BUAP

Benemérita Universidad

Autónoma de Puebla

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

CURSO: SERVICIOS WEB

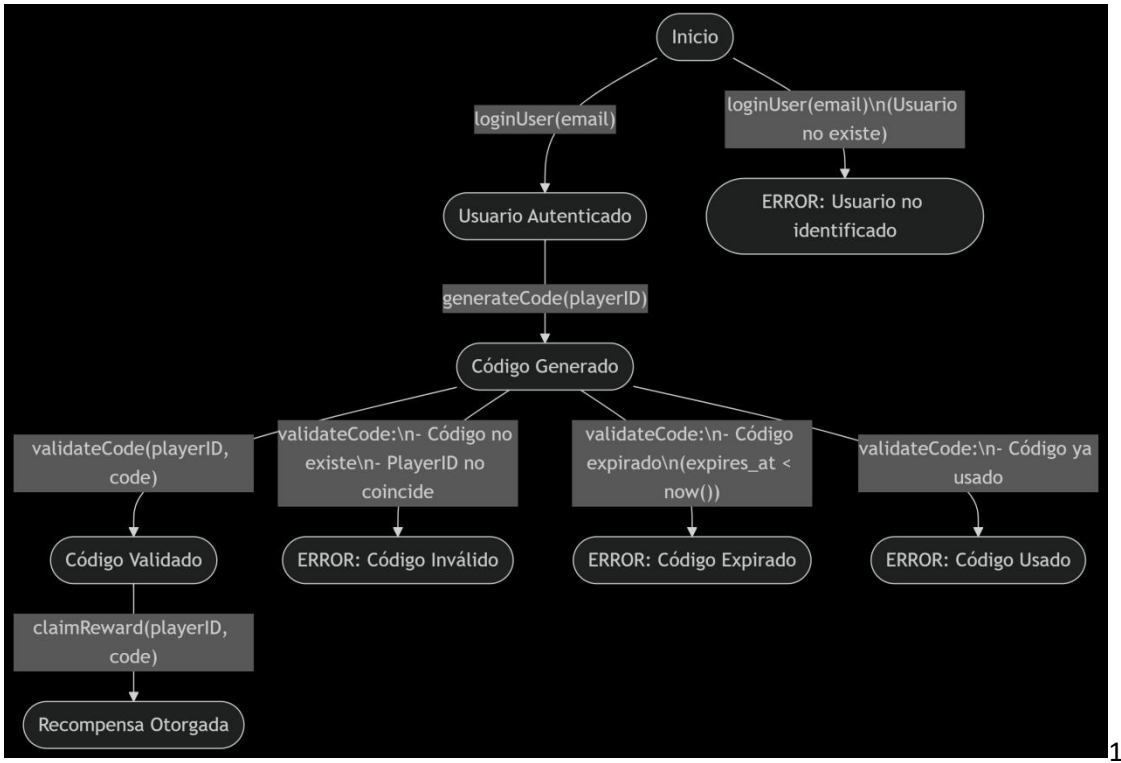
DOCENTE: GUSTAVO EMILIO MENDOZA OLGUIN

ALUMNOS:

JAVIER AGUILAR MACIAS 202142536

JOSÉ ANTONIO COYOTZI JUAREZ 202118536

CONTENIDO



PROPUESTA DEL SERVICIO

El servicio de recompensas fue diseñado para incentivar la participación diaria de los jugadores en una plataforma, ofreciendo códigos únicos que pueden canjearse por recompensas progresivas. La propuesta se centró en tres pilares: generación de códigos no serializables (evitando secuencias predecibles para aumentar la seguridad), progresión diaria (con recompensas acumulativas durante 28 días y un premio final) y validación robusta (garantizando que los códigos solo sean usados por sus dueños y dentro del plazo establecido). El sistema se implementó

utilizando tecnologías como Spyne para el servicio SOAP, Flask para la interfaz web y PostgreSQL para la gestión de datos.

ESTRUCTURA DEL SERVICIO

El servicio se divide en dos componentes principales. El primero es un servicio SOAP (implementado en `reward_services.py`), que utiliza el framework Spyne para exponer operaciones como `loginUser`, `generateCode`, `validateCode` y `claimReward`. Este servicio se inicia con un servidor WSGI en el puerto 8000 y maneja las solicitudes de forma concurrente mediante hilos. El segundo componente es una interfaz web (desarrollada en `app.py` con Flask), que consume el servicio SOAP y permite a los usuarios interactuar mediante una página HTML. La interfaz inicia en el puerto 5000 y se comunica con el backend mediante llamadas SOAP, mostrando mensajes de éxito o error según las respuestas recibidas.

```
# Configuración inicial del servicio SOAPclass RewardService(ServiceBase):
    @rpc(Unicode, _returns=Unicode)
    def loginUser(ctx, email):
        # ... (código de autenticación)

    @rpc(Unicode, _returns=Unicode)
    def generateCode(ctx, playerID):
        # ... (código de generación)

    @rpc(Unicode, Unicode, _returns=Unicode)
    def validateCode(ctx, playerID, code):
        # ... (código de validación)

    @rpc(Unicode, Unicode, _returns=Unicode)
    def claimReward(ctx, playerID, code):
        # ... (código de canje)
# Inicialización del servidor
application = Application([RewardService], tns='rewards', in_protocol=Soap11
(), out_protocol=Soap11())
```

Cliente Flask

```
# Configuración básica de Flask
app = Flask(__name__)
app.secret_key = "supersecret"
client = Client('http://localhost:8000/?wsdl')
@app.route("/", methods=["GET", "POST"])def index():
```

WSDL Y OPERACIONES

El WSDL (Web Services Description Language) se genera automáticamente en `http://localhost:8000/?wsdl` y sirve como contrato técnico para integrar el servicio. Define las cuatro operaciones clave:

- `loginUser(email)`: Autentica al usuario mediante su correo y devuelve su `playerID` y `clientID`.

- `generateCode(playerID)`: Genera un código único con formato `XXX-YYYYYYYYYY`, registrándolo en la base de datos con una expiración a las 5 AM del día siguiente.

- `validateCode(playerID, code)`: Verifica que el código exista, no haya expirado, no esté usado y pertenezca al jugador.

- `claimReward(playerID, code)`: Canjea el código, marca su uso y asigna la recompensa correspondiente.

Todas las operaciones usan el protocolo SOAP 1.1 y devuelven respuestas en formato `cadena|separada|por|pipes` para errores o datos.

```
@rpc(Unicode, _returns=Unicode)def loginUser(ctx, email):
    with conn.cursor() as cur:
        cur.execute("SELECT player_id, client_id FROM users WHERE email=%s",
 (email,))
        user = cur.fetchone()
```

```
        return f"{user[0]}|{user[1]}" if user else "ERROR|Usuario no encontr
ado"
```

BASE DE DATOS Y REGISTROS

La base de datos PostgreSQL se estructura en tres tablas:

1. `users`: Almacena correos electrónicos, `player_id` (clave primaria) y `client_id`.

2. `player_progress`: Registra el día actual del jugador (`current_day`), la fecha de su primer reclamo (`first_request_date`) y la última solicitud (`last_request_date`). Si hay una interrupción de más de un día, el progreso se reinicia.

3. `tokens`: Guarda los códigos generados (`code`), su fecha de expiración (`expires_at`), estado (`used`) y día asociado.

Se implementaron **claves foráneas** para vincular `player_progress` y `tokens` con `users`, asegurando integridad referencial. Además, se agregaron índices en `email` y `code` para optimizar consultas.

```
CREATE TABLE users (
    player_id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    client_id VARCHAR(50) NOT NULL);
CREATE TABLE player_progress (
    player_id INT REFERENCES users(player_id),
    current_day INT NOT NULL,
    first_request_date DATE,
    last_request_date DATE);
CREATE TABLE tokens (
    code VARCHAR(20) PRIMARY KEY,
    player_id INT REFERENCES users(player_id),
    expires_at TIMESTAMP NOT NULL,
    used BOOLEAN DEFAULT FALSE,
    day INT NOT NULL,
    date_generated DATE NOT NULL);
```

LECTURA DEL CORREO Y AUTENTICACIÓN

El proceso comienza cuando el usuario ingresa su correo en la interfaz web. Flask envía este dato al servicio SOAP mediante `loginUser(email)`. El servidor busca el correo en la tabla `users` y, si no existe, retorna `ERROR|Usuario no encontrado`. Si el registro es exitoso, devuelve `playerID|clientID` y almacena el `playerID` en la sesión de Flask. Este paso es crítico para garantizar que solo usuarios registrados puedan generar y canjear códigos.

```
@rpc(Unicode, _returns=Unicode)def loginUser(ctx, email):
    with conn.cursor() as cur:
        cur.execute("SELECT player_id, client_id FROM users WHERE email=%s",
 (email,))
        user = cur.fetchone()
        return f"{user[0]}|{user[1]}" if user else "ERROR|Usuario no encontr
ado"
```

GENERACIÓN Y VALIDACIÓN DE CÓDIGOS

La generación de códigos (`generateCode`) verifica primero si el jugador ya generó un código ese día, consultando la tabla `tokens`. Si no hay registros, se crea un código no serializable usando una combinación de tres letras aleatorias y diez caracteres alfanuméricos. Para evitar secuencias predecibles, se convierte la parte alfanumérica a un entero en base 36 y se compara con el último código generado, asegurando que no sean consecutivos. El

código se guarda en `tokens` con una expiración a las 5 AM del día siguiente.

En la validación (`validateCode`), el servidor verifica tres condiciones en la base de datos:

1. Existencia: El código debe estar registrado en `tokens`.

2. Vigencia: La fecha actual no debe superar `expires_at`.

3. Uso: El campo `used` debe ser `FALSE`.

Si falla alguna validación, se retorna un error específico (ej: `ERROR|Código expirado`). Si todo es correcto, el código se marca como válido para su canje.

Generación de códigos no serializables

```
def generar_codigo_no_seriable(ultimo_codigo):
    letras = ''.join(random.choices(string.ascii_uppercase, k=3))
    # ... (lógica para evitar secuencias)
    return f"{letras}-{alfanum}"
@rpc(Unicode, _returns=Unicode)def generateCode(ctx, playerID):
    # ... (verificación de código duplicado)
    code = generar_codigo_no_seriable(ultimo_codigo)
    expires = datetime.now() + timedelta(days=1)).replace(hour=5, minute=0)
    cur.execute("INSERT INTO tokens (...) VALUES (%s, %s, %s, FALSE, %s, %s)
", (code, playerID, expires, day, today))
    return f"{code}|{day}|{expires.strftime('%Y-%m-%d %H:%M:%S')}"
```

Validación

```
    @rpc(Unicode, Unicode, _returns=Unicode)
    def validateCode(ctx, playerID, code):
        with conn.cursor() as cur:
            cur.execute("SELECT player_id, expires_at, used FROM tokens WHER
E code=%s", (code,))
            token = cur.fetchone()
            if not token:
                return "ERROR|Código inexistente"
            # ... (validaciones de expiración y uso)
            return "VALIDO"
```

CANJE DE RECOMPENSA

Al reclamar una recompensa (`claimReward`), el servidor realiza una última validación del código para prevenir usos fraudulentos. Si es válido, actualiza el campo `used` a `TRUE` en `tokens` y consulta el diccionario `REWARDS` para obtener la recompensa correspondiente al día actual del jugador. Por ejemplo, en el día 5, se devuelve *"Día 5: 100 monedas de oro"*. Si el día supera 28, se retorna un error. Este paso finaliza el ciclo, entregando al usuario su premio o notificando un problema.

```
@rpc(Unicode, Unicode, _returns=Unicode)def claimReward(ctx, playerID, code):
    with conn.cursor() as cur:
        cur.execute("SELECT day FROM tokens WHERE code=%s", (code,))
        day = cur.fetchone()[0]
        cur.execute("UPDATE tokens SET used=TRUE WHERE code=%s", (code,))
        return f"Día {day}: {REWARDS.get(day, 'Recompensa por definir')}"
```

PETICIONES / RESPUESTAS SOAP EN REWARD_SERVICES.PY

El servidor SOAP maneja las solicitudes y genera respuestas en f ormato XML. Aquí están los ejemplos clave:
Operación loginUser

Petición SOAP:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:rew="rewards">
   <soapenv:Body>
      <rew:loginUser>
         <rew:email>usuario@ejemplo.com</rew:email>
      </rew:loginUser>
   </soapenv:Body></soapenv:Envelope>
```

Respuesta exitosa:

```
<soapenv:Envelope>
   <soapenv:Body>
```

```
      <ns1:loginUserResponse xmlns:ns1="rewards">
         <ns1:loginUserResult>123|client_456</ns1:loginUserResult>
      </ns1:loginUserResponse>
   </soapenv:Body></soapenv:Envelope>
```

Respuesta de error:

```
<ns1:loginUserResponse xmlns:ns1="rewards">
   <ns1:loginUserResult>ERROR|Usuario no encontrado</ns1:loginUserResult></n
s1:loginUserResponse>
```

PETICIONES / RESPUESTAS SOAP EN APP.PY

Operación generateCode

Petición SOAP:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:rew="rewards">
   <soapenv:Body>
      <rew:generateCode>
         <rew:playerID>123</rew:playerID>
      </rew:generateCode>
   </soapenv:Body></soapenv:Envelope>
```

Respuesta exitosa:

```
<ns1:generateCodeResponse xmlns:ns1="rewards">
   <ns1:generateCodeResult>ABC-DEF123456|1|2023-10-05 05:00:00</ns1:generate
CodeResult></ns1:generateCodeResponse>
```

Operación validateCode

Petición SOAP:

```
<soapenv:Envelope   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:rew="rewards">

   <soapenv:Body>

      <rew:validateCode>
```

```
         <rew:playerID>123</rew:playerID>
         <rew:code>ABC-DEF123456</rew:code>
      </rew:validateCode>
   </soapenv:Body></soapenv:Envelope>
```

Respuesta exitosa:

```
<ns1:validateCodeResponse xmlns:ns1="rewards">

<ns1:validateCodeResult>VALIDO</ns1:validateCodeResult></ns1:validateCodeResponse>
```

Respuesta de error (código expirado):

```
<ns1:validateCodeResponse xmlns:ns1="rewards">
   <ns1:validateCodeResult>ERROR|Código expirado</ns1:validateCodeResult></ns1:validateCodeResponse>
```

Operación claimReward

Petición SOAP:

```
<soapenv:Envelope   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:rew="rewards">
   <soapenv:Body>
      <rew:claimReward>
         <rew:playerID>123</rew:playerID>
         <rew:code>ABC-DEF123456</rew:code>
      </rew:claimReward>
   </soapenv:Body></soapenv:Envelope>
```

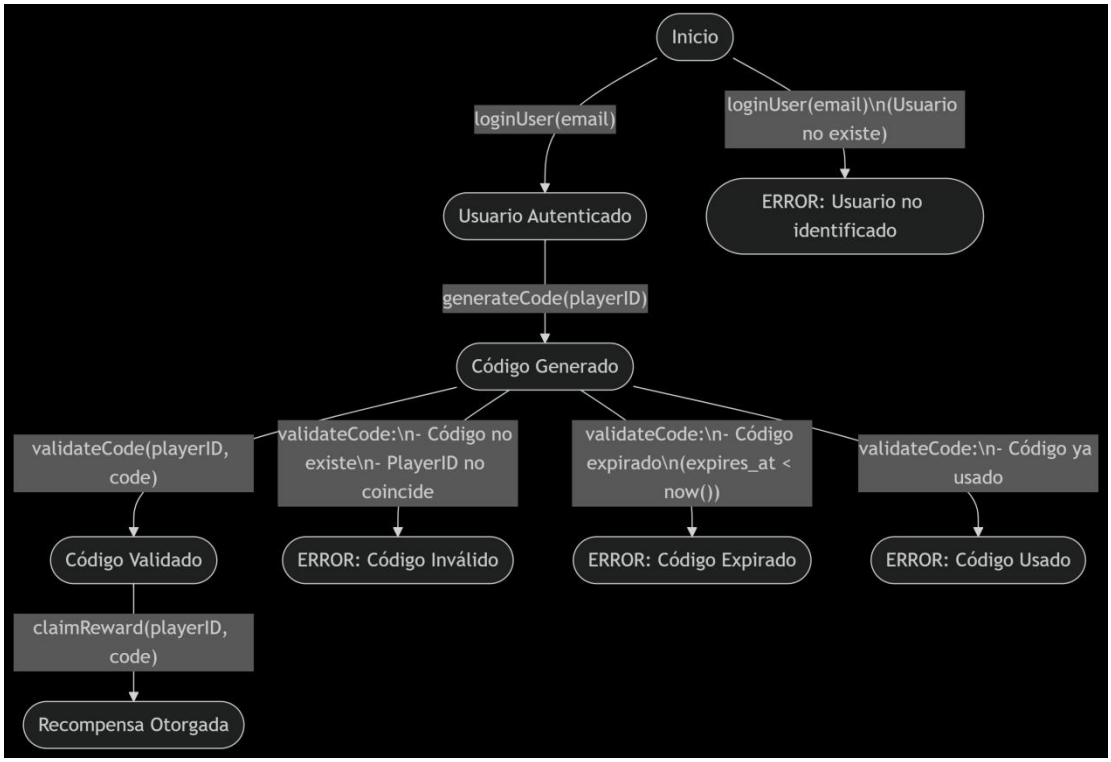
Respuesta exitosa:

```
<ns1:claimRewardResponse xmlns:ns1="rewards">

   <ns1:claimRewardResult>Día 1: Recompensa del día
1</ns1:claimRewardResult></ns1:claimRewardResponse>
```

Respuesta de error:

```
<ns1:claimRewardResponse xmlns:ns1="rewards">

   <ns1:claimRewardResult>ERROR|Código no
válido</ns1:claimRewardResult></ns1:claimRewardResponse>
```

DIAGRAMA DE PETRI



CONCLUSIÓN

El sistema de recompensas integra tecnologías backend (SOAP, PostgreSQL) y frontend (Flask) para ofrecer una experiencia segura y escalable. La estructura modular permite extender funcionalidades, como añadir nuevos tipos de recompensas o métodos de autenticación. Lecciones clave incluyen la importancia de validar entradas en cada paso, gestionar conexiones a la base de datos de forma thread-safe y documentar APIs mediante WSDL para facilitar integraciones futuras. Esta práctica demostró cómo combinar estándares de interoperabilidad con técnicas de seguridad para construir servicios confiables y eficientes.