R.T aplicación de registro de usuario

Requisito funcional:

- Desarrolle una aplicación que exponga una API RESTful de creación de usuarios.
- Todos los endpoints deben aceptar y retornar solamente JSON, inclusive al para los mensajes de error.

Requisito técnico para endpoint de registro:

- Endpoint deberá recibir un usuario con los campos "nombre", "correo", "contraseña",

```
{
    "name": "Juan Rodriguez",
    "email": "juan@rodriguez.org",
    "password": "hunter2",
    "phones": [
        {
            "number": "1234567",
            "citycode": "1",
            "contrycode": "57"
        }
    ]
}
```

más un listado de objetos "teléfono", respetando el siguiente formato:

- Responder el código de status HTTP adecuado
- En caso de éxito, retorna el usuario y los siguientes campos:
    - id: id del usuario (puede ser lo que se genera por el banco de datos, pero sería más deseable un UUID)
    - created: fecha de creación del usuario
    - modified: fecha de la última actualización de usuario
    - last_login: del último ingreso (en caso de nuevo usuario, va a coincidir con la fecha de creación)
    - token: token de acceso de la API (puede ser UUID o JWT)
    - isactive: Indica si el usuario sigue habilitado dentro del sistema.
- Si caso el correo conste en la base de datos, deberá retornar un error "El correo ya registrado".
- El correo debe seguir una expresión regular para validar que formato sea el correcto. (aaaaaaa@dominio.cl)
- La clave debe seguir una expresión regular para validar que formato sea el correcto. (Una Mayuscula, letras minúsculas, y dos números)
- El token deberá ser persistido junto con el usuario

Requisito de stack técnico:

- Banco de datos en memoria, como HSQLDB o H2.
- Proceso de build vía Gradle.
- Persistencia con Hibernate.
- Framework Spring.
- Servidor Tomcat o Jetty Embedded
- Java 8+
- Entrega en un repositorio público (github o bitbucket) con el código fuente y script de creación de BD.
- Entrega diagrama de la solución.
- Pruebas de unidad

Observaciones:

- Para la aplicación se usará una arquitectura Restful con los siguientes detalles:
    - endpoint unificado para los 4 métodos principales (POST, GET, PUT, DELETE).
- La aplicación tendrá endpoints de servicios /user y /login.
- Detalle de los endpoints:
    - /user
        - POST -> registro de usuario
        - GET -> consultar usuario
        - PUT -> actualización de usuario
    - /login
        - POST -> acceso de sesión creación de token
        - GET -> consulta de token de acceso
        - PUT -> actualización / refresh de token
    - Detalles de usuario
        - Registro es solo permitido una vez
        - Contraseña se guarda encriptada y no se puede desencriptar
        - Consulta de datos responde una lista de usuario, de cara a la prueba la búsqueda por email (query param) será requerido
        - Actualización de datos del usuario solo permitirá la modificación de los parámetros token, last_login e isActive. El campo update se actualizará de forma lógica en la aplicación
        - El servicio de eliminación consistirá en un eliminado lógico de la tabla. Se agrega campo status para ello con los estados “active”, “deleted”
    - Detalle de login
        - Creación de token genera un token de sesión nuevo si no hay ninguno activo o expirado. De existir uno vigente mantendrá dicho token y será retornado.
        - En caso de password incorrecta o parámetro isActive igual a falso, retornara un 404, con mensaje “acceso denegado”

    - El usuario tendrá 3 intentos si el parámetro isActive es igual a verdadero. En caso de ser erróneo los 3 intentos, retornará 404, con mensaje “acceso bloqueado”
    - Consulta de login necesita como parámetro requerido email y token vigente, en caso de falta un campo o ambos retornará 400, con mensaje email y token requerido.
    - Consulta de login validará el estado del token, en caso de ser vigente retornará el mismo token. En caso de ser erróneo al email, retornara un 401, con el mensaje token no autorizado.
    - Consulta de login sobre un token expirado, retornará un 404, con mensaje token expirado.
    - La actualización de token permitirá solo refrescar el tiempo de expiración de un token. En caso de un token erróneo o expirado los mensajes de error serán los mismos que el punto anterior.
    - La eliminación de un token, consistirá en destruir la sesión, la tabla de sesión tendrá el parámetro estado que permitirá los valores “active”, “expired”, “destroyed”.

Consideraciones:

- Se toma en cuenta que la postulación al cargo es para un banco, por lo que se infiere la necesidad de una aplicación escalable, mantenible y de alta concurrencia.
- Se solicita usar hibernate y jpa, lo que se mantendrá pero con webflux.
- Se utilizará un enfoque de programación funcional por encima de MVC.
- El token a construir sera un token JWT
- Se estandariza aplicación con 4 capas de desarrollo:
    - settings: package que tendrá clases de configuración en la app.
    - infrastructure: package que tendrá clases de clientes enfocado en base de datos, clientes de servicios u otras librerías.
    - domain: package que tendrá la lógica de negocio de las distintas funcionalidades solicitadas en el requerimiento.
    - api: package que tendrá lógica orientada a restful como validaciones, modelos, handler, mapeo de respuestas de la aplicación.
- Los test unitarios se enfocarán en las clases de la capa domain y api, específicamente lo que tenga que ver con mapeo, handler, validaciones, service respectivamente. Se descartan los test para dtos, model, capa infrastructure y settings.

Manejo de errores:

- Códigos de error aceptados por la aplicación:
    - 400 -> Bad Request
    - 401 -> Not Authorized
    - 404 -> Not Found

- 500 -> Internal Server Error