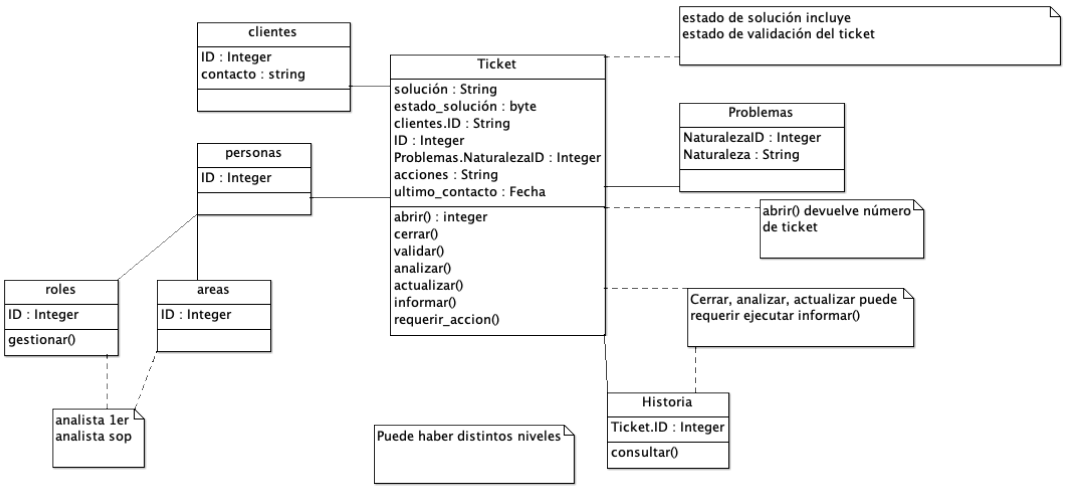
Ingeniería de Software I

Nota de Aplicación

Componentes Serverless -- AWS Full Stack Node.JS

Problema

Dado el planteo de la aplicación para soporte de Mesa de Ayuda se desea hacer un diseño detallado e implementación (parcial) del componente **clientes** en una arquitectura de tipo “Serverless” utilizando los servicios de AWS Full Stack basado en Node.JS.



La guía para la solución a realizar consistirá en:

- Tomar como punto de partida el diagrama de clases básico utilizado en clases y talleres previos para la clase **clientes**.
- Agregar los atributos y métodos que emergieron durante la ejecución de los diferentes prácticos como necesarios para sostener funciones bajo análisis.

    - Login del cliente.
    - Registración del cliente.
    - Información de contacto.
    - Atributos para gestionar el estado de activación y registración.

- Agregar atributos y métodos que permitan gestionar el objeto una vez implementado.

    - Agregar nuevo cliente (add).
    - Obtener los datos de un cliente (get).
    - Modificar los datos de un cliente existente (update).

  - Dar de baja un cliente (delete).
  - Suspender a un cliente (set).
  - Cambiar la password (reset).

- Explicitar las reglas del negocio que serán aplicables durante el ciclo de vida del cliente y que emergen de las reglas implícitas establecidas durante el proceso de análisis, los casos de uso y los requisitos de implementación.
- Actualizar el diagrama de clases con el resultante de las actividades anteriores.
- Hacer una implementación full-stack bajo Node.js en AWS del componente resultante.

  - Implementar un repositorio de persistencia en DynamoDB.
  - Implementar funciones lambda para los métodos.
  - Implementar métodos de acceso para las funciones.
  - Validar y Verificar los componentes.

Actualización del diseño detallado del componente

- Partir del diagrama de clases utilizado durante el proceso de análisis.

    - login(ID:String,password:String):boolean
    - registrar(ID:String,estado:boolean):boolean
    - activar(ID:String,estado:boolean):boolean
    - cambiarPassword(ID:String,password:String):boolean
    - Refinar la información de contacto con nombre y contacto (correo).

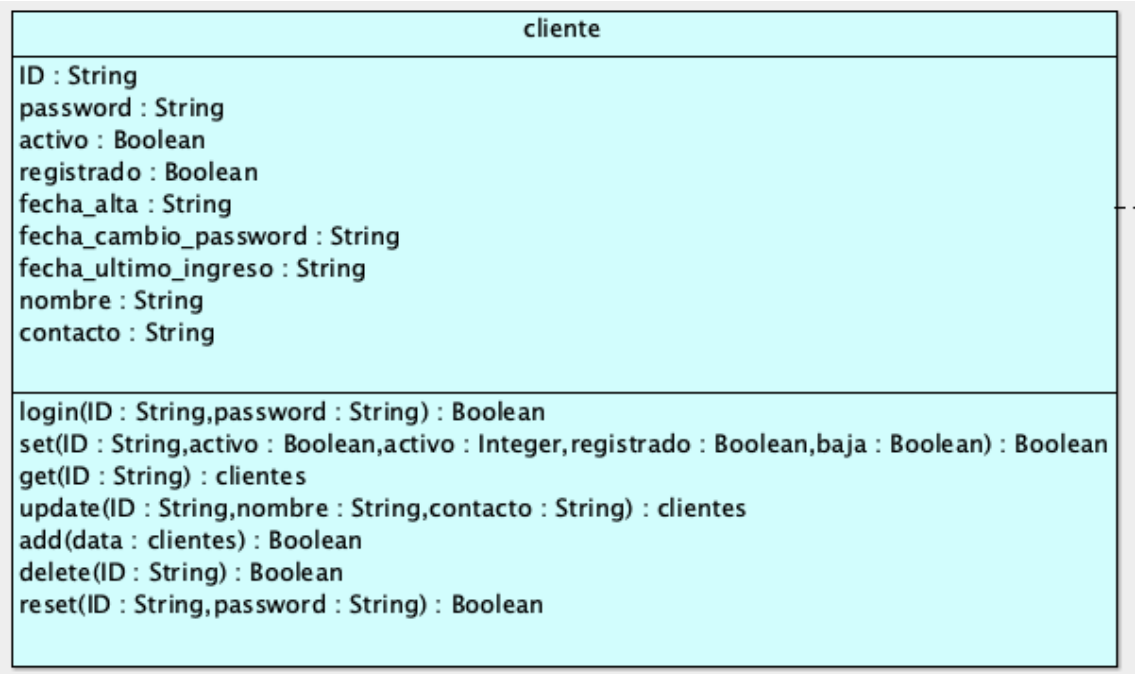
  Siempre es una condición de diseño que los métodos tengan como medida de reducción de acoplamiento retornar el estado de ejecución al finalizar.
  Nótese que tanto la activación como la registración son en realidad máquinas de estados que deben tener un atributo para substanciarse. Cuando hay máquinas de estado es importante que todos los estados de la misma sean alcanzables/modificables, por ello los métodos asociados deben tener el estado como argumento. Las reglas del negocio dictarán que transiciones son válidas y bajo qué condiciones. Actualizar el diagrama de clases.

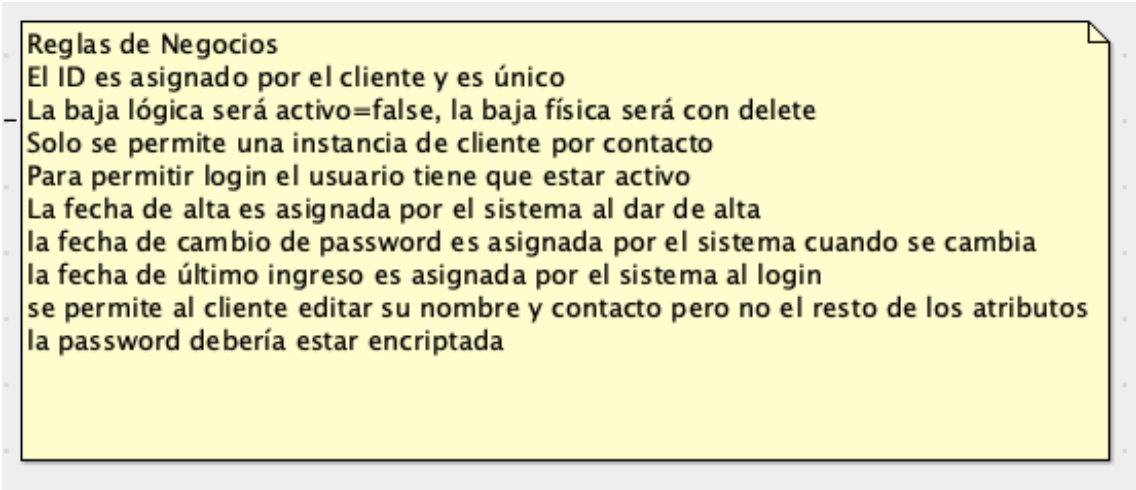
- Son necesarios métodos para gestionar el estado del objeto, y en particular la persistencia de sus atributos.

    - add(data:cliente):String.
    - set(ID:String,data:clientes):boolean
    - get(ID:String):cliente
    - delete(ID:String):boolean

  Las tres operaciones básicas en un objeto son siempre leer (get), grabar (set) y borrar (delete). El add es un caso especial de set donde aún no existe el ID.
  En los casos que el objeto mismo clientes es argumento a retorno se trata de todos los atributos privados de la clase.
  Sin embargo, por consideraciones de seguridad hay que delimitar el alcance de la función set para que exista una que permita actualizar los atributos que el cliente puede modificar (nombre y contacto) y otra para los atributos que la mesa de ayuda pueda modificar (activo, registrado). Al mismo tiempo la actualización de password debe hacer mediante una función separada.
  Por cuestiones de auditoría las fechas no se pueden editar mediante funciones.
  Por su parte get y set dan el mecanismo “helper” para modificar todos los atributos privados de la clase. Actualizar el diagrama de clases.

- Enumerar las reglas de negocio aplicables para asegurar la integridad del componente bajo las distintas operaciones y que deberán ser incorporadas al momento de la implementación del componente.
- Se actualiza el diagrama de clases y se normalizan las funciones al formato de implementación física.





Implementación del componente Full-Stack

Ingresar a AWS

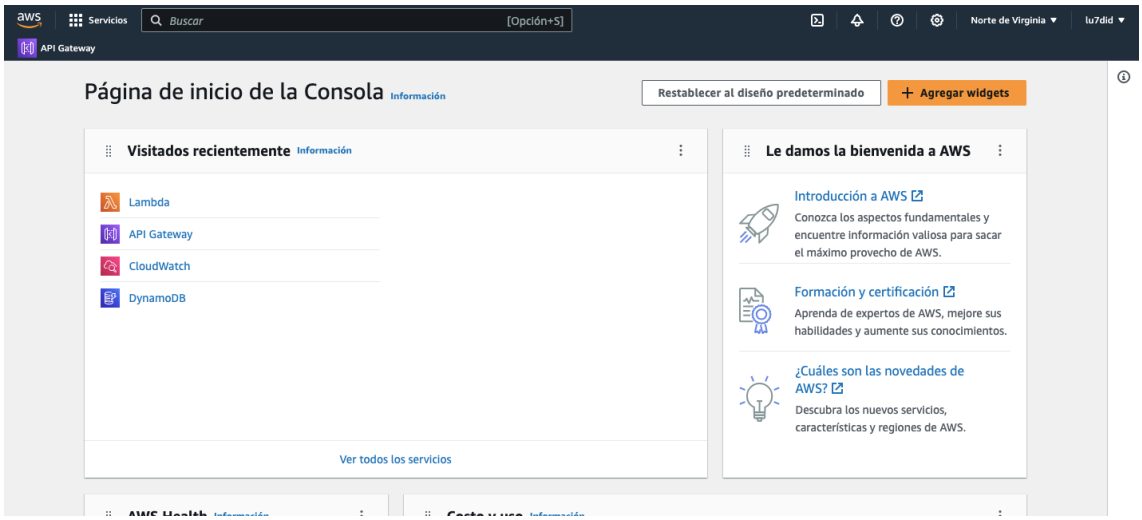
Acceder a la consola de desarrollo del ambiente *Amazon Web Services (AWS)* utilizando un id registrado en AWS. Registrar una cuenta en caso de no tenerla.

El url de acceso es https://aws.amazon.com/es/console/

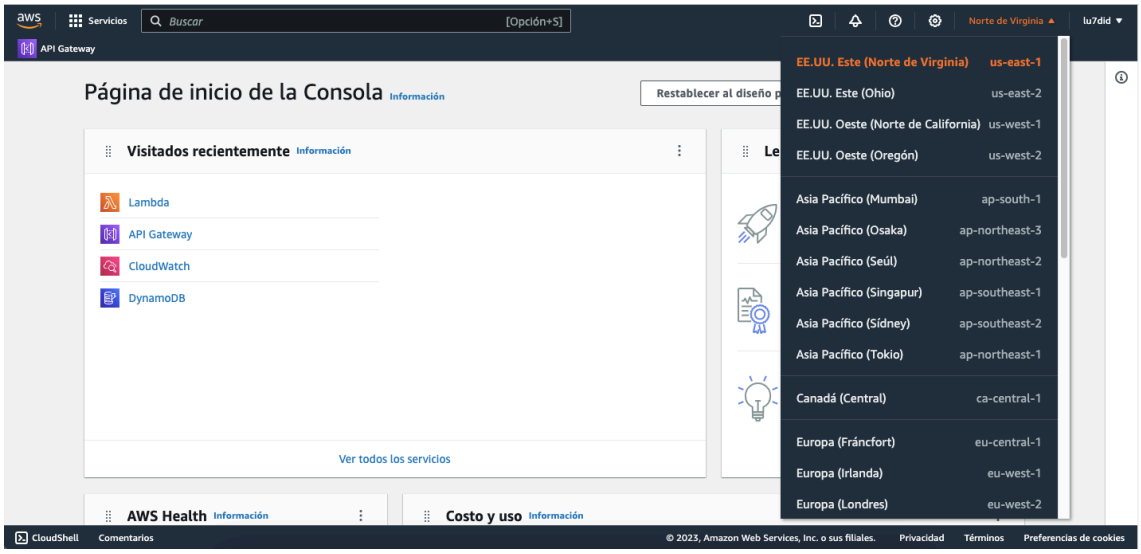
Dar "*Inicie sesión en la consola*", indicar que se opera como "*Usuario raíz*", indicar el usuario registrado y dar "Siguiente".

Indicar la password

Tomar nota de la región en la que se asignó la cuenta AWS (arriba a la derecha), pues será utilizada posteriormente en la definición de algunos servicios.



En éste caso la región es *us-east-1*.



AWS guarda memoria de las herramientas visitadas recientemente , pero si no hubo actividad anterior la página puede tener información diferente en la página de inicio.

Crear tabla cliente

Crear la tabla clientes en *DynamoDB*, generar una clave cualquiera y poblar los atributos necesarios con valores default, exportar a continuación el JSON de elementos de tabla

```
{
"id"                    :       "3f2edee0-f58f-44fc-9945-38903dd5cce",
"activo"                :       false,
"registrado"            :       false,
"contacto"              :       "pedro_colla@hotmail.com",
"fecha_alta"            :       "29/09/2023",
"fecha_cambio_password" :       "29/09/2023",
"fecha_ultimo_ingreso"  :       "29/09/2023",
"nombre"                :       "Pedro E. Colla",
"password"              :       "secret"
}
```

Implementar cada uno de los componentes necesarios:

Definir cada método mediante una función lambda.

- Las funciones lambda a implementar son:
  - addCliente.
  - getCliente.
  - setCliente.
  - resetCliente.
  - updateCliente.
  - listCliente ([1])
  - loginCliente.
- Asegurar las reglas de negocio en el alcance de cada componente.
- Asegurar el procesamiento y validación de los argumentos provistos.
- Proveer los mecanismos de recupero o actualización en base de datos.
- Retornar los argumentos esperados para la función.
- Implementar el mecanismo “API Gateway” para acceder a cada función.
- Asegurar que la función tiene los accesos necesarios a la tabla clientes.

[1] La función *listCliente* no figura en el diseño conceptual pero es necesaria como función de implementación para poder averiguar programáticamente el contenido de la base de datos.

- Los tipos de acceso posible según la función podrán ser:
    - PutItem.
    - GetItem.
    - UpdateItem.
    - Scan.

Ejemplos de implementación de las distintas funciones puede encontrarse en el archivo *MesaAyuda-AWS-FullStack.ZIP* en el aula virtual.

Prueba Unitaria

Probar cada método implementado mediante la herramienta Insomnia a nivel unitario verificando que procesa correctamente los parámetros enviados y produce los resultados indicados por el diseño.

Como guía de despliegue siempre conviene comenzar por los métodos asociados con la gestión del ciclo de vida de cada objeto de forma que permita generar los elementos necesarios para poblar la base de dato, siguiendo luego por los métodos asociados a la funcionalidad requerida y finalizando con los métodos asociados al mantenimiento del objeto y satisfacción de reglas de negocio no-funcionales.