Componente genérico de administración de usuarios

R. Stiven, J. John, P. Francisco, Universidad EAFIT

Abstract – One of the main challenges of the Software industry, is how to create value in a short amount of time, with the less amount of money. The component-based software engineering is one way to achieve this, with the reuse and the easy development of software thru models. In this paper it´s explain an approach to develop a generic component for user's management, develop in Python due the importance of this language in the present.

I. INTRODUCCIÓN

El desarrollo de software basado en componentes se ha vuelto una práctica cada vez más común en Industria del Software, debido a la búsqueda de procesos más eficientes, para así reducir tiempos de entregas y ahorrar dinero. Sin embargo, esta práctica también ha aumentado su popularidad gracias a la concepción del Software como servicio (SaaS) donde a través de una interfaz podemos acceder a funciones genéricas de fácil conexión. Este trabajo tiene como fin seguir explorando el desarrollo de software basado en componentes, con la creación de un componente genérico, que permita cumplir las funciones básicas para la gestión de usuarios de un sistema, partiendo de un análisis de requerimientos y las tecnologías más relevantes.

II. CONTEXTO

Para este trabajo se analizó en primera instancia el estado del arte acerca de componentes reutilizables que se han comportado de manera similar en cuento a la gestión de usuarios de lado del servidor (back-end), donde se tienen en cuenta elementos claves como el registro, la autenticación y la autorización de usuarios para realizar acciones dentro de un dominio en específico.

Es importante llevar a cabo un proceso de investigación a la hora de modelar y desarrollar un componente como éste, el cual se trata de un dominio que es comúnmente manejado en la mayoría de aplicaciones software del mercado. La administración de usuarios es realmente importante, por lo que una buena implementación daría lugar a un posible reúso de dicho componente para desarrollar software sin preocuparse por especificaciones que ya han seguido resueltas a cabalidad.

III. REQUERIMIENTOS

Para el desarrollo del presente trabajo se recibió una lista de requerimiento donde se describían las funcionalidades y los atributos de calidad que debía tener el componente. Además, se especifica la prioridad y el riesgo existente para la implementación de cada requisito.

ID	Requerimiento	Prioridad	Riesgo
FR1.01.01	El Sistema debe soportar el concepto de usuario.	1	C
FR1.02.01	El Sistema debe soportar el concepto de roles de usuario.	1	C
FR1.03.01	El Sistema debe habilitar funciones de acuerdo al rol del usuario.	1	C
FR1.04.01	Los Administradores del Sistema deben poder otorgar o quitar permisos a usuarios.	2	M
FR1.05.01	El Sistema debe almacenar la lista de todos los usuarios.	1	C
FR1.06.01	El Sistema debe atorgar a los usuarios con permisos de administrador, permisos para administrar roles y sus propiedades	1	C
FR1.07.01	El sistema debe permitir que los usuarios cambien su contraseña.	1	C
FR1.08.01	El sistema debe pedir una autenticación, antes de permitir trabajar en el sistema.	1	C
FR1.09.01	Tanto los roles administrativos como de usuario, pueden cambiar su contraseña luego de autenticarse, o en caso de que se les olvide.	1	C

Tabla 1. Especificación de Requerimientos.

IV. ESPECIFICACIÓN DEL COMPONENTE

1. Nombre del componente

El nombre del componente es "User management component".

2. Alternativas de nombre del componente

El componente puede ser llamado en español como componente para "gestión de usuarios" o "administración de usuarios".

3. Propiedades

3.1 Tipo

El componente se caracteriza por ser un componente de sistema, ya que podría ser integrado en el futuro con otras aplicaciones software que requieran una correcta administración de usuarios.

3.2 Subtipo

El componente se caracteriza por ser de implementación, ya que se llevó a cabo un proceso de desarrollo de software tomando en cuenta otros componentes reutilizables que fueron tenidos a consideración para el desarrollo de este.

3.2 Nivel

Se trata de una unidad arquitectural, ya que se comporta como un servicio REST que puede ser consumido por diferentes clientes que necesiten tener un proceso de gestión de usuarios de manera general.

4. Propuesta y contexto

El componente existe en el marco de un desarrollo para un proyecto interno a una dependencia de la Universidad, para esto, se dividieron los requerimientos en 7 grupos, los cuales debían crear un componente ya sea para el Back end o el Front end del proyecto. El trabajo estuvo enfocado en el Back end, el cual a través de un modelado en Python se logro crear un componente genérico para la gestión de usuarios, el cual tenia como principal atributo de calidad, garantizar la seguridad de la información que iba a ser generada, a través de un protocolo de encriptación.

5. Problema

El problema al ser tratado con este componente puede derivar en diferentes implementaciones, puesto que se trata de un servicio REST expuesto para diferentes clientes.

El lenguaje de programación puede jugar un papel importante en dicha implementación, por lo que por este motivo la solución planteada más adelante puede desembocar en algo minimalista pero que aborda las especificaciones dadas por los requisitos.

6. Aplicabilidad

6.1 Ventajas

- Registro de usuarios con un alto grado de seguridad.
- Autenticación con tokens de expiración, por lo que el usuario cada vez que realice un logueo, tendrá diferente token de acceso.
- Autorización de recursos, para que usuarios que tengan roles no permitidos para acceder a un recurso en específico, no lo puedan hacer.

6.2 Desventajas

- Las funcionalidades estarían disponibles para bases de datos relacionales. Con NO-SQL podría llegar a presentar problemas debido a que el mapeo no estaría orientado a tablas, sino a documentos.

7. Descripción

7.1. Estructura de solución

El modelo de componentes llevado a cabo se encuentra relacionado con la implementación de un componente como SERVICIO REST, el cual presenta una comunicación con una base de datos para llevar a cabo todo este proceso de gestión de usuarios.

La estructura básica del componente se puede ver reflejada en la siguiente gráfica:

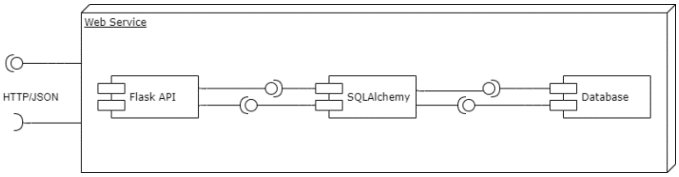

Figura 1. Diagrama de Componentes.

7.2 Solución – Estrategia

Para solucionar los requisitos planteados anteriormente se utilizó el lenguaje de programación Python, el cual permitió la implementación del componente back-end a través de los siguientes subcomponentes o frameworks:

- Flask: Es un framework minimalista para crear servicios web a través de rutas y da la posibilidad de exponerlo como una API REST. [1]
- SQLAlchemy: Es una capa de abstracción de base de datos común y un mapeador relacional de objetos. [2]
- Base de datos: Es el gestor de información para la administración de los usuarios. [3]

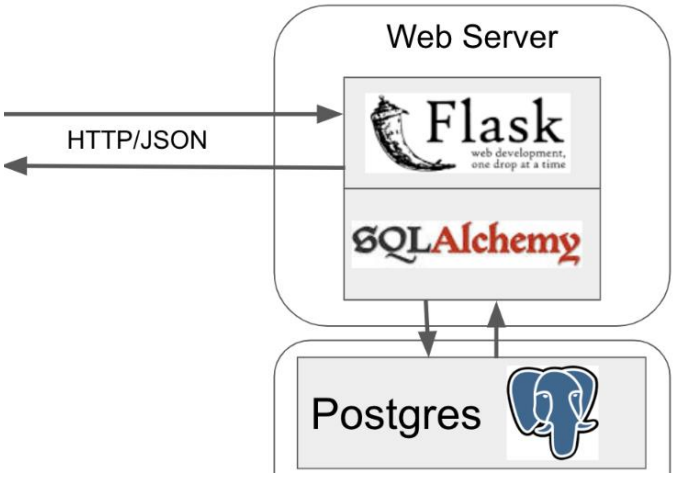

Figura 2. Diagrama de alto nivel del servicio web.

7.3 Implementación

La solución se encuentra basada en los siguientes pasos:

1. Configuración de la base de datos con SQLAlchemy.
2. Creación de tabla con roles.
3. Especificación de los modelos tanto de usuario con los campos id, nombre de usuario, contraseña, rol, nombre completo, título comercial, nombre de la empresa, cargo que ocupa, correo electrónico y número de teléfono.
4. Creación de métodos para llevar a cabo proceso de autenticación por tokens, por lo que se tuvo en cuenta la creación y validación del token como la respectiva encriptación de la contraseña.
5. Creación de endpoints que permiten la creación de usuarios en la base de datos, añadir rol, autenticar al usuario y permitirle acceder a ciertos recursos con una autorización especificada.

8. Interfaces

El componente se pone a disposición a través de un API Rest la cual puede acceder a diferentes end points.

Los protocolos HTTP/JSON serían aquellas interfaces que finalmente podrían enviar peticiones de tipo GET, POST, PUT y DELETE.

9. Fuerzas

Las razones por las cuales se podría usar este componente son:

- Un alto grado de seguridad al permitir la encriptación de contraseñas en la base de datos.
- Un acceso a diferentes bases de datos relacionales como podría ser MySQL, PostgreSQL, SQLite, entre otros.

10. Características de calidad y subcaracterísticas

El atributo de calidad que prima es la seguridad, debido a que la gestión de usuarios debe garantizar el buen uso de la información.

11. Ejemplo de código

A partir de la necesidad de seguridad, se genera un proceso de encriptación, para evitar que usuarios mal intencionados accedan a la base de datos y obtengan información sensible.

```
app.config['SECRET_KEY']                                =
os.environ.get('SECRET_KEY',   '3df9bbf3-1a43-43db-
9ebb-f395cf555491')
```

Se puede ver la configuración de cualquier base de datos relacional y una por defecto en caso de no especificar ninguna así:

```
app.config['SQLALCHEMY_DATABASE_URI']                =
os.environ.get('DATABASE_URI',
'sqlite:///db.sqlite')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS']         =
False
  app.config['SQLALCHEMY_COMMIT_ON_TEARDOWN']
  = True
```

La creación de una API a través de Flask es relativamente sencilla y se puede hacer de la siguiente manera:

```
from flask import Flask
app = Flask(__name__)
```

Finalmente, para la creación de un endpoint que permita en este caso, como muestra el ejemplo, llevar a cabo un proceso de autorización, podría ser la siguiente:

```
from flask import Flask, request, abort, jsonify

@app.route('/api/authenticate', methods=['POST'])
def authenticate_api():
   user_name = request.json.get('user_name')
   password = request.json.get('password')

   if None in (user_name, password):
      abort(400)
   user                                              =
User.query.filter_by(user_name=user_name).first()

   if user is None:
      abort(400)

   if not user.authenticate(password):
      return jsonify({
         'authenticated': False
      }), 200
```

```
    return jsonify({
        'authenticated': True,
        'auth_token': user.get_token()
    }), 200
```

12. Variantes del componente

Por el tipo de componente puede ser implementado en cualquier uso comercial, debido a que la gestión de usuarios es un proceso necesario para cualquier sistema.

13. Componentes relacionados

Este componente se puede relacionar con cualquier tipo de aplicaciones, además de componentes Front end como una interfaz de usuario, e inclusivo componentes Back end que necesiten una autenticación antes de ser utilizados.

14. Repositorio del componente

El código fuente del componente se encuentra en GitHub y disponible para su uso público. El enlace al mismo es https://github.com/stivenramireza/user-management-component.

V. CONCLUSIÓN

Como pudimos observar el desarrollo de software basado en componentes tiene grandes ventajas a la hora de gestionar un sistema, debido a que permite una fácil escalación y reutilización, además el uso de Python permitió hacer el desarrollo de una manera minimalista, sin comprometer el cumplimiento de los requisitos.

VI. BIBLIOGRAFÍA

[1] Flask (2020). Flask, web development, one drop at a time. Available in https://flask.palletsprojects.com/en/1.1.x/.

[2] SQLAlchemy (2020). The Python SQL Toolkit and Object Relational Mapper. Available in https://www.sqlalchemy.org/.

[3] Full Stack Python. Databases. Available in https://www.fullstackpython.com/databases.html.

[4] Paludo, M “et al” (2011) Applying pattern structure to document and reuse components in components based software engineering environments.