diagrama de Flujo (Secuencia) del Servicio



4. Diagrama de componentes



5. Diagrama de Despliegue



5. Integración

MÉTODO	POST
URL	localhost:8080/usuarios/registro
HEADER	application/json

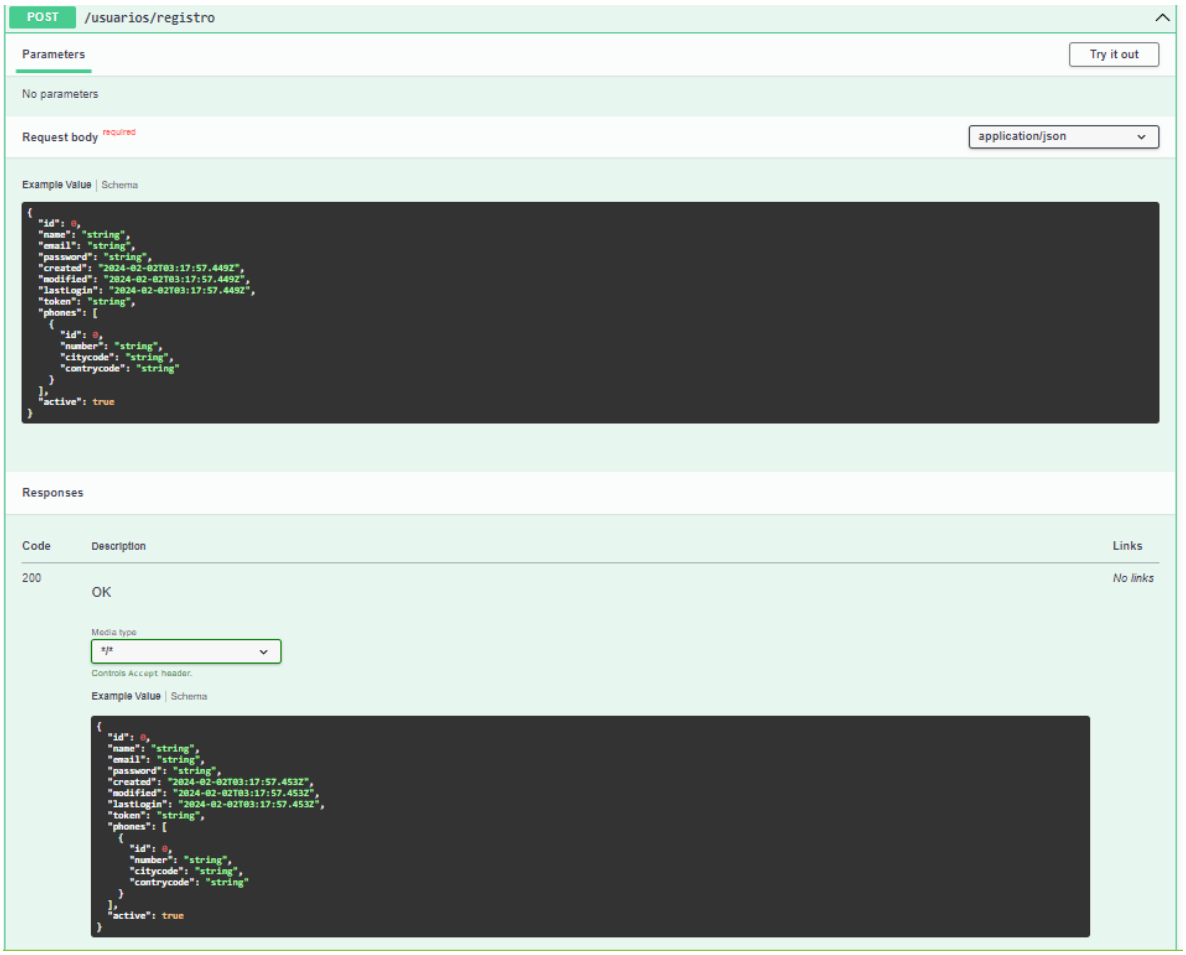
BODY	<pre>{
	"name": "Juan Rodriguez",
	"email": "juas@rodriguez.com",
	"password": "Eder12345",
	"phones": [
	{
	"number": "1234567",
	"citycode": "1",
	"contrycode": "57"
	}
	]
	}</pre>

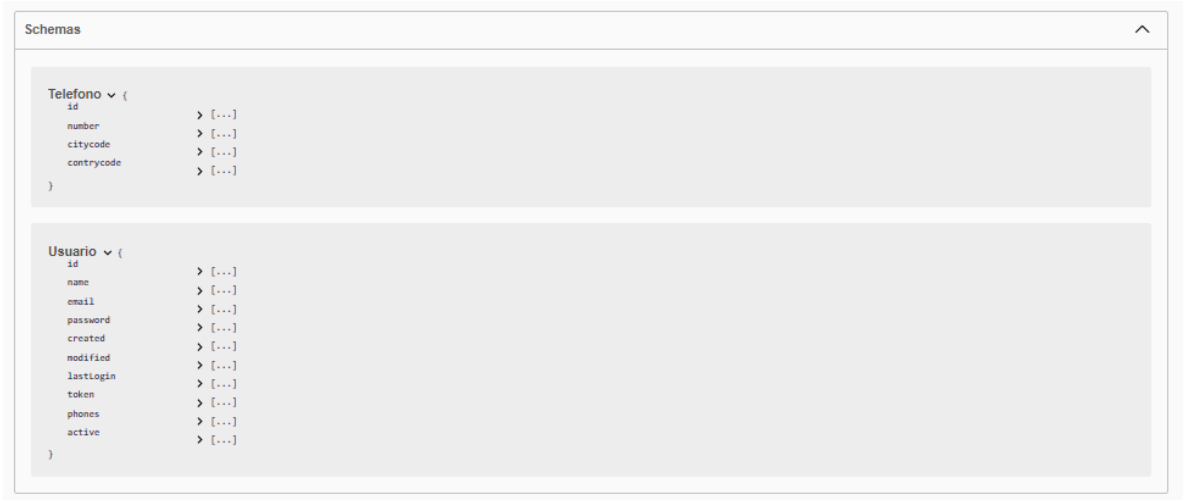
<pre>{
    "id": 2,
    "name": "Juan Rodriguez",
    "email": "juas3@rodriguez.com",
    "password": "Eder12345",
    "created": "2024-02-02T15:34:41.676+00:00",
    "modified": "2024-02-02T15:34:41.676+00:00",
    "lastLogin": "2024-02-02T15:34:41.676+00:00",
    "token": "c71653f3-eebf-4b61-9ac3-bbe3d92e544f",
    "phones": [
        {
            "id": 2,
            "number": "1234567",
            "citycode": "1",
            "contrycode": "57"
        }
    ],
    "active": true
    ]
}</pre>

6. Documentación de API

Se realizo con swagger.

http://localhost:8080/swagger-ui/index.html





7. MANEJO DE MENSAJE DE ERRORES

- Correo electrónico en uso

```
{
    "mensaje": "Este correo electrónico ya está en uso. Por favor, elige otro."
}
```

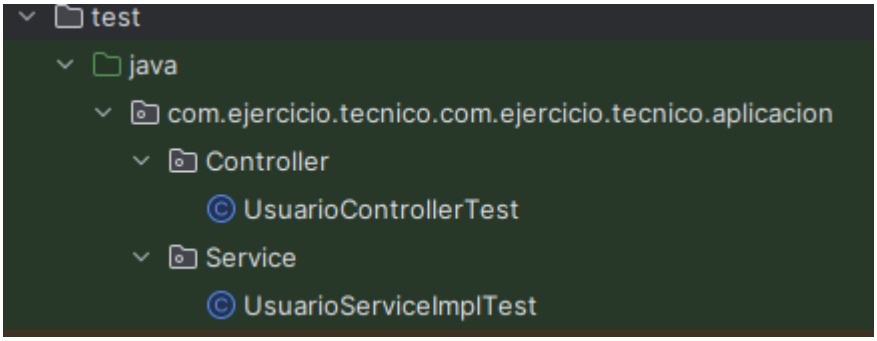
- Correo electrónico invalido.

```
{
    "mensaje": "Por favor, introduce un correo electrónico válido"
}
```

- Clave no cumple con formato

```
{
    "mensaje": "La contraseña debe contener al menos una mayúscula, una minúscula y un
número"
}
```

8. TEST UNITARIOS




```java
@Test
public void testRegistrarUsuario() {
    // Configuración
    MockitoAnnotations.openMocks( testClass: this);

    Usuario usuario = new Usuario();
    usuario.setName("Juan Rodriguez");
    usuario.setEmail("juas4@rodriguez.com");
    usuario.setPassword("Eder12345");

    Usuario usuarioRegistrado = new Usuario();
    usuarioRegistrado.setId(1L);
    usuarioRegistrado.setName("Juan Rodriguez");
    usuarioRegistrado.setEmail("juas4@rodriguez.com");
    usuarioRegistrado.setPassword("Eder12345");
    Mockito.when(usuarioService.registrarUsuario(any(Usuario.class))).thenReturn(usuarioRegistrado);

    // Ejecución
    ResponseEntity<Usuario> responseEntity = usuarioController.registrarUsuario(usuario);

    // Verificación
    assertEquals(HttpStatus.CREATED, responseEntity.getStatusCode());
    Usuario usuarioResponse = responseEntity.getBody();
    assertEquals( expected: 1L, usuarioResponse.getId());
    assertEquals( expected: "Juan Rodriguez", usuarioResponse.getName());
    assertEquals( expected: "juas4@rodriguez.com", usuarioResponse.getEmail());
    assertEquals( expected: "Eder12345",usuarioResponse.getPassword());
}
}
```

MENSAJE EXITOSO DE REALIZAR EL TEST