Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Grado en Ingeniería Informática

Trabajo de Fin de Grado

Medallas digitales para el ámbito educativo

Digital badges for the educational field

Alfredo Martín Belda Sosa

La Laguna, 8 de septiembre de 2019

D. **Jesús Miguel Torres Jorge**, con N.I.F. 43826207-Y profesor contratado Doctor de la Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas, como tutor.

D. **José Demetrio Piñeiro Vera**, con N.I.F. 43774048-B profesor titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas, como cotutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

“Medallas digitales en el ámbito educativo”

ha sido realizada bajo su dirección por D. **Alfredo Martín Belda Sosa**,

con N.I.F. 79065162-V.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 8 de septiembre de 2019

Agradecimientos

En primer lugar quiero agradecer a mis familiares y amigos personales por su apoyo y ayuda en el proceso de desarrollo de este trabajo de fin de grado.

También agradecer al personal docente de la Escuela Superior de Ingeniería y Tecnología por su disponibilidad y trato personal a lo largo de todos estos años, resolviendo mis dudas en aquellas asignaturas en las que más dificultades he encontrado además de aportarme multitud de conocimientos de alto valor.

Especial mención a Jesús, que me ha tutorizado en este trabajo. Su trato ha sido inmejorable, ofreciéndome su ayuda cada vez que la necesitaba con una explicación detallada para cualquier pregunta o dificultad que he encontrado por el camino.

También, como no, a todos aquellos compañeros y amigos que me han aportado risas, inspiración y motivación durante todo este tiempo.

En estos años han habido épocas de la carrera en las que entre el tiempo que he pasado en clase más el que he pasado estudiando en la biblioteca he hecho más vida en la universidad que en otro sitio. Es por ello que debo dirigir también parte de estos agradecimientos al personal de servicios, por unas instalaciones de calidad donde poder estudiar y una cafetería que me ha mantenido en pie a base de cafés en los días más largos.

Con esta memoria se cierra una etapa que deja atrás buenos y malos momentos, ambos me aportan experiencias de gran valor que agradezco haber tenido.

A partir de aquí comienza algo nuevo, un camino lleno de oportunidades y desafíos que sólo puede ser igual o mejor que el que ya he recorrido.

Gracias.

Licencia

Resumen

La finalidad de este trabajo de fin de grado ha sido el diseño y desarrollo de una aplicación para crear, emitir y mostrar insignias digitales. Cumple un estándar internacional de insignias digitales denominado "Open Badges", que describe un conjunto de especificaciones para permitir que los receptores de insignias puedan portarlas en un archivo de imagen. Mediante portales web que soportan este estándar, las imágenes pueden verificarse como insignias válidas y ser compartidas en redes sociales. La funcionalidad de la aplicación se consigue mediante la integración con el chat de Slack: una herramienta de comunicación para equipos de trabajo accesible a través de la web. La aplicación añade varios comandos al chat para realizar las acciones descritas y visualizar el resultado.

Se espera que la aplicación contribuya a los procesos de aprendizaje, dando a los profesores la posibilidad de emitir insignias digitales para premiar conocimientos y competencias adquiridas por parte de los alumnos. En el desarrollo se han aplicado metodologías ágiles para crear una aplicación funcional, modular y mantenible.

Esta memoria comienza con una introducción donde se pretende poner en situación al lector. A continuación se desarrolla con una descripción detallada de las tecnologías, patrón de diseño y estructura de la aplicación, y se cierra con dos apartados que exponen las conclusiones, líneas futuras y el presupuesto.

Palabras clave: Insignia, Digital, Open, Badges, Slack, Aprendizaje, Aplicación.

Abstract

The purpose of this final degree project has been the design and development of an application to create, issue and display digital badges. It meets an international standard for digital badges called "Open Badges", which describes a set of specifications to allow badge receivers to carry them in an image file. Through web portals that support this standard, images can be verified as valid badges and shared on social networks. The functionality of the application is achieved by integrating with Slack chat: a communication tool for work teams accessible through the web. The application adds several commands to the chat to perform the described actions and visualize the result.

The application is expected to contribute to the learning processes, giving teachers the ability to issue digital badges to reward knowledge and skills acquired by students. In the development, agile methodologies have been applied to create a functional, modular and maintainable application.

This report begins with an introduction where it is intended to put the reader in situation. It is then developed with a detailed description of the technologies, design pattern and structure of the application, and closes with two sections that expose the conclusions, future lines and the budget.

***Keywords**: Badge, Digital, Open, Badges, Slack, Learning, Application.*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

1.1 Antecedentes y estado actual del arte

Tradicionalmente, las medallas, insignias o recompensas han sido usadas por todo tipo de organizaciones como forma de verificar la distinción honorífica de un individuo. Entre sus usos encontramos premios en eventos y competiciones deportivas, condecoraciones militares o civiles como reconocimiento de un servicio al país, o galardones como el Premio Nobel para reconocer las contribuciones a un campo de conocimiento o algún servicio humanitario. En todos estos casos la idea siempre es la misma: verificar que un miembro de un grupo u organización ha logrado un mérito notable.

El concepto de medallas digitales nace del mundo de los videojuegos. En 2005, la empresa Microsoft anuncia un sistema de logros virtuales para consolas conocido como el ***Xbox Gamerscore****[1]* mediante el cual se permite que los jugadores obtengan trofeos virtuales por completar una tarea o desafío. Cada trofeo incrementa una puntuación global que se compara con la de otros jugadores. El factor competitivo que este sistema añade a la experiencia jugable ha supuesto un gran éxito y ha dado paso a la popularidad de las medallas digitales.

En el ámbito educativo existe una técnica que consiste en trasladar las mecánicas de los juegos con el fin de obtener mejores resultados en la enseñanza. Esta técnica se conoce como "**ludificación**" o "**gamificación**". De este concepto nace “**Open Badges**”, un proyecto que persigue regular la emisión de medallas digitales para que sean verificables.

Open Badges[2] es un proyecto que reúne un conjunto de especificaciones para comunicar habilidades y logros mediante imágenes que contienen datos verificables. Fue creado en 2011 por la organización **Mozilla**[3] y un grupo de colaboradores como una manera de verificar el aprendizaje de una habilidad en cualquier lugar. En la actualidad, el desarrollo de las especificaciones del estándar Open Badges está dirigido por los miembros de **IMS Global Learning Consortium**[4], una organización sin ánimo de lucro a la que pertenecen universidades, colegios, asociaciones, y otras organizaciones del ámbito de la educación.

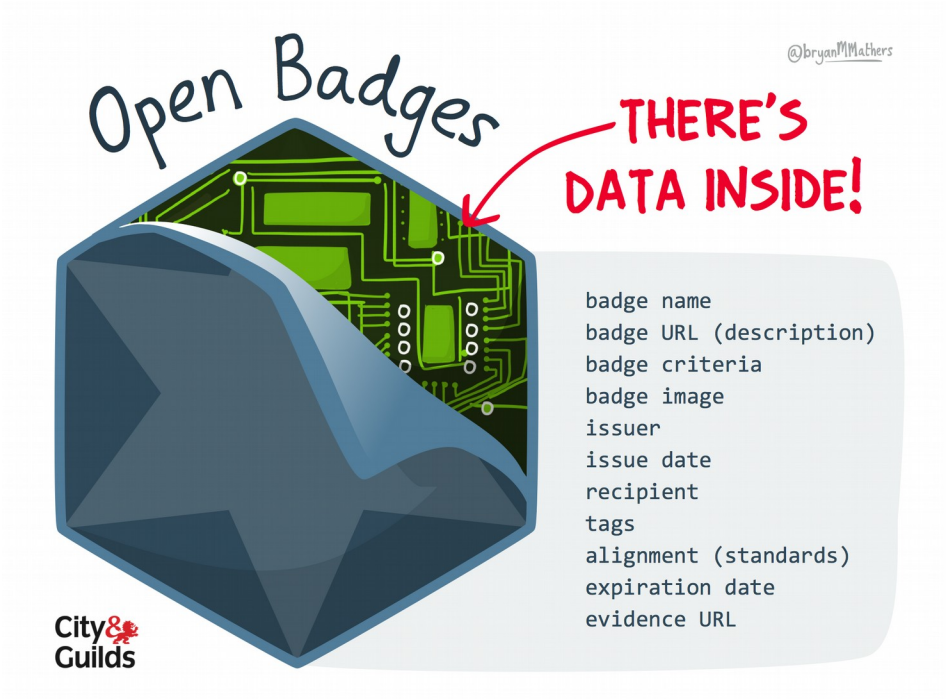

Figura 1.1: "Open Badges Peeled" por Bryan Mathers, representación de una medalla Open Badges

(CC-BY-ND)

Una medalla “Open Badges” es una imagen que contiene información acerca de quién, cuándo, por qué y para qué se emitió. Esta información está codificada en la medalla en forma de **metadatos**. Señala entre otros datos un dominio de Internet que emitió la medalla, y el correo electrónico del receptor, permitiendo a quien la recibe portar la medalla a través de la web de manera verificable.

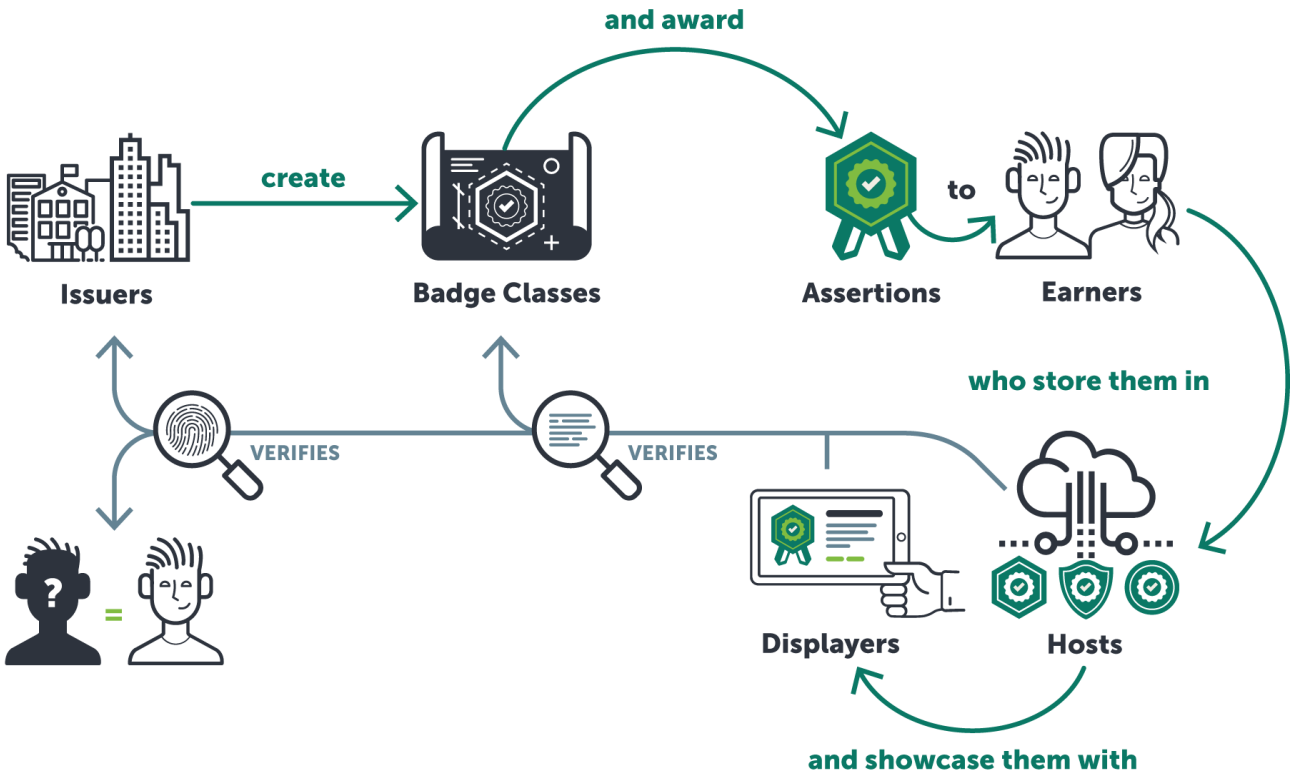

Figura 1.2: Implementación de Open Badges. Fuente: "Open Badges 2.0 Implementation Guide IMS Final Release"

En términos generales, el proceso consiste en que una **entidad emisora** aloje las medallas que puede emitir en archivos con un formato que define el estándar Open Badges. Cuando se emite una medalla a un usuario, se crea una **asociación** de los datos de la medalla con los datos del usuario. Para permitir su comprobación, los datos de esta asociación se guardan en la entidad emisora y en los metadatos de un archivo de imagen que se envía al usuario. Existen servicios web como **Badgr**[5] que soportan el estándar, permitiendo subir este tipo de imágenes a su portal, donde son verificadas y pueden compartirse en redes sociales.

Debido al carácter virtual de Open Badges, para emitir una medalla es necesario un **canal de comunicación** por el que enviar la imagen resultado del proceso. En el ámbito educativo es interesante usar herramientas de comunicación en las que puedan participar los miembros de una asignatura para obtener y compartir información o resolver dudas.

Slack es una herramienta que facilita la colaboración en equipos de trabajo mediante un chat[6]. Cada usuario de Slack puede crear espacios de trabajo e invitar a otros usuarios a unirse. Los espacios de trabajo están organizados en canales y el administrador del espacio de trabajo puede crear nuevos canales. Lo interesante de Slack es que proporciona herramientas de integración permitiendo

desarrollar **aplicaciones** que se integran en el servicio.

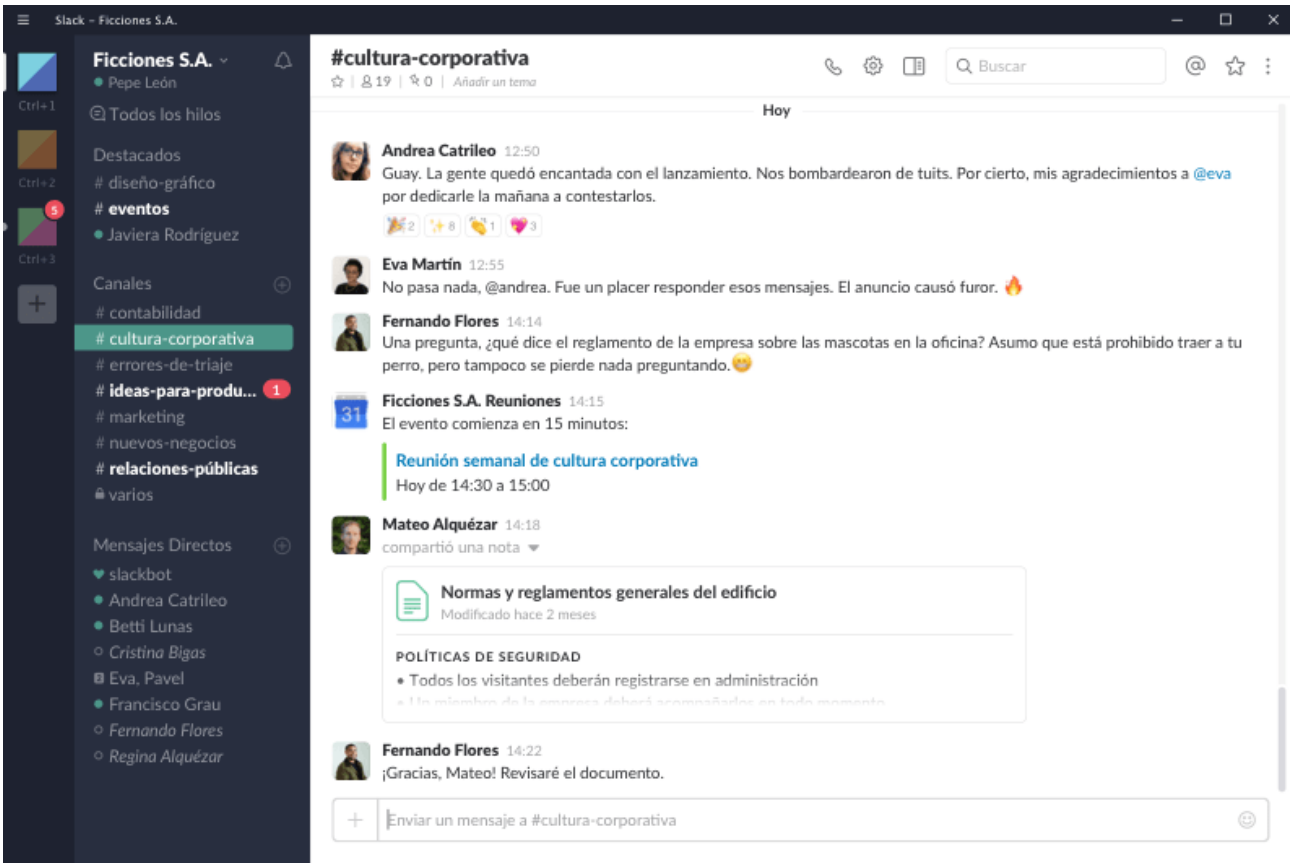

Figura 1.3: Aspecto de un espacio de trabajo de Slack. Fuente: “Slack para Windows”

El **objetivo** de este Trabajo de Fin de Grado es la creación de una aplicación para emitir medallas digitales que siguen el estándar en un espacio de trabajo de **Slack**. Para ello se usará el chat como interfaz, en donde se podrán usar una serie de **comandos** para emitir y listar medallas. Para poder verificar las medallas, el estándar pide alojar la información en un dominio, por lo que habrá que crear un **servicio web** mediante el cual la aplicación actuará como **entidad emisora**. Por otro lado, la aplicación tendrá una **interfaz de administración** que podrá ser accedida mediante línea de comandos para crear medallas y administrar los permisos que tiene cada usuario del espacio de Slack para usar los comandos.

Se espera que la aplicación contribuya al **aprendizaje** de los alumnos en las asignaturas de la Universidad donde los profesores deseen usar Slack como forma de comunicación, fomentando el aprendizaje a través del diseño y emisión de **medallas verificables** que recompensen el logro de una tarea o aportación.

1.2 Objetivos

En este apartado se describirán los objetivos clave a lo largo del proyecto

Ámbito	Objetivo
Diseño	Diseñar la estructura lógica de la aplicación
Desarrollo	Validación de los datos al crear una medalla
Desarrollo	Gestión de las imágenes de las medallas
Desarrollo	Comando para listar medallas en Slack
Desarrollo	Comando para emitir una medalla en Slack
Desarrollo	Comando para listar las medallas emitidas a un usuario en Slack
Desarrollo	Comando para mostrar ayuda
Desarrollo	Interfaz de administración
Desarrollo	Línea de comandos para administrar aplicación
General	Redactar la memoria del TFG

Tabla 1.1: Descripción de los objetivos

1.3 Descripción de los objetivos

- **Estructura lógica de la aplicación**: antes de comenzar a desarrollar el proyecto, es importante definir una estructura lógica para usar buenas prácticas de desarrollo y facilitar el futuro mantenimiento de la aplicación. En este caso se ha optado por hacer uso de la arquitectura hexagonal, que se describirá con detalle en el siguiente capítulo.
- **Línea de comandos para administración**: este objetivo ha consistido en diseñar una pequeña aplicación de línea de comandos en python que permite realizar tareas administrativas en la aplicación principal, accediendo a su interfaz de administración.
- **Validación de los datos al crear una medalla**: debido a que la administración se realiza de manera externa a la aplicación, es importante validar los datos para que no se introduzca información errónea y para prevenir acciones de usuarios desautorizados.
- **Gestión de las imágenes de las medallas**: debido al consumo de recursos que supone abrir una imagen para leerla, es importante que solo se realice esta tarea cuando se solicite específicamente la imagen. Como las imágenes están asociadas a las medallas es necesario una implementación que optimice este caso de uso, para evitar abrir una imagen cuando sólo se va a acceder a los datos de la medalla.
- **Comando para listar medallas en Slack**: en este punto comienza la integración de la aplicación con Slack, implementando un comando que solicita qué medallas hay disponibles y las muestra en el chat de Slack.
- **Comando para emitir una medalla en Slack**: mediante este comando, se obtiene la información del usuario de Slack en la cual está su correo electrónico y se asocia a una fecha y medalla para emitirla con el estándar de OpenBadges.
- **Comando para listar las medallas emitidas a un usuario de Slack**: una vez emitida la medalla se pueden mostrar en Slack, las que ha recibido un determinado usuario.
- **Comando para mostrar ayuda**: en cualquier aplicación destinada a los usuarios es importante tener una guía de uso

rápida. Este comando sirve para tal objetivo.

- **Interfaz de administración**: es necesaria una interfaz protegida mediante la cual el administrador de la aplicación pueda realizar acciones como crear medallas o modificar permisos de los usuarios.
- **Redactar la memoria del TFG**: una vez terminada la aplicación se escribirá la memoria donde se describirá el proceso de desarrollo de la aplicación y toda la información acerca de la misma.

Capítulo 2
Metodología y desarrollo

2.1 Tecnologías y herramientas usadas

La aplicación ha sido desarrollada íntegramente en el lenguaje de programación **Python**[7]. Este lenguaje cuenta con un amplio conjunto de módulos desarrollados por la comunidad de software libre que proveen recursos útiles para desarrollar aplicaciones.

El funcionamiento de la aplicación se sirve del framework web **aiohttp**[8] que usa un paradigma de programación asíncrono introducido en la versión 3.5 de Python y su librería **asyncio**. Permite desarrollar aplicaciones monohilo concurrentes usando corrutinas, multiplexando la entrada/salida sobre sockets, ejecutando clientes y servidores web y otras primitivas relacionadas [9].

Para usar los puertos web, requeridos por la aplicación, hacen falta permisos de administrador. Ejecutar la aplicación como administrador sería un riesgo de seguridad, por ello se usa un servicio **NGINX** que actúa de puente entre el puerto web y el de la aplicación.

Ha sido necesaria una herramienta de gestión de paquetes para instalar sin conflicto los módulos que se han usado en la aplicación: **Pipenv**. Se trata de una herramienta que permite gestionar un entorno virtual para el proyecto, instalar o desinstalar paquetes y seleccionar la versión adecuada de cada librería para evitar conflictos en el árbol de dependencias[10].

Se ha usado el editor de texto **Vim** para escribir el código de la aplicación, la versión mejorada del editor Vi que está presente en todos los sistemas UNIX[11].

El desarrollo de la aplicación ha sido efectuado en un **servidor Ubuntu** que ha proporcionado el servicio IaaS de la Universidad.

Además, también se ha utilizado la API REST de Slack[12], mediante la cual es posible que la aplicación reciba y envíe

mensajes a Slack.

En el proceso de desarrollo ha sido crucial el uso de un sistema de control de versiones, el cual ha sido **Git**[13].

Para la comunicación con el tutor se ha usado la plataforma **GitHub**[14], a través de la cual se han discutido y aceptado las nuevas funcionalidades y cambios a medida que se iban introduciendo mediante el sistema de "pull requests". El **código fuente** de la aplicación puede encontrarse en el repositorio de GitHub[15].

2.2 Arquitectura hexagonal

La aplicación ha sido desarrollada con un patrón de diseño conocido como **arquitectura hexagonal**[16].

La arquitectura hexagonal es un patrón de diseño de software que ayuda en los procesos de construcción y desarrollo de un sistema. Se trata de una arquitectura que cumple los principios **SOLID**[17] de la arquitectura de software, permitiendo separar responsabilidades entre capas y definir reglas de dependencia entre ellas.

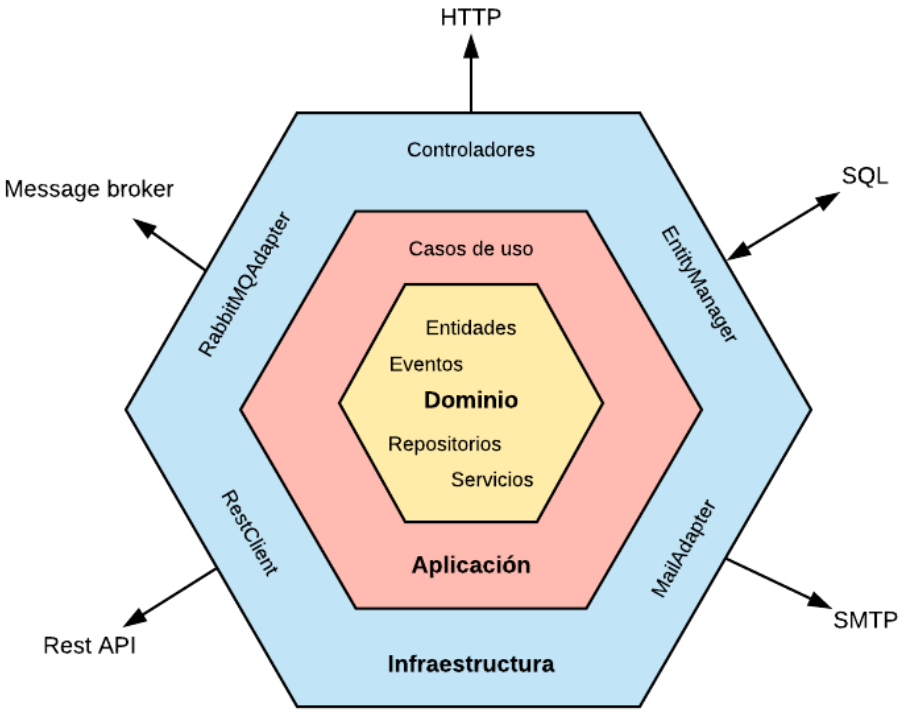

Figura 2.1: Esquema de la arquitectura hexagonal.

En esta arquitectura se aísla la lógica del modelo de negocio en un núcleo interior que puede ser accedido usando distintos tipos de cliente o sistemas. El acceso se realiza a través de puertos, que son el punto de entrada, y adaptadores, que son las implementaciones de tecnologías concretas como bases de datos o clientes de correo electrónico.

En la Figura 2.1 se puede observar la representación típica de la arquitectura hexagonal, donde cada uno de los lados del hexágono interior representa un puerto hacia dentro o fuera de la aplicación. Por ejemplo, un puerto puede ser usado por un adaptador que implementa una **API REST** para hacer peticiones a la aplicación.

El núcleo de las capas es el **dominio**, donde se definen las entidades que intervienen en los casos de uso de la aplicación. Cada una de las capas de la arquitectura es independiente de cualquier elemento externo.

En la capa de **aplicación** se encuentran los **servicios**, que implementan los casos de uso. Para acceder a los servicios se requieren interfaces que definan las reglas de acceso y proporcionen los puntos de entrada o salida al centro de la lógica. A esto se le llama **puertos** y podemos encontrar dos tipos: primarios y secundarios.

Un **puerto primario** es la definición de una interfaz mediante la cual la aplicación recibe comandos de un **adaptador primario**. Los puertos primarios permiten a los adaptadores primarios usar los servicios para modificar, borrar o crear elementos en las capas interiores.

Un **puerto secundario** es una interfaz para usar un **adaptador secundario**. En la mayoría de casos un puerto secundario es una clase abstracta que debe ser implementada. A diferencia de un puerto primario, un puerto secundario es la interfaz para que un servicio use un adaptador secundario, por ejemplo para guardar una entidad en el sistema de persistencia al crearla.

En el caso de los puertos secundarios, se usa la **inyección de dependencias** para mantener las capas lo más desacopladas posibles.

La inyección de dependencias es una técnica de diseño de software donde un objeto proporciona las dependencias a otro objeto. Para ello es necesario definir una interfaz que indique los métodos que deben ser implementados. Esta interfaz es usada en el código y su implementación es “inyectada” en tiempo de ejecución. De esta manera se evita utilizar una implementación

concreta en múltiples lugares del código, facilitando el reemplazo de un componente por otro y la reutilización de código.

Capítulo 3
Aplicación

En este capítulo se describe todo lo relacionado con la estructura y funcionalidad de la aplicación.

3.1 Estructura

Como se describió en el apartado 2.2, la aplicación sigue un patrón de diseño que consiste en puertos y adaptadores.

Figura 3.1: Estructura de directorio

La aplicación principal está en el módulo **slack_badges_bot** y tiene cuatro submódulos: *adapters, entities, services* y *utils.*

3.1.1 Entidades

Las entidades son la unidad más pequeña de la lógica con la que se puede operar, representa los recursos de la aplicación y son elementos de la capa de dominio. Están definidas como clases en el submódulo *entities.*

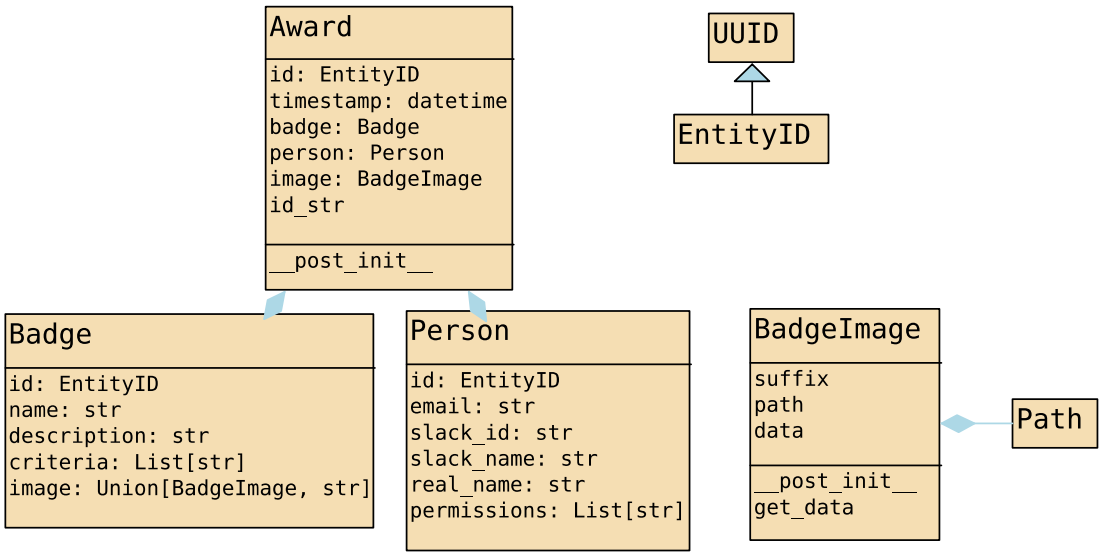

Figura 3.2: Entidades

- **EntityID**: todas las entidades heredan esta clase que define un método para generar identificadores únicos. En la aplicación cada entidad tiene un campo “id” que se usa para identificar de manera única al recurso.
- **Badge**: representa una medalla que contiene los criterios para ganarla, la descripción, su nombre y un atributo que guarda la imagen en una clase “BadgeImage”.
- **Person**: esta clase representa un usuario de la aplicación. Almacena su información de Slack y los permisos que tiene con la nomenclatura del sistema de permisos que se detalla en el apartado 3.5.
- **Award**: esta entidad representa la emisión de una medalla. Tiene como atributos una entidad “Person”, una entidad “Badge”, una fecha, y una imagen con los metadatos que permiten verificar la emisión.
- **BadgeImage**: aunque no es una entidad porque no tiene identificador, esta clase está en el submódulo *entities* porque proporciona una interfaz para guardar las imágenes. Para evitar tener que cargar la imagen cuando se recupera una entidad, proporciona el método get_data(), que puede ser llamado si la tarea requiere específicamente cargar la imagen.

3.1.2 Servicios y puertos

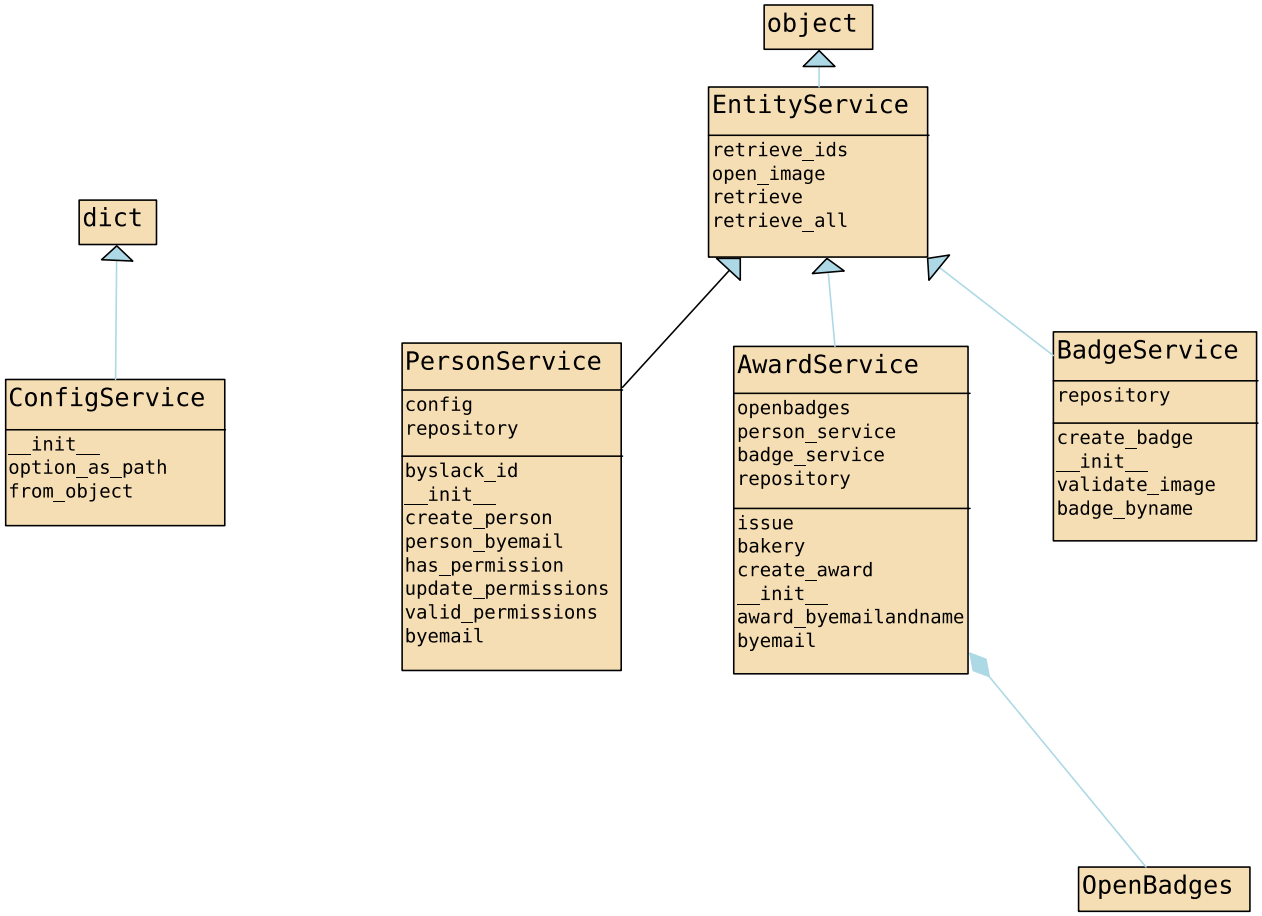

Figura 3.3: Servicios y puertos

Los servicios son clases que implementan los casos de uso de las entidades de la aplicación y proporcionan un acceso al centro de la lógica. Son por lo tanto los que implementan los **puertos** y están en el submódulo *services*.

EntityService

Es una clase abstracta con algunos métodos definidos. Los servicios que operan con entidades heredan esta clase. Tiene cuatro métodos:

- *open_image()*: permite cargar la imagen de una entidad "Badge" o "Award".
- Métodos que comienzan con *retrieve():* llaman a los **puertos secundarios** para usar la persistencia.

Cada clase que hereda EntityService debe tener un atributo ***repository***. Este atributo es una instancia de *EntityRepository*

(apartado 3.1.3) que es la clase que define la interfaz con la persistencia, es decir, un puerto secundario.

AwardService

Servicio para gestionar las emisiones de medallas, representadas por entidades “Award”. Sus métodos son:

- *award_byemailandname()*: permite recuperar una medalla que se ha emitido a una persona con el correo electrónico buscado.
- *bakery()*: es usado para crear las imágenes de medalla con la información codificada en los metadatos.
- *byemail()*: sirve para recuperar todas las medallas asociadas a un determinado correo electrónico.
- *create_award():* este método crea una entidad Award y la almacena.
- *issue()* usa los métodos *bakery()* y *create_award()* para emitir una medalla.

BadgeService

Es un servicio para gestionar la creación y recuperación de medallas, representadas por entidades “Badge”. Sus métodos son:

- *badge_byname()* permite recuperar una medalla buscándola por su nombre.
- *create_badge()* y *validate_image()*: se usan para crear medallas y verificar que la imagen cumple las normas para imágenes de medalla de Open Badges.

PersonService

Este servicio implementa los casos de uso sobre entidades “Person”. Los métodos son:

- *byemail()* y *byslack_id():* permite recuperar una persona mediante su email o su id de Slack respectivamente.
- create_person(): sirve para crear entidades Person y guardarlas con el sistema de persistencia.
- *has_permission()*: permite comprobar si una persona tiene un permiso.
- *update_permission()*: permite modificar los permisos de una persona.
- *valid_permission()*: permite comprobar si una cadena cumple con la estructura del sistema de permisos.

Estos tres últimos métodos son usados por la interfaz de administración para gestionar los permisos.

ConfigService

Este servicio proporciona los parámetros de configuración iniciales al resto de componentes de la aplicación.

3.1.3 Puertos secundarios

La interfaz con la persistencia es un puerto secundario, debido a que son los servicios los que usan la interfaz en sus métodos.

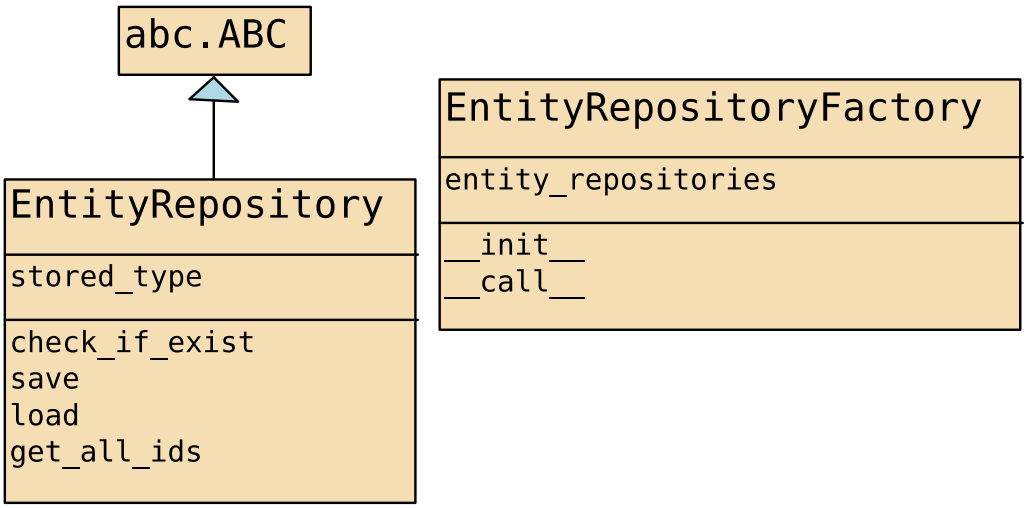

Figura 3.4: Puertos secundarios

EntityRepository

Esta clase define la interfaz de un repositorio de entidades y es un puerto secundario. Es una clase abstracta mediante la cual los servicios usan la persistencia. La implementación de la persistencia es “inyectada” en tiempo de ejecución mediante la técnica de inyección de dependencias.

EntityRepositoryFactory

Es una clase que recibe un conjunto de repositorios de entidades cuando se instancia. Define el método __call__, que permite recuperar el repositorio de un tipo de entidad. En el apartado 3.1.7 se aclara su uso con un ejemplo de código.

3.1.4 Adaptadores primarios

Los adaptadores primarios están en el módulo *adapters* y hay tres de ellos:

- *OpenBadgesWebService*: implementa un servicio web para mostrar las entidades como objetos JSON[1] con los requisitos de Open Badges. Explicado en el apartado 3.2.
- *SlackApplication*: implementa el cliente que permite a la aplicación integrarse en Slack. Se explica con detalle en el apartado 3.4.
- *WebService*: implementa el servicio web para administrar la aplicación. Se explica con detalle en el apartado 3.6.

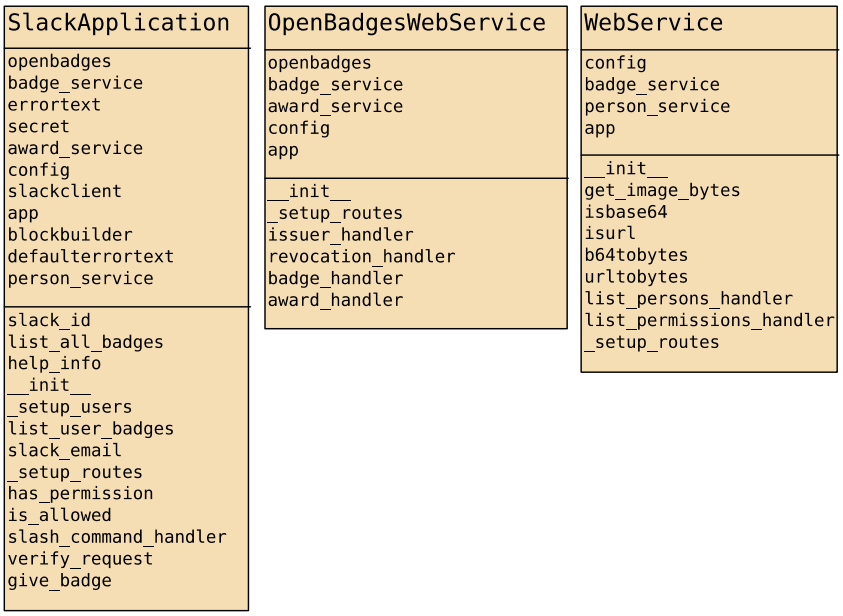

Figura 3.5: Adaptadores primarios

1 JSON (JavaScript Object Notation) es un formato de texto que está constituido por una colección de pares nombre/valor. Cada nombre es una cadena de caracteres y los valores pueden ser números, cadenas de texto, booleanos, listas de valores y otras colecciones de pares nombre/valor. https://www.json.org/

3.1.5 Adaptadores secundarios

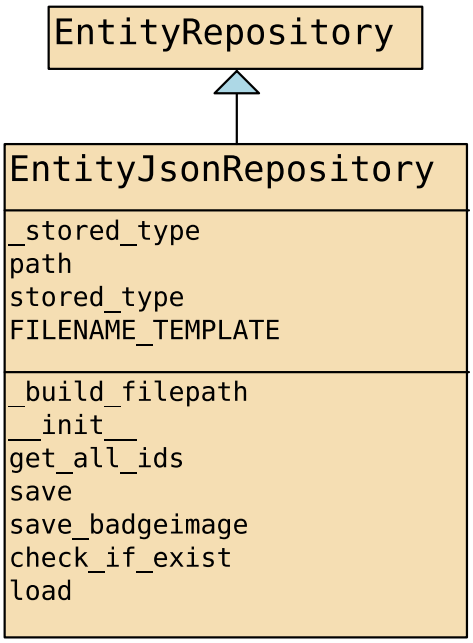

Figura 3.7: Implementación de la persistencia

Los adaptadores secundarios también están en el submódulo *adapters* y en concreto hay un adaptador secundario: EntityJsonRepository. Se trata de una clase que implementa la interfaz *EntityRepository*. Posteriormente se "inyecta" en la aplicación mediante la técnica de inyección de dependencias. El apartado 3.1.7 ilustra este proceso en un ejemplo de código.

3.1.6 Utilidades

Las utilidades son clases auxiliares que se utilizan en los adaptadores. Se encuentran en el módulo *utils*.

En este módulo hay dos utilidades:

- **blockbuilder.py**: implementa la clase *BlockBuilder***,** usada por *SlackApplication*. Permite dar formato a los mensajes que se envían a Slack, siguiendo el formato definido para los mensajes que envían las aplicaciones a Slack[18].
- **openbadges.py**: implementa la clase *OpenBadges*, usada por el adaptador *OpenBadgesWebService*. Permite convertir las entidades en diccionarios que cumplen los requisitos de Open Badges.

3.1.7 Programa principal

El programa principal está en el fichero **app.py** y se encarga de interconectar las relaciones de dependencia de las capas de la aplicación.

BadgeService tiene una dependencia con *EntityRepositoryFactory* para obtener el repositorio que guarda las entidades del servicio.

```
class BadgeService(EntityService):
    def __init__(self, entity_repository_factory: EntityRepositoryFactory):
        self.repository = entity_repository_factory(Badge)
```

Figura 3.8: Dependencias de BadgeService en el código de services/badge.py

EntityRepositoryFactory, a su vez, depende de una lista de objetos *EntityRepository*.

```
class EntityRepositoryFactory:

    def __init__(self, entity_repositories: List[EntityRepository]):
        self.entity_repositories = entity_repositories
```

Figura 3.9: Dependencias de EntityRepositoryFactory en el código de adapters/repositories.py

El módulo **punq** es una librería para la inyección de dependencias, el cual permite registrar las dependencias en un contenedor que las resuelve posteriormente con la llamada a un método.

```
container = punq.Container()
container.register(services.repositories.EntityRepository,
                   adapters.repositories.EntityJsonRepository,
                   stored_type=entities.Badge,
                   path=config.option_as_path('DATA_PATH') / 'badges')

container.register(services.repositories.EntityRepository,
                   adapters.repositories.EntityJsonRepository,
                   stored_type=entities.Person,
                   path=config.option_as_path('DATA_PATH') / 'persons')

container.register(services.repositories.EntityRepository,
                   adapters.repositories.EntityJsonRepository,
                   stored_type=entities.Award,
                   path=config.option_as_path('DATA_PATH') / 'awards')
```

Figura 3.10: Registrando dependencias en el código de app.py

En la Figura 3.10 se registran las implementaciones de *EntityRepository*. Posteriormente se resuelven mediante el método *resolve()* de **punq**, usado por ejemplo al instanciar *SlackApplication*:

```
app.add_subapp('/slack', adapters.slack.SlackApplication(
      config=config,
      badge_service=container.resolve(services.badge.BadgeService),
      award_service=container.resolve(services.award.AwardService),
      person_service=container.resolve(services.person.PersonService)
  ).app)
```

Figura 3.11: Resolución de dependencias en app.py

BadgeService se registró previamente en el contenedor, y ahora **punq** sabe cómo resolver las dependencias *BadgeService* y *EntityRepositoryFactory* (figuras 3.8 y 3.9).

De esta manera se abstrae la relación de dependencias a un único lugar del código, facilitando posibles reemplazos de un sistema de persistencia por otro. Las implementaciones de la persistencia siempre deberán respetar la interfaz de *EntityRepository*.

Al ejecutar el script **app.py** se resuelven las dependencias, se pasan los parámetros de configuración a los servicios y se unen los tres adaptadores primarios (la interfaz de administración, la interfaz de Open Badges y la interfaz de Slack) en una aplicación.

El framework **aiohttp** permite ejecutar estos tres servicios web en un sólo hilo de ejecución de forma concurrente y asíncrona.

3.2 Interfaz de Open Badges

En Open Badges, una medalla emitida se representa con varios objetos que se transmiten a través de Internet en formato JSON[1]. La estructura de cada objeto está definida en las especificaciones de Open Badges[19]. Esta interfaz implementa el estilo de arquitectura de software denominado **API REST**, un tipo de arquitectura que permite obtener los datos de una aplicación mediante el protocolo HTTP utilizando identificadores para cada recurso.

La interfaz de Open Badges es un servicio web que expone los objetos en URL's[2] creadas de forma dinámica. Cada medalla y cada

1 JSON (JavaScript Object Notation) es un formato de texto que está constituido por una colección de pares nombre/valor. Cada nombre es una cadena de caracteres y los valores pueden ser números, cadenas de texto, booleanos, listas de valores y otras colecciones de pares nombre/valor. (RFC 8259)

2 Una URL (Uniform Resource Locator) es una secuencia de caracteres que identifica de forma única un recurso físico o virtual y describe el mecanismo para acceder a él a través de una red. (RFC 3986)

emisión tiene un identificador único dentro de la aplicación. Con este se construye la URL de la interfaz para exponer el objeto. Cuando el identificador coincide con el de una entidad, esta se transforma dinámicamente en formato JSON mediante la clase auxiliar *OpenBadges* y se responde a la petición con los datos generados.

Se trata de una “traducción” de entidades del dominio a los objetos que definen las especificaciones de Open Badges. Esto se hace para que, en caso de que se desee usar un estándar distinto a Open Badges, las entidades del dominio no tengan dependencia del estándar.

3.2.1 Badge Class

Según las especificaciones, una medalla se representa mediante un objeto “Badge Class”. Este objeto tiene los siguientes pares nombre/valor:

Nombre	Valor
name	Nombre de la medalla.
description	Descripción de la medalla.
image	Imagen que representa el logro. Puede tener formato Datos URI[1] o ser URL donde se aloja.
criteria	Criterios para conseguir la medalla. Puede ser una lista de valores o una URL.
issuer	URL que aloja un objeto Issuer Organization.

Tabla 3.1: Estructura de un objeto "Badge Class"

Cuando se solicita una URL cuya terminación comienza por **openbadges/badges/** seguida del identificador de una entidad “Badge” y terminada con **/json**, se “traduce” esta entidad en una “Badge Class” y se envía a través de Internet.

```
https://vituin-chat.iaas.ull.es/openbadges/badges/0ee0b33017d7488b9fa9e58c7c91b4dc/json
```

Figura 3.12: Ejemplo de URL de una “Badge Class”

1 Los datos URIs son una nomenclatura que permite incorporar datos binarios de pequeños archivos en un documento que va a ser transmitido por Internet. Para ello se codifican en base 64 y se incorporan en una cadena de caracteres. (RFC 2397)

Para cada entidad “Badge” hay tres URL's que siguen la misma estructura pero con el final de su terminación diferente. A parte de **/json** también se responde a:

- **/image**: muestra la imagen de una entidad “Badge”.
- **/criteria**: muestra los criterios necesarios para ganar la medalla.

Estas URL's se utilizan en los campos de la “Badge Class” y se pueden apreciar el ejemplo de la siguiente figura.


```
{
        name: "medalla de plata",
        description: "descripcion",
        criteria:
"https://vituin-chat.iaas.ull.es/openbadges/badges/0ee0b33017d7488b9fa9e58c7c91b4dc/criteria",
        image:
"https://vituin-chat.iaas.ull.es/openbadges/badges/0ee0b33017d7488b9fa9e58c7c91b4dc/image",
        issuer: "https://vituin-chat.iaas.ull.es/openbadges/issuer"
}
```


Figura 3.13: Ejemplo de una “Badge Class” en el navegador

3.2.2 Issuer Organization

El campo “issuer” del objeto “Badge Class” es una URL que aloja un objeto “Issuer Organization”. Este tipo de objeto contiene la información del emisor de una medalla y tiene la siguiente estructura:

Nombre	Valor
name	Nombre del emisor de la medalla.
URL	URL de la página principal del emisor.
description	Descripción del emisor.

Tabla 3.2: Estructura de un objeto "Issuer Organization"

La interfaz de Open Badges devuelve este objeto cuando se solicita la siguiente URL con terminación **/openbadges/issuer**:

```
https://vituin-chat.iaas.ull.es/openbadges/issuer
```

Figura 3.14: URL del emisor

3.2.3 Badge Assertion

Cuando se emite una medalla en la aplicación, se crea una entidad del dominio: “Award”. La interfaz de Open Badges expone estas entidades como objetos “Badge Assertion”, que tienen la siguiente estructura:

Nombre	Valor
uid	Id de la emisión de la medalla.
recipient	Datos del beneficiario de la medalla, correo electrónico.
badge	URL que aloja la “Badge Class” que describe a la medalla emitida.
verify	URL que aloja este objeto “Badge Assertion”
issuedOn	Fecha en formato UTC en que se emitió la medalla
image	URL que aloja la imagen de la medalla emitida

Tabla 3.3: Estructura de un objeto "Badge Assertion"

Para acceder a un objeto “Badge Assertion” se debe solicitar una URL con terminación **/openbadges/award/** seguido del **identificador** y terminado por **/json**. Al igual que los objetos “Badge Class” también se puede terminar la URL con **/image** para obtener la imagen de la medalla que tiene los metadatos sobre su emisión.

3.3 Metadatos en las imágenes

Las medallas de Open Badges se pueden portar como archivos de imágenes que incluyen el objeto “Badge Assertion” en forma de metadatos. De esta manera, las imágenes se pueden guardar y mostrar en cualquier lugar.

Codificar estos metadatos en la imagen de la medalla se denomina “cocinar la imagen”. Cuando el usuario sube una imagen de medalla “cocinada” a una página preparada para leer este estándar, el software puede comprobar si la medalla que sube esa persona es legítima[20].

3.4 Interfaz de Slack

La interfaz de Slack se implementa mediante el adaptador primario *SlackClient* y consiste en un servicio web que atiende peticiones de Slack.

En un espacio de trabajo de Slack se ha configurado un “Slash Command”, que es un mecanismo para que Slack envíe datos a una aplicación. Este “Slash Command” está configurado para que al escribir el comando /**badges** se envíe una petición desde Slack a ***https://vituin-chat.iaas.ull.es/slack***, que es donde está esperando peticiones *SlackClient.*

Cuando se activa este mecanismo, se envía una carga de datos que contiene detalles acerca de quién lo envió, desde qué espacio de trabajo y qué canal, el texto adicional con el que se escribió el mensaje, etc. [21]

Distinguiendo el texto que va detrás del comando **/badges**, se implementan tres comandos en *SlackClient:*

- **/badges list all**: este comando muestra todas las medallas que hay disponibles en el espacio de trabajo.

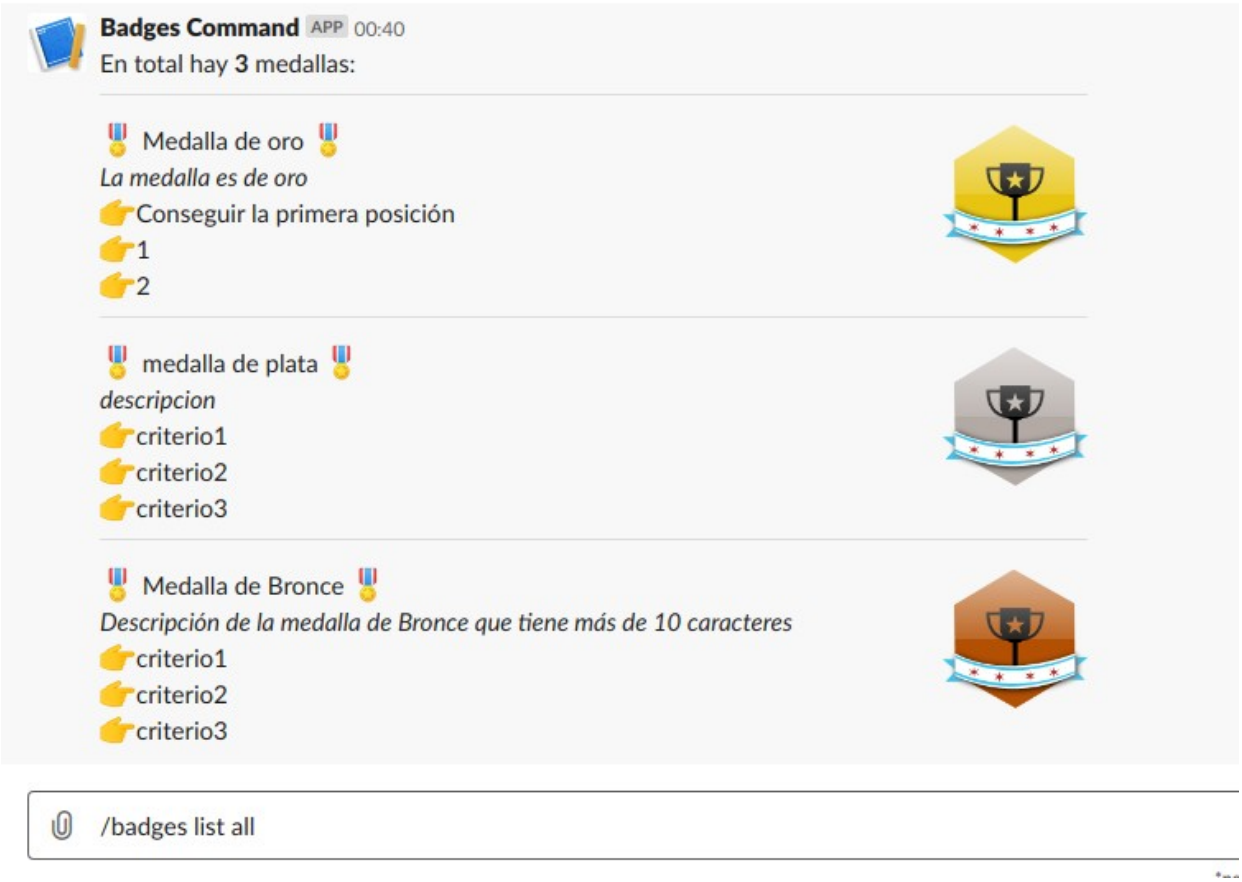

Figura 3.15: Comando /badges list all

- **/badges give [@USUARIO] [NOMBRE DE LA MEDALLA]**: con este comando se emite una medalla a un usuario. El servidor se encarga de meter los metadatos en una imagen y enviarla por el chat de Slack, posteriormente se puede verificar la medalla.

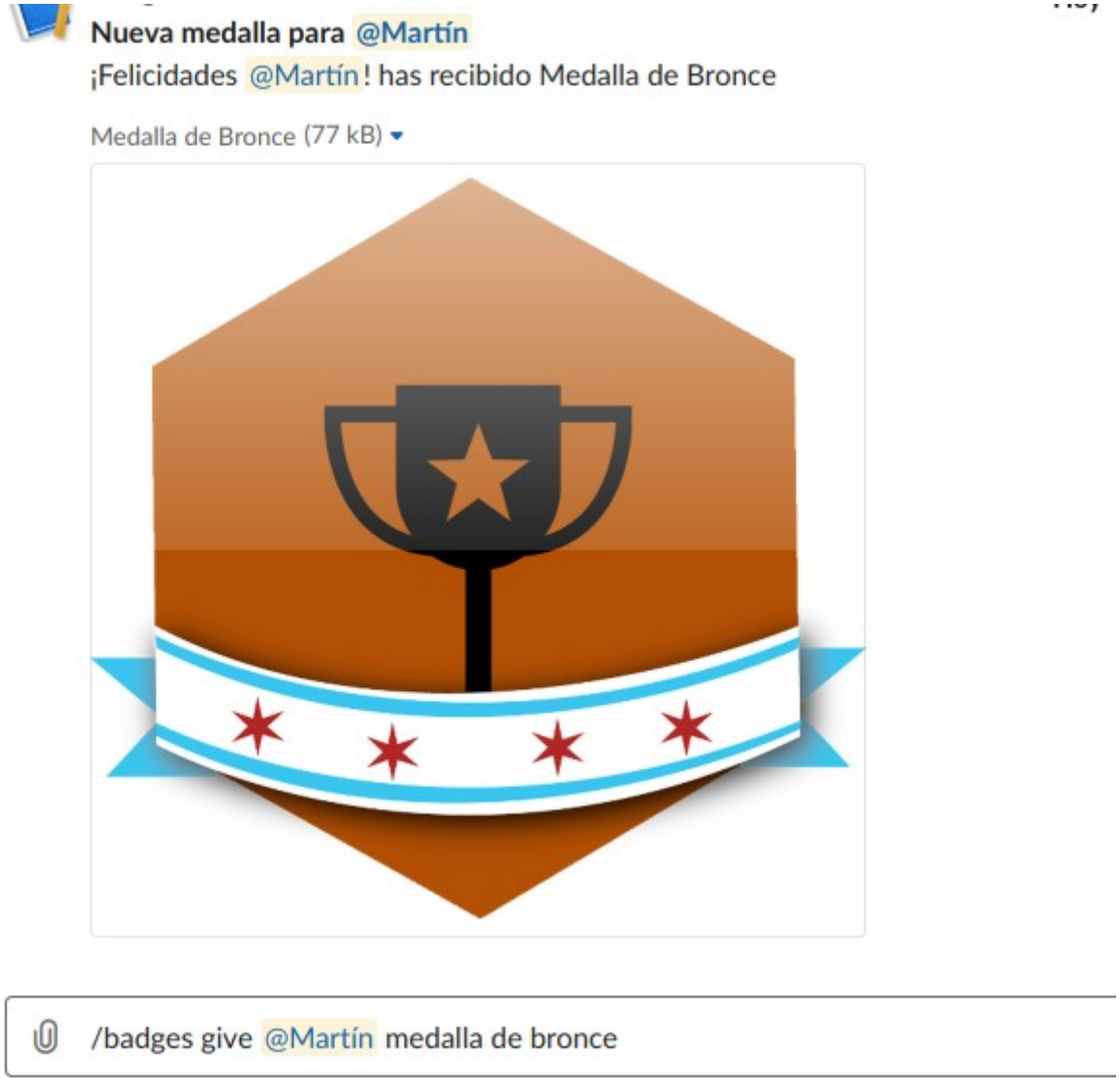

Figura 3.16: Comando /badges give

- **/badges list [@USUARIO]**: este comando permite listar las medallas que tiene un usuario del espacio de trabajo.

Figura 3.17: Comando /badges list [@USUARIO]

3.5 Sistema de permisos

El sistema de permisos consiste en un conjunto de permisos para restringir el uso de los comandos de la aplicación en slack. Cada permiso es una cadena de caracteres que se ha inspirado en los "Slack Scopes"[22]. Para cada comando hay permisos que un usuario necesita tener para poder ejecutarlo.

La cadena de caracteres de un permiso está compuesta por tres campos que están separados por el signo de puntuación doble de la siguiente manera:

```
recurso:permiso:ámbito
```

Figura 3.18: Campos de un permiso

- Recurso: este campo representa el nombre general del recurso al que está asociado ese permiso, por ejemplo una medalla o la asociación de una medalla.
- Permiso: este campo representa el permiso que se da sobre un

recurso, por ejemplo listar o crear.

- Ámbito: indica en qué ámbito se aplica el permiso, puede ser sobre los recursos que pertenecen al usuario que tiene el permiso o sobre todos los recursos. Se puede omitir para indicar que el ámbito es sobre todos los recursos.

En la siguiente tabla hay un recurso por cada columna que se asocia con una acción sobre ese recurso por cada fila:

	Medallas disponibles	**Medallas del usuario**	**Medallas de otros**
Crear	-	awards:create:self	awards:create:others
Listar	badges:list	awards:list:self	awards:list:others

Tabla 3.4: Sistema de permisos

A continuación, se describen los recursos:

- **Medallas disponibles**: son las medallas que la aplicación tiene disponibles para emitir tantas veces como se desee, no se pueden crear mediante la interfaz de Slack, por lo que se indica que no existe el permiso con un **guión**. Pueden ser listadas por un usuario si tiene el permiso **badges:list**.
- **Medallas del usuario**: son las medallas emitidas que el usuario de slack ha recibido. Un usuario puede emitir medallas para sí mismo si tiene el permiso **awards:create:self**. Para listar sus medallas, el usuario necesita el permiso **awards:list:self**.
- **Medallas de otros**: son las medallas emitidas a otros usuarios del espacio de slack. Para emitir medallas a otro usuario se necesita el permiso **awards:create:others**. Para listar las medallas de otro usuario se necesita el permiso **awards:list:others**.

Al iniciar la aplicación se recopila la información de todos los usuarios del espacio de Slack y se guardan como usuarios de la aplicación. Hay establecidas dos políticas de permisos por defecto:

Tipo de usuario	**Permisos**
Administrador del espacio de Slack	Todos los permisos
Usuario normal	badges:list, awards:list:self

Tabla 3.5: Permisos por defecto

- Los usuarios normales tienen permiso para listar las medallas disponibles y las medallas que han recibido.
- El propietario del espacio tiene todos los permisos disponibles.

Cuando se usa un comando en el chat, la aplicación comprueba que el usuario tiene los permisos necesarios antes de ejecutarlo y muestra un mensaje de error si esta comprobación falla.

3.6 Interfaz de administración

La aplicación tiene una interfaz de administración que al igual que la interfaz de Open Badges, sigue la arquitectura **API REST**. En la configuración por defecto, todas las URL cuya terminación comienza por **/api** pertenecen a la interfaz de administración. Para acceder a la interfaz de administración es necesario **autenticarse** con el usuario y contraseña de administrador definidos en la configuración de la aplicación.

3.6.1 Crear una medalla

Terminación de la URL	Método HTTP
/api/badges/create	POST

Tabla 3.6: URL para crear una medalla

La cabecera de esta petición debe tener el campo *Content-Type: application/json*.

En el cuerpo de la petición se envían los siguientes esquemas JSON para indicar la información de la medalla a crear:

1) Esquema que incluye imagen como datos URI's:

```
{
        "name": "nombre de la medalla",
        "description": "descripción de la medalla",
        "criteria": [
        "criterio 1",
        "criterio 2",
        "criterio 3"
        ],
        "image": "data:[<mediatype>][;base64],<data>"
}
```

Figura 3.19: Esquema JSON para crear una medalla (1)

2) Esquema con la URL que aloja la imagen:

```
{
        "name": "nombre de la medalla",
        "description": "descripción de la medalla",
        "criteria": [
            "criterio 1",
            "criterio 2",
            "criterio 3",
        ],
        "image": "https://example.com/image.png"
}
```

Figura 3.20: Esquema JSON para crear una medalla (2)

Es requerido que la imagen de la medalla cumpla con las siguientes características:

- La imagen tiene que tener formato **png** o **svg**.
- El tamaño del archivo no puede ser mayor que **256kb**.
- La imagen debe ser **cuadrada**.
- Los lados deben ser de **90px** o más.

Cuando se intenta crear una medalla mediante la interfaz de administración, se devuelve un error descriptivo si no se cumple algún requisito necesario.

3.6.2 Obtener una lista de personas

Terminación de la URL	Método HTTP
/api/persons/list	GET

Tabla 3.7: URL y método para obtener una lista de personas

Cuando se hace una petición a esta URL, se recibe una lista de personas donde cada elemento es la visualización de toda la información que guarda cada entidad “Person” del dominio.

3.6.3 Obtener una lista con todos los permisos

Terminación de la URL	Método HTTP
/api/persons/permissions/list	GET

Tabla 3.8: URL y método para obtener todos los permisos

Una petición en esta URL muestra una lista completa de los permisos que se le pueden asignar a una persona.

3.6.4 Modificar los permisos de una persona

Terminación de la URL	Método
/persons/permissions/update	POST

Tabla 3.9: URL y método para actualizar los permisos

Para modificar los permisos de una persona se hace una petición POST en esta URL incluyendo en el cuerpo un esquema JSON con la siguiente estructura:

```
{
  "person_id": "id de la persona",
  "permissions": ["permiso", "permiso", "permiso"]
}
```

Figura 3.21: Estructura del objeto JSON que se envía para modificar permisos

3.7 Aplicación de línea de comandos

Para facilitar la usabilidad de la interfaz de administración se ha desarrollado un cliente de línea de comandos con el módulo **click** de python. Esta herramienta se encuentra en la carpeta **badge-cli**.

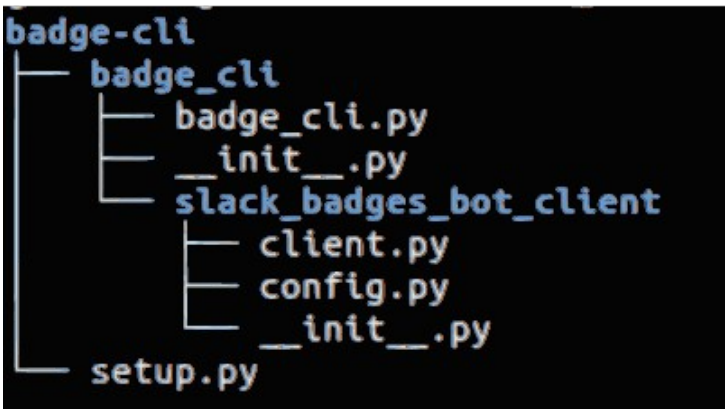

Figura 3.22: Estructura de la aplicación de línea de comandos

Como se puede apreciar en la figura 3.22, se trata de un módulo con el nombre **badge-cli**. Contiene un fichero setup.py, en donde se configura como un módulo que puede ser distribuido con **distutils**, para facilitar la instalación. La configuración de este fichero permite añadir el nombre **badgecli** a la línea de comandos cuando se instala la aplicación.

Al ejecutar este comando sin argumentos, se obtiene un mensaje de ayuda:

```
Usage: badgecli [OPTIONS] COMMAND [ARGS]...

  Comando administrar la aplicacion Slack-Badges-Bot

  Información sobre cada comando:

  badgecli [comando] --help

Options:
  --apihelp  Solicita ayuda de la API
  --help     Show this message and exit.

Commands:
  config  Configurar parámetros de la línea de comandos
  create  Crear medallas.
  list    Listar personas y permisos.
  perm    Modificar los permisos de una ersona.
```

Figura 3.23: Mensaje de ayuda

3.7.1 Configuración

El comando **badgecli config** permite ver o modificar la configuración.

```
Usage: badgecli config [OPTIONS] [PARAMETER]

  Configurar parámetros de la línea de comandos

Options:
  -u, --user      Nombre de usuario
  -p, --password  Contraseña del administrador
  -s, --server    Ruta de administracion del servidor
  -l, --list      mostrar toda la config
  --help          Show this message and exit.
```

Figura 3.24: Mensaje de ayuda del comando badgecli config

Los parámetros por defecto de la configuarción son los que figuran en la siguiente tabla:

Parámetro	Valor por defecto
user	admin
password	password
server	http://localhost/api

Tabla 3.10: Configuración por defecto de la aplicación

Usando las opciones que se ven en la figura 3.24 se pueden modificar estos parámetros.

3.7.2 Creación de una medalla

Para crear una medalla se usa el comando **badgecli create.** Este comando toma como primer argumento un fichero con el esquema JSON que requiere la interfaz de administración y un segundo argumento opcional en donde se puede indicar un archivo de imagen. En caso de no indicar un archivo de imagen se deberá indicar una imagen como URL en el archivo con uno de los esquemas JSON que figura en el apartado 3.6.1. La imagen de la medalla puede crearse usando la herramienta web “Chicago Summer of Learning Badge Studio”[23], pero también se admiten los formatos indicados

```
Usage: badgecli create [OPTIONS] JSON_FILE [IMAGE_FILE]

  Crear medallas.

  Más información en la wiki del proyecto:

  https://github.com/alu0100832211/slack-badges-bot/wiki/Creaci%C3%B3n-de-
  una-medalla

Options:
  --help  Show this message and exit.
```

Figura 3.25: Mensaje de ayuda del comando badgecli create

3.7.3 Listar los usuarios y permisos de la aplicación

Mediante el comando **badgecli list** se pueden listar los usuarios y permisos de la aplicación.

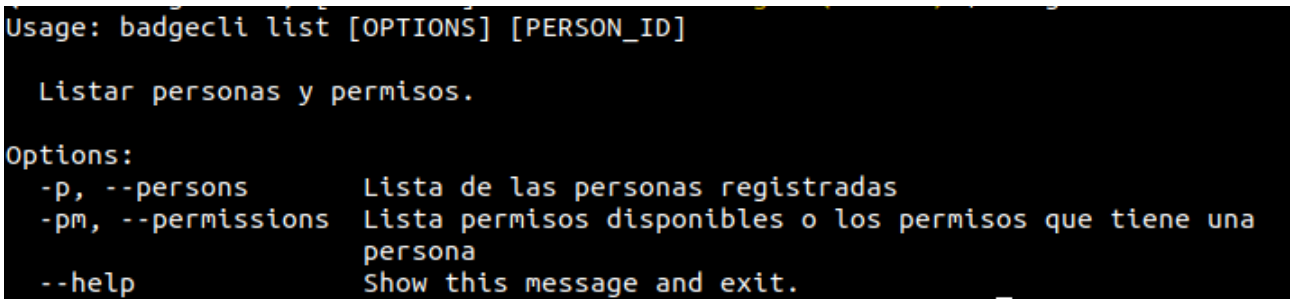

Figura 3.26: Mensaje de ayuda del comando badgecli list

Cada una de las opciones permite obtener una lista general si no se añaden más parámetros. Por ejemplo, al ejecutar **badgecli list -p** se obtiene un resumen de los usuarios de la aplicación en el que se ve su identificador emparejado con su nombre.

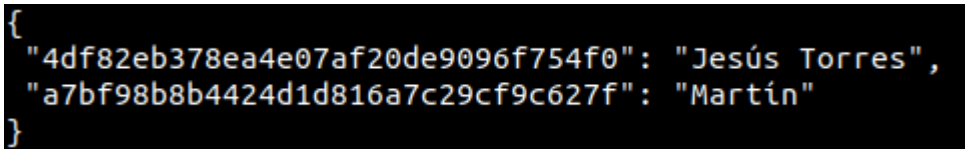

Figura 3.27: Lista resumida de los usuarios de la aplicación

El identificador permite obtener toda la información acerca de una persona. La siguiente figura ilustra un ejemplo del resultado de usar el parámetro **-p** seguido del id de una persona:

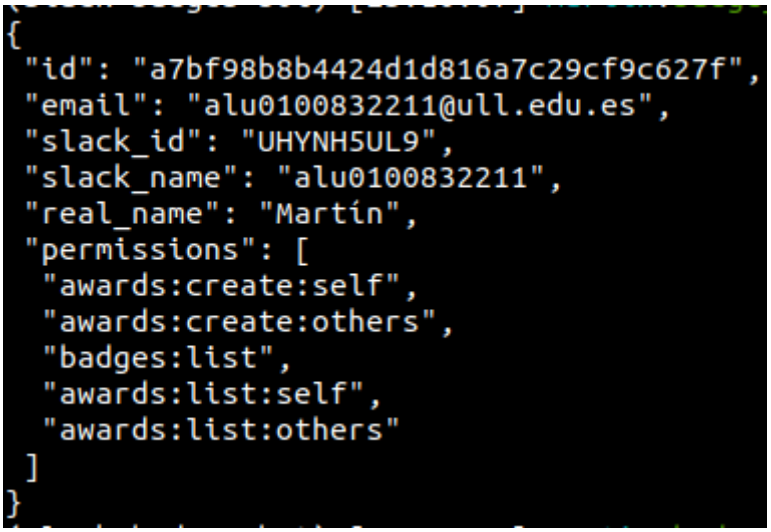

Figura 3.28: Detalles acerca de una persona

3.7.4 Modificar los permisos de un usuario

El comando **badgecli perm** permite ver y modificar los permisos que tienen los usuarios.

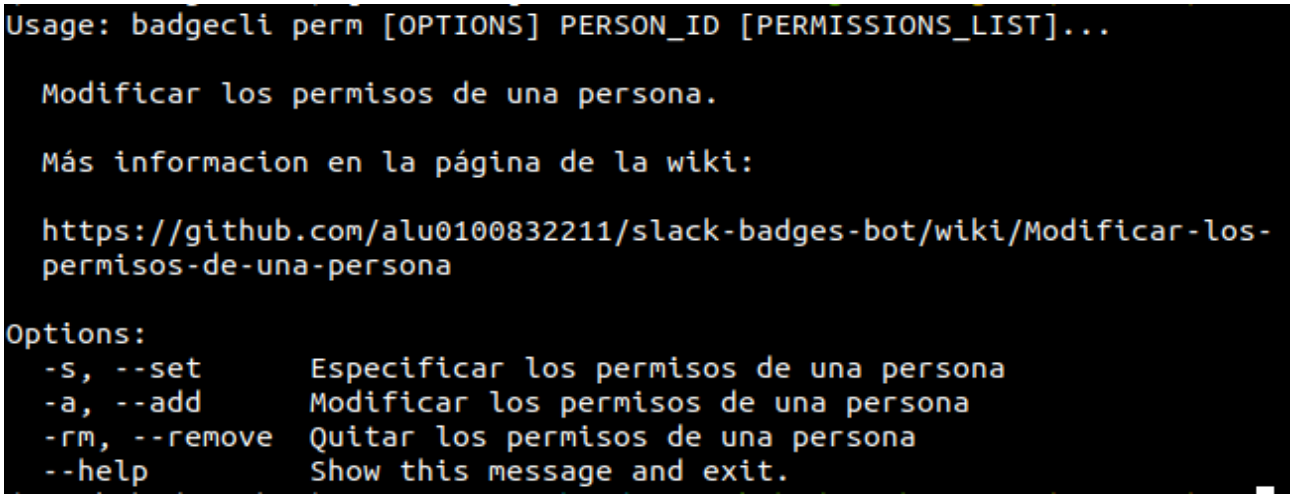

Figura 3.29: Mensaje de ayuda del comando badgecli perm

Este comando permite usar las tres opciones de la figura 3.29 seguidas del “id” de una persona, y una lista de permisos separada mediante espacios.

3.8 Seguridad

Como medidas de seguridad, la aplicación cuenta con las siguientes características:

- **Validación de las peticiones de Slack**

Cuando Slack envía un paquete de petición a la aplicación, incluye en la cabecera un campo con una firma. Esta firma se crea combinando el token secreto de la aplicación con el cuerpo de la petición mediante el algoritmo de hashing **HMAC-SHA256**. El token secreto es una cadena de caracteres larga que tiene tanto el espacio de trabajo de Slack como la aplicación. Este token nunca se transmite a través de la red.

La aplicación combina el cuerpo de la petición recibida con el token secreto de la aplicación, si el resultado es el mismo que la firma enviada por Slack entonces se puede confirmar que el mensaje proviene de los servidores de Slack.

- **Comunicaciones cifradas**

Todas las comunicaciones de la aplicación se cifran con OpenSSL y los datos viajan usando el protocolo HTTPS.

- **Seguridad en la interfaz de administración**

La interfaz de administración está protegida mediante **usuario y contraseña**. Dichas credenciales se definen en la configuración de la aplicación.

Además se usa el módulo **aiohttp_validate** para validar el esquema de los objetos JSON recibidos.

Capítulo 4
Conclusiones y líneas futuras

En términos generales el desarrollo de este trabajo se ha iniciado con una idea simple: crear una aplicación que permita ejecutar varios comandos en un chat de Slack para dar recompensas a los usuarios. Sin embargo, el traslado de esto a la práctica ha presentado todos los obstáculos que son intrínsecos del desarrollo de software. A medida que crece la aplicación también crece su complejidad, es por ello que ha sido crucial mantener una **estructura limpia** y pensar en la **compatibilidad** de las funcionalidades que se iban añadiendo.

Durante este periodo se han cumplido los objetivos establecidos después de mucho trabajo y esfuerzo, sin embargo también se han descartado muchas ideas por falta de tiempo. La versión actual proporciona un diseño **robusto** y **escalable** que permite futuras ampliaciones. En concreto se podrían valorar las siguientes ideas.

- **Interfaz de administración web**: para facilitar la usabilidad, sería conveniente diseñar una interfaz de administración que pueda ser accedida con un navegador web. El módulo **aiohttp_admin** puede ser de gran ayuda para este fin.
- **Diseño de un logo**: la aplicación necesita un logo personalizado para mostrar en el chat de Slack.
- **Lista de revocación**: las especificaciones de Open Badges permiten retirar las medallas que alguien haya recibido si se emiten por error. Para ello hay que implementar una lista de revocación en el servidor y un comando en la interfaz de Slack para permitir esta acción.

El mantenimiento futuro de la aplicación tiene como propósito implementar estas funcionalidades. Las dos tecnologías que une esta herramienta son cada vez ganan más popularidad en el ámbito académico y el desarrollo de su software es una buena manera de ganar conocimientos en el mundo de las aplicaciones web.

Capítulo 5
Summary and Conclusions

In general terms the development of this work has started with a simple idea: creating an application that runs several commands in a Slack chat in order to give rewards to users. However, putting it into practice has presented all the obstacles that are intrinsic to software development. As the application grows its complexity also grows, which is why it has been crucial to maintain a **clean structure** and think about the **compatibility** of the functionalities before adding them.

During this period, after much work and effort, the established objectives have been fulfilled, however many ideas have also been discarded due to the lack of time. The current version provides a **robust** and **scalable** design that allows future extensions. In particular, the following ideas could be valued.

- **Web administration interface**: to facilitate usability, it would be convenient to design an administration interface that can be accessed with a web browser. The **aiohttp_admin** module can be of great help for this purpose.
- **Logo design**: the application needs a custom logo to display in the Slack chat.
- **Revocation list**: Open Badges specifications allow you to withdraw the medals someone has received if they are issued in error. To do this, you must implement a revocation list on the server and a command on the Slack interface that allow this action.

The future maintenance of the application is intended to implement these functionalities. The two technologies that this tool unites are increasingly gaining popularity in the academic field and the development of its software is a good way to gain knowledge in the world of web applications.

Capítulo 6
Presupuesto

Para desarrollar la aplicación, además de las horas de trabajo ha sido necesario un servidor donde desplegarla. Como se comentó en el apartado 2.1, este ha sido proporcionado por el servicio IaaS de la Universidad y su coste se ha determinado a partir de lo que cuesta un servidor del servicio Amazon EC2 con las mismas características[24].

6.1 Desglose de recursos y precios

Elemento	Cantidad	Precio unitario	Precio total
Recursos usados			
Servidor	255 horas	0,05€	12,75€
Desarrollo de la aplicación			
Requisitos	10 horas	22€	220€
Diseño	25 horas	22€	550€
Implementación	200 horas	22€	4400€
Pruebas	30 horas	22€	660€
Mantenimiento	25 horas	22€	550€
Presupuesto total: 6341,75€			

Tabla 6.1: Presupuesto

Bibliografía

[1] Mikael Jakobsson. “Understading Xbox 360 GamerScore”. 6 Febrero 2011. http://gamestudies.org/1101/articles/jakobsson

[2] IMS Global Learning Consortium. “Open Badges 2.0 Implementation Guide IMS Final Release". 18 Octubre 2018. https://www.imsglobal.org/sites/default/files/Badges/OBv2p0Final/impl/index.html

[3] Fundación Mozilla. https://www.mozilla.org/es-ES/about/

[4] IMS Global Learning Consortium. “About the IMS Global Consortium”. https://www.imsglobal.org/aboutims.html

[5] Servicio de visualización “Badgr”. https://badgr.io

[6] Stewart Butterfield, Eric Costello, Cal Henderson, Serguei Mourachov. Slack. https://slack.com/

[7] Guido Van Rossum. Python. https://www.python.org/

[8] Nikolay Kim. "Welcome to aiohttp”. https://aiohttp.readthedocs.io/en/stable/

[9] Guido Van Rossum. Asynchronous IO Support Rebooted: the “asyncio” Module. 12 Diciembre 2012. https://www.python.org/dev/peps/pep-3156/

[10] Kenneth Reitz. “Pipenv: Python Dev Workflow for Humans”. https://pipenv.readthedocs.io/en/latest/

[11] Bram Moolenaar. “Vim – the ubiquitous text editor.” https://www.vim.org/

[12] Slack. “An introduction to Slack apps”. https://api.slack.com/start/overview

[13] Linus Torvalds. Git. https://git-scm.com/

[14] Tom Preston-Werner, Chris Wanstrath, PJ Hyett. Github. https://github.com/

[15] Código fuente de la aplicación descrita en esta memoria. https://github.com/aplatanado/slack-badges-bot

[16] Alistair Cockburn. “Hexagonal Architecture”. https://alistair.cockburn.us/hexagonal-architecture/

[17] Robert C. Martin. “The Principles of Object Oriented Design”. 2003. http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

[18] Slack Technologies. “Introducing Block Kit” https://api.slack.com/block-kit

[19] Fundación Mozzilla. “Open Badges Assertion”. 4 Diciembre 2013. https://github.com/mozilla/openbadges-specification/blob/master/Assertion/latest.md

[20] IMS Global Consortium. “Open Badges Baking Specification. IMS Final Release”. 12 Abril 2018. https://www.imsglobal.org/sites/default/files/Badges/OBv2p0Final/baking/index.html

[21] “What are Slash Commands?” https://api.slack.com/slash-commands#app_command_handling

[22] Slack. “Scopes and permissions”. https://api.slack.com/scopes

[23] Atul Varma (@toolness). Chicago Summer of Learning Badge Studio. http://toolness.github.io/chicago-badge-studio/studio.html

[24] AWS Simple Monthly Calculator. Calculadora de costes de servidor. https://calculator.s3.amazonaws.com/index.html

[25] Mariano Anaya. “Clean Code in Python: Refactor Your Legacy”. 29 Agosto 2018.

[26] Robert C. Martin. “Clean Architecture: A Craftsman’s Guide to Software Structure and Design”. 20 Septiembre 2017.