Universitat Oberta de Catalunya

Raquel Garrido Crespo
Grado en Ingeniería Informática
Java EE

Consultor: Josep María Camps Riba

Profesor responsable de la asignatura: Santi Caballé Llobet

FICHA DEL TRABAJO FINAL

Título del trabajo:	*WorldWideJobs*
Nombre del autor:	*Raquel Garrido Crespo*
Nombre del consultor/a:	*Josep Maria Camps Riba*
Nombre del PRA:	*Santi Caballé Llobet*
Fecha de entrega (mm/aaaa):	01/2021
Titulación:	*Grado en Ingeniería Informática*
Área del Trabajo Final:	*Java EE*
Idioma del trabajo:	*Castellano*
Palabras clave	*Java EE, API REST, Spring*

Resumen del Trabajo (máximo 250 palabras):

Este proyecto tiene como objetivo la creación de una API Rest con la finalidad de llevar a una práctica real y completa los conocimientos y competencias adquiridos durante el grado y en el itinerario de Ingeniería del Software.

El desarrollo de este sistema surge durante la pandemia debida al COVID-19 de 2020, en la cual muchas empresas reticentes a permitir el teletrabajo entre sus empleados se ven obligadas a implantarlo como único medio para seguir funcionando.

El aumento de las políticas de teletrabajo y conciliación que se han venido desarrollando en los últimos años encuentran su punto álgido con la reciente pandemia y cada vez más profesionales que no se sentían satisfechos con su empleo se deciden a lanzarse al mercado de ofertas de trabajo en remoto, empresas y startups para las que cada vez importa menos el horario a cumplir y que valoran mucho más los resultados.

He seguido una metodología de trabajo ágil, en la que se han documentado los requerimientos del sistema a través de historias de usuario y el desarrollo se ha organizado por sprints cortos, en los cuales se abordan el número de tareas adecuadas a la temporalidad del sprint y varían en número según la complejidad estimada de las mismas.

Abstract (in English, 250 words or less):

This project's goal is to develop a Rest API to apply in a real and complete practice, the knowledge and skills acquired during the studies of Computer Science Degree with specialization in Software Engineering.

The development of this system came up during COVID19 pandemic situation in 2020, during this situation most companies were enforced to send their employees home and let them work from their own houses as the only way to maintain business during those moments.

Remote working and work-family balance policies that companies have been developing found their turning point in this pandemic situation and more unsatisfied professionals started looking for new job positions in these remote working conditions or maybe looking for those type of companies that value delivering quality work on time over strict time schedules.

I've followed an agile approach, documenting requirements through user stories and development has been organized through short duration sprints, filled with a number of tasks adjusted to the sprint predicted duration and different number of them based on their estimated complexity.

Agradecimientos

A mi abuela y segunda madre, la gran perjudicada de esta aventura, a la que no he podido dedicarle todo el tiempo que quisiera, te prometo que nos pondremos al día, ahora empieza nuestro proyecto común.

A mi madre, por enseñarme desde pequeña que las cosas sólo se consiguen trabajando mucho, siendo constante y no tirando la toalla, ojalá algún día contemos con más mujeres luchadoras como tú.

A Rubén, por todo el tiempo que no he podido dedicarte, por todo el apoyo y los ánimos.

A Inma e Iñigo, prometo que recuperaremos el tiempo.

A mis tíos Jerónimo y Antonio, por creer en mí y animarme.

A mi tutor durante este trabajo, Josep gracias por tu orientación, tu ayuda, sin tu motivación inicial esta API REST no sería una realidad hoy.

A mi tutora desde el inicio, Isa Lamas, gracias infinitas, no podría haber caído en mejores manos, por tu asesoramiento constante, por tu rapidez, por tu eficacia y por tus ánimos.

A mis compañeros, gracias a esta experiencia he conocido a personas maravillosas a las que hoy puedo llamar amigos, no puedo nombrarlos a todos. Manuel gracias por no dejarme abandonar y tirar de mi cuando quería rendirme. Félix gracias por tu optimismo y tus ánimos constantes.

Índice

Índice de ilustraciones

Índice de tablas

1. Introducción

1.1. Contexto y justificación del Trabajo.

Como programadora inquieta que me considero, a pesar de que no me he visto en situación de desempleo nunca, siempre me he mantenido abierta al mercado laboral y en busca y captura del trabajo perfecto.
Entendiendo el trabajo perfecto por aquel que te permite crecer profesionalmente, te supone algún tipo de desafío, te ofrece la flexibilidad que necesitas y te permite balancear vida profesional y privada.

Las ventajas que ofrece el trabajo remoto creo que son bastantes, como por ejemplo flexibilidad horaria, conciliación, autonomía, etc… pero la principal para mi, es que, si eres una persona de mente inquieta, tienes la posibilidad de cambiar de empresa cuando el proyecto haya perdido interés para ti o ya no suponga un desafío. Facilita enormemente la progresión técnica y evita el estancamiento profesional sin que tengas que trasladar tu residencia constantemente o alejarte de tus seres queridos.

De un tiempo a esta parte y cada vez más, han cobrado relevancia los puestos de trabajo en remoto. Hay estudios que revelan que cada vez más profesionales rechazan ofertas in situ y sólo están dispuestos a abandonar la zona de confort para trabajar en remoto, así mismo revelan que poco a poco se incrementa el porcentaje de profesionales (hasta un 37%) que estarían dispuestos a percibir un salario un poco más reducido a cambio de trabajar permanentemente desde casa.(1)

Este año 2020 ha sido muy complicado a nivel mundial para todos. La pandemia de la COVID-19 ha marcado un antes y un después en la forma de trabajar de muchos sectores.
Algunos negocios se han visto obligados a cerrar sus puertas, otras empresas que no contemplaban la posibilidad del trabajo en remoto ni siquiera a largo plazo, se han visto forzadas a mandar a los empleados a casa y que continuaran trabajando desde allí, poniendo en muchos casos sus propios recursos al servicio de la empresa.(2)
Muchas personas se han visto desempleadas en un corto plazo de tiempo y para los autónomos ha sido una odisea.

Compartimos entre compañeros ofertas de empleo remotas todas las semanas y generalmente, siempre nos quejamos de las mismas carencias sea cual sea la plataforma a la que acudimos a buscar.

Las posibilidades para filtrar ofertas en la mayoría de los casos por zonas horarias son inexactas y difíciles de entender en muchas ocasiones, ¿no sería más sencillo definir una ciudad cualquiera dentro de la zona horaria de partida y verificar si el usuario se ubica a más o menos X horas de esa ciudad? Con el servicio adecuado bastaría con recoger la ubicación del cliente como primer dato de filtrado. Alguna plataforma de las visitadas subsana parcialmente esta forma de presentar la información, pero adolece de la posibilidad de filtrar las ofertas por otros parámetros.

1.2. Objetivos del Trabajo

El objetivo principal del sistema **WorldWideJobs** es ofrecer una **capa de servicio basada en una API REST** que permita a los candidatos filtrar de forma ágil las ofertas de trabajo de la plataforma utilizando más filtros que los que ofrecen las plataformas actuales, especialmente el filtrado por ubicación del candidato y zona horaria de la oferta, además del idioma preferido de trabajo.
Para ello, la publicación de cada oferta requerirá los datos necesarios que permitan posteriormente llevar a cabo este filtrado.

El proyecto a grandes rasgos contempla un escenario para la puesta en práctica de las habilidades y capacidades adquiridas en el itinerario de Ingeniería del Software, desde la definición de las tareas a alto nivel, la documentación de requisitos y la puesta en marcha del entorno tecnológico elegido.

1.2.1. Funcionalidades generales

Definir e implementar una capa de servicio que permita realizar búsquedas de ofertas de empleo en remoto, para ello la capa de servicio deberá contemplar también los casos de alta/baja/modificación de ofertas y empresas.

1. Empresas
    1. Registro de empresas
    2. Listado de empresas
2. Ofertas
    1. Registro de ofertas
    2. Listado de ofertas
    3. Búsqueda y filtrado de ofertas
3. Categorías
    1. Registro de categorías
    2. Listado de categorías
4. Candidatos
    1. Registro de candidatos
    2. Listado de candidatos
    3. Inscripciones a ofertas

El modelado de las ofertas deberá contemplar todos los filtros necesarios para hacer el servicio útil a cualquier cliente que deba consumirlo.

1.2.2. Stack tecnológico

JDK 1.8
Eclipse IDE for Enterprise Java Developers 2019-12

Framework Spring-Boot 2.3.4 (que incluye el servidor tomcat, spring-data, spring-security,...)
Apache Maven
Bitbucket
EGit - Source Tree
NoSQL

Gantt Project
Trello
Magic Draw (licencia UOC)

Justificación del stack tecnológico

He elegido un entorno tecnológico basado en Java porque a pesar del gran auge que están teniendo otros lenguajes de programación, a día de hoy Java sigue siendo uno de los lenguajes líderes en el mercado y lo ha sido históricamente desde hace más de 10 años según el índice TIOBE de Septiembre de 2020[3]

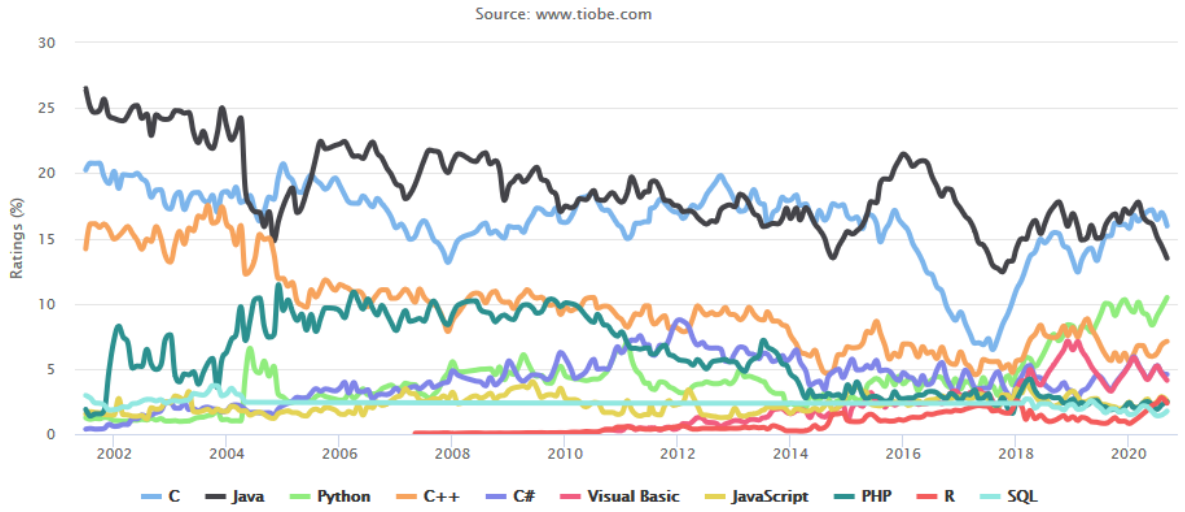


ILUSTRACIÓN 1 – EVOLUCIÓN GRÁFICA DISTINTOS LENGUAJES PROGRAMACIÓN

Very Long Term History

To see the bigger picture, please find below the positions of the top 10 programming languages of many years back. Please note that these are *average* positions for a period of 12 months.

Programming Language	2020	2015	2010	2005	2000	1995	1990	1985
Java	1	2	1	2	3	-	-	-
C	2	1	2	1	1	2	1	1
Python	3	7	6	6	20	20	-	-
C++	4	3	3	3	2	1	2	9
C#	5	5	5	7	9	-	-	-
JavaScript	6	8	8	10	6	-	-	-
PHP	7	6	4	5	19	-	-	-
SQL	8	-	-	-	-	-	-	-
Swift	9	16	-	-	-	-	-	-
R	10	12	52	-	-	-	-	-
Lisp	27	24	15	14	8	6	4	2
Fortran	31	25	24	15	15	4	3	5
Ada	33	27	23	17	17	5	9	3
Pascal	241	15	14	22	16	3	10	6

ILUSTRACIÓN 2 – TABLA EVOLUCIÓN LENGUAJES

Además, las herramientas que proporciona Spring para el desarrollo basado en java ofrecen muchas ventajas, entre ellas:

- Alta escalabilidad gracias a la programación orientada a aspecto
- Simplificación del acceso a capa de datos
- Permite una alta cohesión y bajo acoplamiento
- Utiliza el patrón de inyección de dependencias.
- Spring-data permite trabajar fácilmente con cualquier tipo de base de datos.

He optado por una capa de datos NoSQL porque creo que este sistema necesitará ser escalable en un futuro dado que el volumen de empresas y ofertas de trabajo que puede llegar a almacenar así lo requiere. Además, llevo tiempo con el gusanillo de trabajar con NoSQL, ámbito en el que no tengo experiencia previa y así consigo más aprendizaje para mi con este proyecto.

1.3. Enfoque y método seguido

Con este trabajo pretendo crear un servicio rest que permita la búsqueda de ofertas de empleo orientadas exclusivamente a formas de trabajo remoto, con especial hincapié en la búsqueda por franja horaria. Dado que el periodo para el desarrollo de este trabajo está definido por un semestre académico, desarrollaré un proyecto piloto que cubra las funcionalidades básicas y que podrá ser extendido y mejorado en un futuro.

El método seguido será un desarrollo iterativo basado en metodologías ágiles como SCRUM, con implementación guiada por pruebas (TDD), organizado por diferentes sprints que quedarán documentados en detalle en el capítulo de análisis de la presente memoria.

1.4. Planificación del Trabajo

La planificación viene dada por el calendario de la asignatura TFG en el área Java EE, por tanto, se organiza en cuatro fases, siendo la segunda y la tercera las que conlleven más tiempo de trabajo y dedicación si se siguen las indicaciones del tutor y la memoria se desarrolla en paralelo.

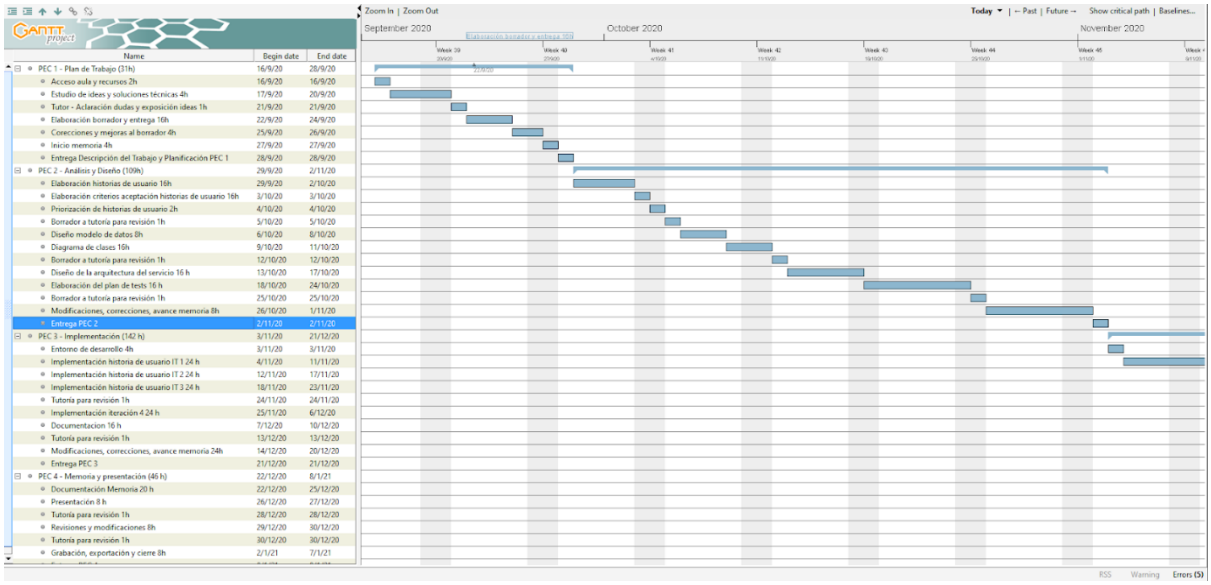


ILUSTRACIÓN 3 – PLANIFICACIÓN GENERAL

1.4.1. Plan de trabajo

En esta primera fase el objetivo principal es establecer un primer contacto con el tutor de la asignatura, validar la idea a desarrollar e implementar como TFG y obtener un plan de trabajo a alto nivel.

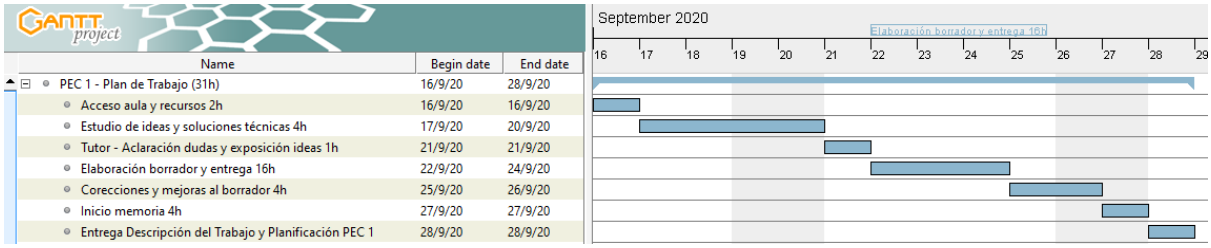


ILUSTRACIÓN 4 – PLANIFICACIÓN PEC 1

1.4.2. Análisis y Diseño

En la segunda fase comenzaremos con el análisis y diseño de la solución que se quiere implementar. Empezaremos por las historias de usuario a validar, desarrollaremos el modelo de datos necesario y elaboraremos el plan de pruebas, definiendo así todos los componentes que integrarán el sistema final.

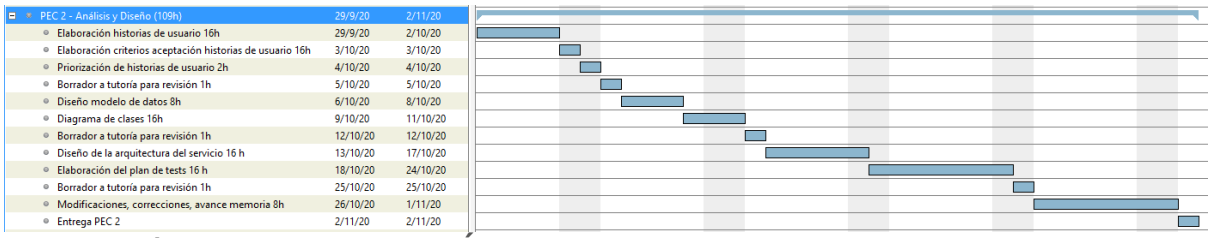


ILUSTRACIÓN 5 – PLANIFICACIÓN PEC 2

1.4.3. Implementación

Ahora toca instalar el entorno de trabajo, crearemos el proyecto inicial, lo subiremos a github y comenzaremos el desarrollo siguiendo la documentación obtenida en el punto anterior.

Seguiremos una metodología de desarrollo orientado a test (TDD) y por iteraciones. Una vez implementada la primera historia, será más sencillo avanzar en el resto de las historias con el aprendizaje adquirido y pudiendo reutilizar algunos componentes. Aplicaremos lo aprendido sobre patrones de diseño y buenas prácticas de codificación para obtener un código mantenible y escalable.

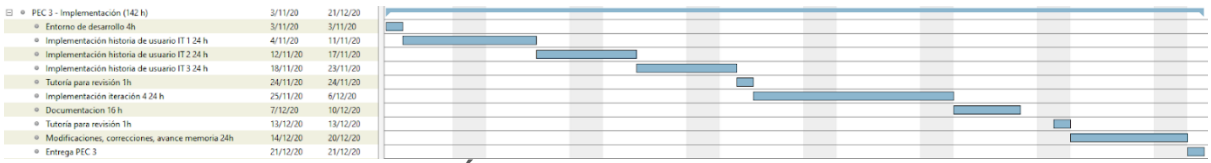


ILUSTRACIÓN 6 – PLANIFICACIÓN PEC 3

1.4.4. Memoria y Presentación

En esta última fase será necesario finalizar toda la documentación de la memoria, que habré llevado un paralelo a lo largo del semestre, elaborar la presentación al tribunal que evaluará mi trabajo e incluir un manual de instalación si me fuera requerido.

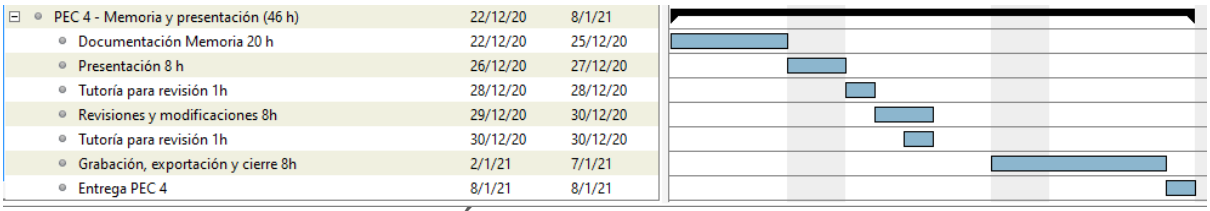


ILUSTRACIÓN 7 – PLANIFICACIÓN PEC 4

1.5. Breve sumario de productos obtenidos

- Plan de Trabajo
  Donde se definen los requisitos funcionales, el stack tecnológico elegido, el objetivo del proyecto y la planificación del temporal por sprints.

- Análisis y Diseño del sistema
  Documento de análisis y diseño del proyecto a desarrollar, con la especificación de los casos de uso, los sprints y su alcance. Además, se incluyen los diagramas de arquitectura y de clases.

- Implementación
  Implementación completamente funcional en cuanto a los requisitos especificados para este trabajo. Además, se genera un manual de instalación y toda la documentación de las pruebas del sistema.

- Presentación y Memoria
  Finalmente, se obtendrá el presente documento completo junto con una presentación del trabajo realizado en diapositivas y un video como presentación virtual del trabajo desarrollado.

1.6. Breve descripción de los otros capítulos de la memoria

- Análisis: En este apartado analizaré los requisitos y funcionalidades necesarias para resolver el problema planteado.

- Diseño: Aquí afrontaré el diseño elegido para implementar el trabajo a desarrollar, todos los componentes necesarios para cubrir los requerimientos, estudiaré el punto de vista de la información y la arquitectura del sistema.

- Implementación: En este capítulo concretaré a más bajo nivel como se ha llevado a cabo la implementación del proyecto en las diferentes capas, detallaré los inconvenientes y sensaciones a lo largo del desarrollo del trabajo y por último indicaré las herramientas utilizadas.

- Pruebas: Este capítulo sirve como introducción y planteamiento del juego de pruebas llevado a cabo, que posteriormente se desarrolla con más detalle en el Anexo de Tests unitarios y pruebas del sistema.

- Mejoras y Evolución del producto: A continuación, detallaré todas las mejoras posibles y soluciones que me hubiera gustado abordar pero que por la limitación temporal de este trabajo, he tenido que dejar fuera del alcance del mismo.

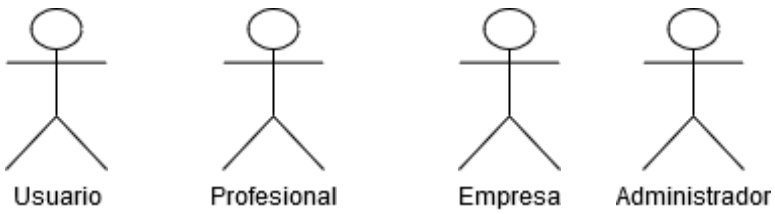
- Conclusiones: En este apartado, comentaré un poco mi perspectiva sobre todo el proceso de elaboración de este trabajo

2. Análisis

En base a los objetivos establecidos en la primera PEC sobre el proyecto a desarrollar, a continuación, identificamos los roles implicados en el escenario, los requisitos a cumplir, los módulos implicados y la arquitectura del sistema.

2.1. Roles

A partir de los requisitos del sistema, tenemos que se contemplan cuatro roles, los cuales se detallan a continuación:



➔ **Usuario**: Es un usuario anónimo de la API REST y que por tanto sólo podrá acceder a los recursos publicados en URI no protegidas y públicas.

➔ **Profesional**: Además de poder acceder a los recursos públicos, podrá acceder a recursos protegidos asociados a su rol, recursos asociados a peticiones POST, PUT y DELETE sobre sus datos como profesional y sus preferencias.

- **Empresa**: Además de poder acceder a los recursos públicos, podrá acceder a recursos protegidos asociados a su rol, recursos asociados a peticiones POST, PUT y DELETE sobre sus datos y sus ofertas.

- **Administrador**: Además de poder acceder a todos los recursos públicos, podrá acceder a recursos protegidos asociados a su rol, tales como peticiones POST, PUT y DELETE sobre categorías y habilidades y PUT y DELETE sobre empresas y profesionales.

2.2. Requisitos no funcionales

El desarrollo del proyecto seguirá un enfoque ágil, basado en SCRUM. Las características de SCRUM que más nos interesan podemos contar:

- Productividad y calidad, que dado el corto tiempo de desarrollo del que disponemos, que el desarrollo que se haga sobre todo sea de calidad
- Su carácter incremental basado en iteraciones cortas y fijas
- La priorización de requisitos de mayor valor para el proyecto.

Si bien REST no es un standard como lo es HTTP, la arquitectura de la API debe seguir las convenciones y guías de desarrollo ampliamente aceptadas por la comunidad.

El sistema trabajará con JSON como formato para transferir la información.

La api se desarrollará siguiendo metodología TDD (Test Driven Development)
La base de datos a usar en previsión de un gran crecimiento será NoSQL

El acceso debe ser seguro
El sistema debe ofrecer unos tiempos de respuesta razonables

2.3. Requisitos funcionales

Según el sistema que queremos desarrollar, para el servicio que queremos ofrecer establecemos los requisitos funcionales que se muestran a continuación organizados en las siguientes categorías:

- **Datos de los profesionales.** Mantenimiento de los profesionales, el cual debe poder registrarse en el sistema, dar de alta una breve introducción de sí mismos y sus datos personales, dar de alta su formación académica, su experiencia profesional, crear alertas sobre posibles ofertas que les interesen e inscribirse a ofertas existentes. Es importante que puedan indicar la zona horaria desde la que trabajan o desean trabajar y si son nómadas digitales. (Profesional)

- **Datos de las empresas.** Mantenimiento de empresas, las cuales deben poder registrarse en el sistema, dar de alta sus datos y gestionar sus ofertas de empleo. (Empresa)

- **Gestión de la oferta de empleo.** Mantenimiento de ofertas, pueden darse de alta ofertas, modificarlas o eliminarlas. Debe poder indicar la franja horaria

para la oferta, el idioma del puesto, la fecha de publicación, la categoría en la que se engloba y las habilidades requeridas para el puesto. (Empresa)

- ➔ **Datos de administración.** El administrador podrá llevar a cabo el mantenimiento de categorías, subcategorías y habilidades. Así mismo podrá desactivar o eliminar, perfiles profesionales y de empresas considerados inadecuados según sus criterios.

- ➔ **Búsqueda de ofertas de empleo.** Consulta de ofertas de empleo, filtradas por categorías, habilidades, idiomas, franjas horarias y para nómadas digitales. (Usuario)
- ➔ **Búsqueda de profesionales.** Consulta de profesionales por ubicación, franja horaria, nómadas digitales, categorías y habilidades e idiomas. (Usuario)

2.4. Historias de usuario

TABLA 1 – HISTORIAS DE USUARIO

Historias de usuario	
US - 001	Cómo profesional quiero registrarme en la plataforma para dar de alta mis datos personales y profesionales y estar visible en la plataforma.
US - 002	Cómo profesional quiero crear alertas sobre publicación de ofertas de trabajo en determinadas categorías y habilidades que poseo para estar al tanto de todas las posibilidades
US - 003	Cómo profesional quiero incluir mi formación académica en mi perfil para competir en el mercado laboral
US - 004	Cómo profesional quiero incluir mi experiencia profesional en mi perfil para competir en el mercado laboral
US - 005	Cómo empresa quiero registrarme en la plataforma para dar de alta mis datos y estar visible en la plataforma
US - 006	Cómo empresa quiero gestionar mis ofertas en la plataforma para captar los mejores profesionales.
US - 007	Cómo usuario quiero realizar búsquedas de ofertas de empleo estableciendo franja horaria, idioma y/o ubicación para encontrar el trabajo deseado.
US - 008	Cómo usuario quiero realizar búsquedas de empresas estableciendo la zona horaria, idioma o ubicación.
	Cómo usuario quiero realizar búsquedas de profesionales estableciendo los idiomas, la franja horaria y/o la ubicación para

US - 009	captar profesionales o comprobar las tendencias del sector.
US - 010	Cómo usuario quiero consultar todas las ofertas de una categoría determinada para estar al día
US - 011	Cómo administrador quiero gestionar categorías, subcategorías y habilidades para tener una clasificación de ofertas consistentes.
US - 012	Cómo administrador quiero poder desactivar perfiles profesionales y de compañías si considero que son inadecuados (descripciones fuera de lugar, etc…) para que no aparezcan en las búsquedas ni listados
US - 013	Cómo administrador quiero poder eliminar perfiles profesionales o de compañías por reincidir en malas prácticas para mantener la reputación de la plataforma

2.5. Especificación de las historias de usuario.

TABLA 2 – DETALLE HISTORIA 001

US - 001	Cómo profesional quiero registrarme en la plataforma para dar de alta mis datos personales y profesionales y estar visible en la plataforma.
Criterios de aceptación	
Dado un profesional que accede al registro en una aplicación cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros email y password	
Entonces el cliente recibe un código de estado 201 (creado)	
Y en la respuesta se recibe la siguiente información: userToken, userId	
API DOC :	
POST	/professional
RESPONSE STATUS	201 Created
RESPONSE	tokenId, userId

PARAMS	
ALTERNATIVE RESPONSE STATUS	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error

TABLA 3 -DETALLE HISTORIA 002

US - 002	Cómo profesional quiero crear alertas sobre publicación de ofertas de trabajo en determinadas categorías y habilidades que poseo para estar al tanto de todas las posibilidades
Criterios de aceptación	
Dado un profesional que desea crear una alerta desde un cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros categoría, habilidades y franja horaria	
Entonces el cliente recibe un código de estado 201 (creado)	
Y en la respuesta se recibe la siguiente información: alertId	
API DOC :	
POST	/alert
RESPONSE STATUS	201 Created
RESPONSE PARAMS	alertId
ALTERNATIVE RESPONSE STATUS	401 Unathorized (si el usuario no está logado o la petición no contiene el tokenId)
	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error

US - 003	Cómo profesional quiero incluir mi formación académica en mi perfil para competir en el mercado laboral
Criterios de aceptación	
Dado un profesional que accede a la API desde un cliente y se autentica	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros de la formación académica y el tokenId del usuario	
Entonces el cliente recibe un código de estado 201 (creado)	
Y en la respuesta se recibe la siguiente información: educationId	
API DOC :	
POST	/professional/{userId}/education
RESPONSE STATUS	201 Created
RESPONSE PARAMS	educationId
ALTERNATIVE RESPONSE STATUS	401 Unathorized (si el usuario no está logado o la petición no contiene el tokenId)
	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error

TABLA 4 – DETALLE HISTORIA 004

US - 004	Cómo profesional quiero incluir mi experiencia profesional en mi perfil para competir en el mercado laboral
Criterios de aceptación	
Dado un profesional que accede a la API desde un cliente y se autentica	

Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros de la experiencia profesional y el tokenId del usuario	
Entonces el cliente recibe un código de estado 201 (creado)	
Y en la respuesta se recibe la siguiente información: experienceId	
API DOC :	
POST	/professional/{userId}/experience
RESPONSE STATUS	201 Created
RESPONSE PARAMS	experienceId
ALTERNATIVE RESPONSE STATUS	401 Unathorized (si el usuario no está logado o la petición no contiene el tokenId)
	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error

TABLA 5 – DETALLE HISTORIA 005

US - 005	Cómo empresa quiero registrarme en la plataforma para dar de alta mis datos y estar visible en la plataforma
Criterios de aceptación	
Dado una empresa que accede al registro en una aplicación cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros email y password	
Entonces el cliente recibe un código de estado 201 (creado)	
Y en la respuesta se recibe la siguiente información: userToken, companyId	

API DOC :	
POST	/company
RESPONSE STATUS	201 Created
RESPONSE PARAMS	userToken, companyId
ALTERNATIVE RESPONSE STATUS	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error

TABLA 6 – DETALLE HISTORIA 006

US - 006	Cómo empresa quiero gestionar mis ofertas en la plataforma para captar los mejores profesionales.
Criterios de aceptación	
Dado una empresa que accede a la API en una aplicación cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros relativos a una oferta de trabajo	
Entonces el cliente recibe un código de estado 201 (creado)	
Y en la respuesta se recibe la siguiente información: jobOfferId	
API DOC :	
POST	/jobOffer
RESPONSE STATUS	201 Created
RESPONSE PARAMS	jobOfferId
ALTERNATIVE	401 Unathorized (si el usuario no está logado o la petición no contiene el tokenId)

RESPONSE STATUS	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error
PUT	/jobOffer/{jobOfferId}
RESPONSE STATUS	200 OK
RESPONSE PARAMS	jobOfferId
ALTERNATIVE RESPONSE STATUS	401 Unathorized (si el usuario no está logado o la petición no contiene el tokenId)
	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error
DELETE	/jobOffer/{jobOfferId}
RESPONSE STATUS	200 OK
RESPONSE PARAMS	
ALTERNATIVE RESPONSE STATUS	401 Unathorized (si el usuario no está logado o la petición no contiene el tokenId)
	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error

TABLA 7 – DETALLE HISTORIA 007

US - 007	Cómo usuario quiero realizar búsquedas de ofertas de empleo estableciendo franja horaria, idioma y/o ubicación para encontrar el trabajo deseado.
Criterios de aceptación	

Dado un usuario que accede a la API desde un cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros de filtrado	
Entonces el cliente recibe un código de estado 200 (OK)	
Y en la respuesta se recibe la siguiente información: jobOffers	
API DOC :	
GET	/jobOffers
RESPONSE STATUS	200 OK
RESPONSE PARAMS	jobOffers
ALTERNATIVE RESPONSE STATUS	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error

TABLA 8 – DETALLE HISTORIA 008

US - 008	Cómo usuario quiero realizar búsquedas de empresas estableciendo la zona horaria, idioma o ubicación.
Criterios de aceptación	
Dado un usuario que accede a la API desde un cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros de filtrado	
Entonces el cliente recibe un código de estado 200 (OK)	
Y en la respuesta se recibe la siguiente información: companies	
API DOC :	

GET	/companies
RESPONSE STATUS	200 OK
RESPONSE PARAMS	jobOffers
ALTERNATIVE RESPONSE STATUS	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error

TABLA 9 – DETALLE HISTORIA 009

US - 009	Cómo usuario quiero realizar búsquedas de profesionales estableciendo los idiomas, la franja horaria y/o la ubicación para captar profesionales o comprobar las tendencias del sector.
Criterios de aceptación	
Dado un usuario que accede a la API desde un cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros de filtrado	
Entonces el cliente recibe un código de estado 200 (OK)	
Y en la respuesta se recibe la siguiente información: professionals	
API DOC :	
GET	/professionals/{params}
RESPONSE STATUS	200 OK
RESPONSE PARAMS	professionals
ALTERNATIVE RESPONSE STATUS	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error

US - 010	Cómo usuario quiero consultar todas las ofertas de una categoría determinada para estar al día
Criterios de aceptación	
Dado un usuario que accede a la API desde un cliente	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene al menos un identificador de categoría o subcategoría	
Entonces el cliente recibe un código de estado 200 (OK)	
Y en la respuesta se recibe la siguiente información: jobOffers	
API DOC :	
GET	/jobOffers/{categoryId}
RESPONSE STATUS	200 OK
RESPONSE PARAMS	jobOffers
ALTERNATIVE RESPONSE STATUS	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error

TABLA 10 – DETALLE HISTORIA 011

US - 011	Cómo administrador quiero gestionar categorías, subcategorías y habilidades para tener una clasificación de ofertas consistentes.
Criterios de aceptación	
Dado un administrador que accede a la API en una aplicación cliente y se ha autenticado	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene AuthToken y userId	

Entonces el cliente recibe un código de estado 200 (OK)	
Y en la respuesta se recibe la siguiente información: categoryId	
API DOC :	
POST	/category
RESPONSE STATUS	201 Created
RESPONSE PARAMS	categoryId
ALTERNATIVE RESPONSE STATUS	401 Unathorized (si el usuario no está logado o la petición no contiene el AuthToken)
	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error
PUT	/category/{categoryId}
RESPONSE STATUS	200 OK
RESPONSE PARAMS	categoryId
ALTERNATIVE RESPONSE STATUS	401 Unathorized (si el usuario no está logado o la petición no contiene el tokenId)
	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error
DELETE	/category/{categoryId}
RESPONSE STATUS	200 OK
RESPONSE PARAMS	

ALTERNATIVE RESPONSE STATUS	401 Unathorized (si el usuario no está logado o la petición no contiene el tokenId)
	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error

TABLA 11 -DETALLE HISTORIA 012

US - 012	Cómo administrador quiero poder desactivar perfiles profesionales y de compañías si considero que son inadecuados (descripciones fuera de lugar, etc…) para que no aparezcan en las búsquedas ni listados
Criterios de aceptación	
Dado un administrador que accede a la API en una aplicación cliente y se ha autenticado	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros professionalId o companyId	
Entonces el cliente recibe un código de estado 200 (OK)	
Y en la respuesta se recibe la siguiente información: professionalId ó companyId	
API DOC :	
PUT	/professional/{professionalId}
RESPONSE STATUS	200 OK
RESPONSE PARAMS	professionalId
ALTERNATIVE RESPONSE STATUS	401 Unathorized (si el usuario no está logado o la petición no contiene el tokenId)
	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error

PUT	/company/{companyId}
RESPONSE STATUS	200 OK
RESPONSE PARAMS	companyId
ALTERNATIVE RESPONSE STATUS	401 Unathorized (si el usuario no está logado o la petición no contiene el tokenId)
	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error

TABLA 12 – DETALLE HISTORIA 013

US - 013	Cómo administrador quiero poder eliminar perfiles profesionales o de compañías por reincidir en malas prácticas para mantener la reputación de la plataforma
Criterios de aceptación	
Dado un administrador que accede a la API en una aplicación cliente y se ha autenticado	
Cuando la aplicación realiza la petición al end point del servicio REST adecuado con una request válida que contiene los parámetros professionalId o companyId y el AuthToken	
Entonces el cliente recibe un código de estado 200 (OK)	
Y en la respuesta no se recibe información	
API DOC :	
DELETE	/professional/{professionalId}
RESPONSE STATUS	200 OK
RESPONSE PARAMS	

ALTERNATIVE RESPONSE STATUS	401 Unathorized (si el usuario no está logado o la petición no contiene el tokenId)
	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error
DELETE	/company/{companyId}
RESPONSE STATUS	200 OK
RESPONSE PARAMS	
ALTERNATIVE RESPONSE STATUS	401 Unathorized (si el usuario no está logado o la petición no contiene el tokenId)
	400 Bad Request (si la petición recibida por la api es incorrecta)
	500 Internal Server Error

2.6. Sprints backlog

El desarrollo se afrontará por sprints según los backlogs que se destallan a continuación, cada sprint backlog se compone de una serie de tarjetas que representan las tareas a completar, cada una de ellas se relaciona con una tarea de puesta en marcha o bien con un caso de uso a implementar. El número de tarjetas que se han incluido en cada sprint varían según la complejidad estimada aproximada de cada una, de forma que la duración de los sprints sea homogénea excepto para el primer sprint que tendrá una duración superior.

El primer sprint tendrá una duración de dos semanas, dado que incluye la instalación de la base de datos, inicialización de ésta y puesta en marcha de todo el entorno tecnológico, del cual no se tiene experiencia previa en algunos componentes. Se implementarán los primeros tests unitarios que servirán como modelo para los sprints posteriores.

TABLA 13 - DETALLE SPRINT BACKLOG 1

Sprint Backlog 1	
T - 001	Inicializar la base de datos
T - 002	Precargar un juego de datos para poder desarrollar las implementaciones de forma independiente.
T - 003	Implementar los tests unitarios para US - 001 (POST)
T - 004	Desarrollar las funcionalidades necesarias para cubrir los tests unitarios para US - 001 (POST)
T - 005	Verificar criterios de aceptación y tests unitarios US - 001
T - 006	Implementar los tests unitarios para US - 007 (GET)
T - 007	Desarrollar las funcionalidades necesarias para cubrir los tests unitarios para US - 007 (GET)
T - 008	Verificar criterios de aceptación y tests unitarios US - 007
T - 009	Implementar los tests unitarios para US - 005 (POST)
T - 010	Desarrollar las funcionalidades necesarias para cubrir los tests unitarios para US - 005 (POST)
T - 011	Verificar criterios de aceptación y tests unitarios US - 005

En el segundo sprint se abordará el desarrollo de tareas más puramente de negocio y tendrá una duración de una semana, al igual que todos los sprints restantes hasta finalizar el desarrollo. No se espera que en este sprint sea necesario abordar tareas inconclusas del sprint anterior.

TABLA 14 - DETALLE SPRINT BACKLOG 2

Sprint Backlog 2	
T - 001	Implementar los tests unitarios para US - 003 (POST)
T - 002	Desarrollar las funcionalidades necesarias para cubrir los tests unitarios para US - 003 (POST)

T - 003	Verificar criterios de aceptación y tests unitarios US - 003
T - 004	Implementar los tests unitarios para US - 004 (POST)
T - 005	Desarrollar las funcionalidades necesarias para cubrir los tests unitarios para US - 004 (POST)
T - 006	Verificar criterios de aceptación y tests unitarios US - 004
T - 007	Implementar los tests unitarios para US - 002 (POST)
T - 008	Desarrollar las funcionalidades necesarias para cubrir los tests unitarios para US - 002 (POST)
T - 009	Verificar criterios de aceptación y tests unitarios US - 002

En el tercer sprint se continuará con la línea de trabajo del segundo sprint y se incluirán en el mismo si fuera necesario las tareas no finalizadas del sprint anterior.

TABLA 15- DETALLE SPRINT BACKLOG 3

Sprint Backlog 3	
T - 001	Implementar los tests unitarios para US - 006 (POST, PUT, DELETE)
T - 002	Desarrollar las funcionalidades necesarias para cubrir los tests unitarios para US - 006 (POST, PUT, DELETE)
T - 003	Verificar criterios de aceptación y tests unitarios US - 006
T - 004	Implementar los tests unitarios para US - 008 (POST)
T - 005	Desarrollar las funcionalidades necesarias para cubrir los tests unitarios para US - 008 (POST)
T - 006	Verificar criterios de aceptación y tests unitarios US - 008
T - 007	Implementar los tests unitarios para US - 009 (POST)
T - 008	Desarrollar las funcionalidades necesarias para cubrir los tests unitarios para US - 009 (POST)
T - 009	Verificar criterios de aceptación y tests unitarios US - 009

En el cuarto y último sprint se continuará con la línea de trabajo y desarrollo de los sprints anteriores y nuevamente se rescatarán las tareas inconclusas del sprint anterior.

TABLA 16 - DETALLE SPRINT BACKLOG 4

Sprint Backlog 4	
T - 001	Implementar los tests unitarios para US - 010 (GET)
T - 002	Desarrollar las funcionalidades necesarias para cubrir los tests unitarios para US - 010 (GET)
T - 003	Verificar criterios de aceptación y tests unitarios US - 010
T - 004	Implementar los tests unitarios para US - 011 (POST, PUT, DELETE)
T - 005	Desarrollar las funcionalidades necesarias para cubrir los tests unitarios para US - 011 (POST, PUT, DELETE)
T - 006	Verificar criterios de aceptación y tests unitarios US - 011
T - 007	Implementar los tests unitarios para US - 012 (PUT)
T - 008	Desarrollar las funcionalidades necesarias para cubrir los tests unitarios para US - 012 (PUT)
T - 009	Verificar criterios de aceptación y tests unitarios US - 012
T - 010	Implementar los tests unitarios para US - 013 (DELETE)
T - 011	Desarrollar las funcionalidades necesarias para cubrir los tests unitarios para US - 013 (DELETE)
T - 012	Verificar criterios de aceptación y tests unitarios US - 013 (DELETE)

La duración total de los cuatro sprints conlleva 5 semanas, con lo que aún quedarían algunos días de margen para asumir posibles retrasos y preparar la entrega de esta fase.

3. Diseño

3.1. Diagrama de desde el punto de vista de la información

En base a los requisitos previstos, podemos elaborar el diagrama desde el punto de vista de la información con los datos con lo que tendrá que trabajar nuestro sistema.

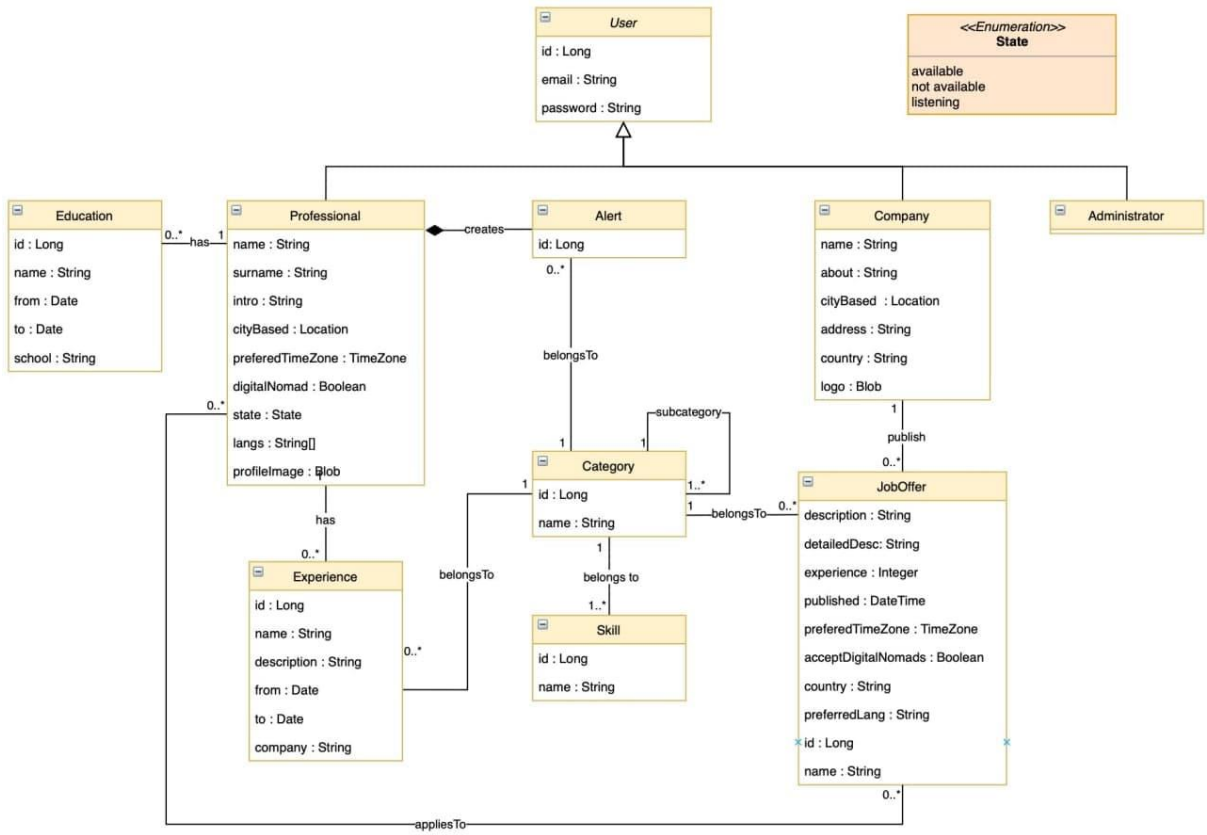


ILUSTRACIÓN 8 – DIAGRAMA PUNTO DE VISTA DE LA INFORMACIÓN

Detalle de clases

➔ **User.** El usuario es una generalización de los usuarios que se registrarán en el sistema y de los cuales tendremos email y contraseña para hacer login.

➔ **Professional.** De los profesionales, guardaremos nombre y apellido, una breve introducción a su perfil, la ciudad en la que habitan habitualmente o de origen, la zona horaria en la que prefieren o desean trabajar, indicarán si son nómadas

digitales, su estado actual (disponible, no disponible, escucha ofertas) y los idiomas en los que pueden trabajar.

- **Administrator.** Del administrador sólo necesitamos email y password que le autentican en el sistema.

- **Company.** De las compañías, guardaremos el nombre, la ciudad y país principal de la misma y una dirección.
- **Education.** El profesional puede tener de ninguno a varios registros de educación relevantes para su perfil, de cada uno guardaremos el nombre, la fecha de inicio, la fecha de fin y la escuela.

- **Experience.** Así mismo, el profesional puede tener ninguna a varias experiencias profesionales previas, de las cuales guardaremos el nombre del puesto, la descripción del stack de la experiencia, la fecha de inicio, la fecha de fin y la compañía.

- **JobOffer.** Las empresas publicarán ofertas de empleo, de las cuales guardaremos, el nombre del puesto ofertado, la descripción general, la experiencia previa mínima en años, la fecha de publicación, la franja horaria en la que deberán estar los profesionales, el país e idioma preferido del puesto de trabajo, así como guardar si se aceptan nómadas digitales para ese puesto ofertado.

- **Category.** Los puestos de trabajos y las experiencias se recogen dentro de una categoría, que a su vez será subcategoría, de cómo mínimo una superior. De las categorías guardaremos el nombre.

- **Skills.** Cada categoría o subcategoría, conlleva una serie de habilidades o conocimientos específicos de la categoría, que conformarán el stack de conocimientos necesarios o desarrollados. De las habilidades guardaremos el nombre.

- **Alert.** Los profesionales pueden crear alertas para recibir avisos cada vez que se publique una oferta en una determinada categoría y/o para una serie de habilidades.

3.2. Diagrama entidad-relación orientado a NoSQL

A continuación, estudiamos el diagrama de entidades relacionales que persistirán en nuestra base de datos, de forma orientada al sistema NoSQL elegido

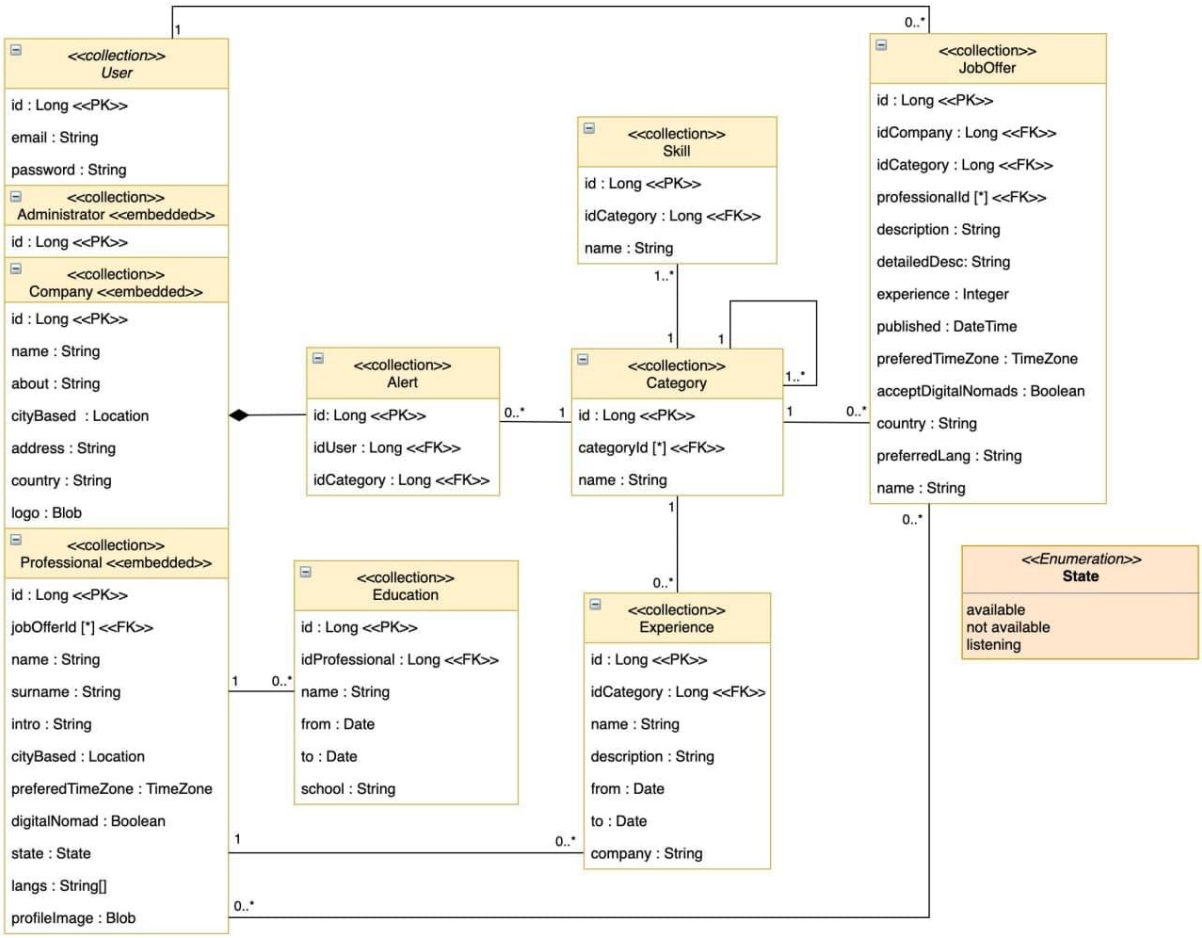


ILUSTRACIÓN 9 – DIAGRAMA DE CLASES NOSQL

Cómo podemos observar en el diagrama anterior, las entidades <<embedded>> son aquellas que tienen una relación de herencia con User, tal y como se modeló en el punto de vista de la información.

3.3. Diagrama de arquitectura

El sistema se desarrollará sobre un modelo de tres capas, en el que nos encontramos: EndPoints, Business e Integración

3.3.1. Punto de vista de la computación

A continuación, en el punto de vista de la computación, podemos ver el diagrama en el que se incluyen los componentes principales del sistema agrupados por funcionalidad y como se relacionan entre ellos, sin entrar en detalle y con poca granularidad. El objetivo de esta primera aproximación es definir la interacción entre los distintos componentes funcionales distribuidos en las tres capas que conformarán la arquitectura del sistema.

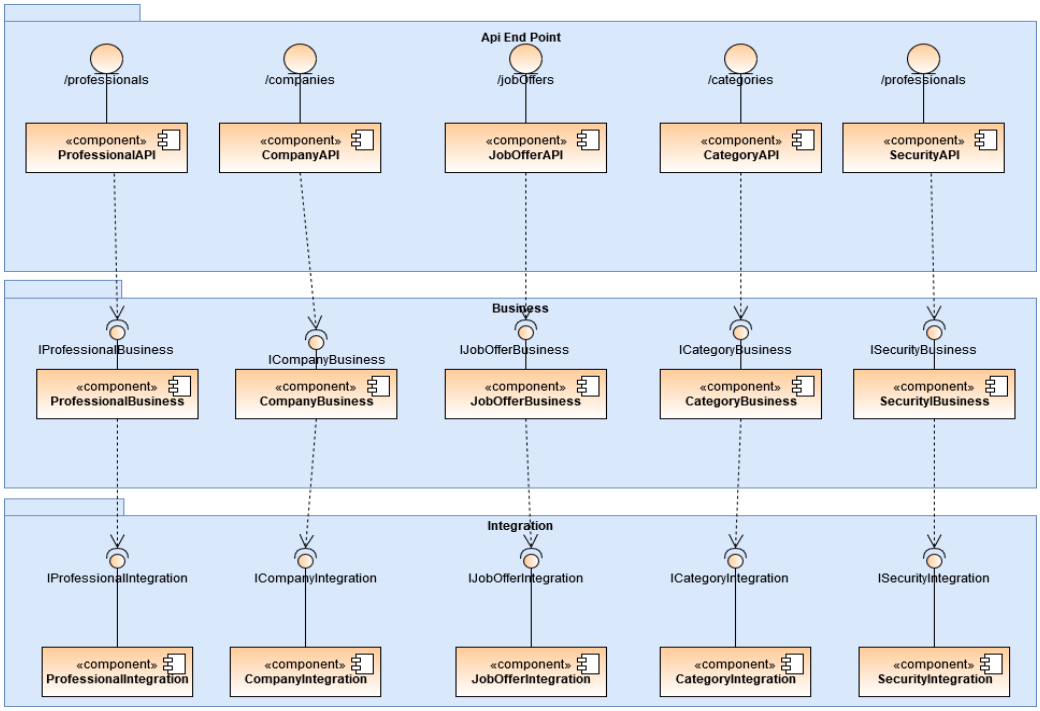


ILUSTRACIÓN 10 – PUNTO DE VISTA DE LA COMPUTACIÓN

El detalle de los métodos que expone cada componente entre capas lo obviamos en este punto por motivos de legibilidad, pero lo vemos a continuación en diagramas más detallados de cada capa.

3.3.2. Diagrama End Point Rest Controllers

ILUSTRACIÓN 11 – DIAGRAMA CAPA END POINT

3.3.3. Diagrama Lógica de Negocio

ILUSTRACIÓN 12 – DIAGRAMA CAPA NEGOCIO

3.3.4. Diagrama Integración (Persistencia)

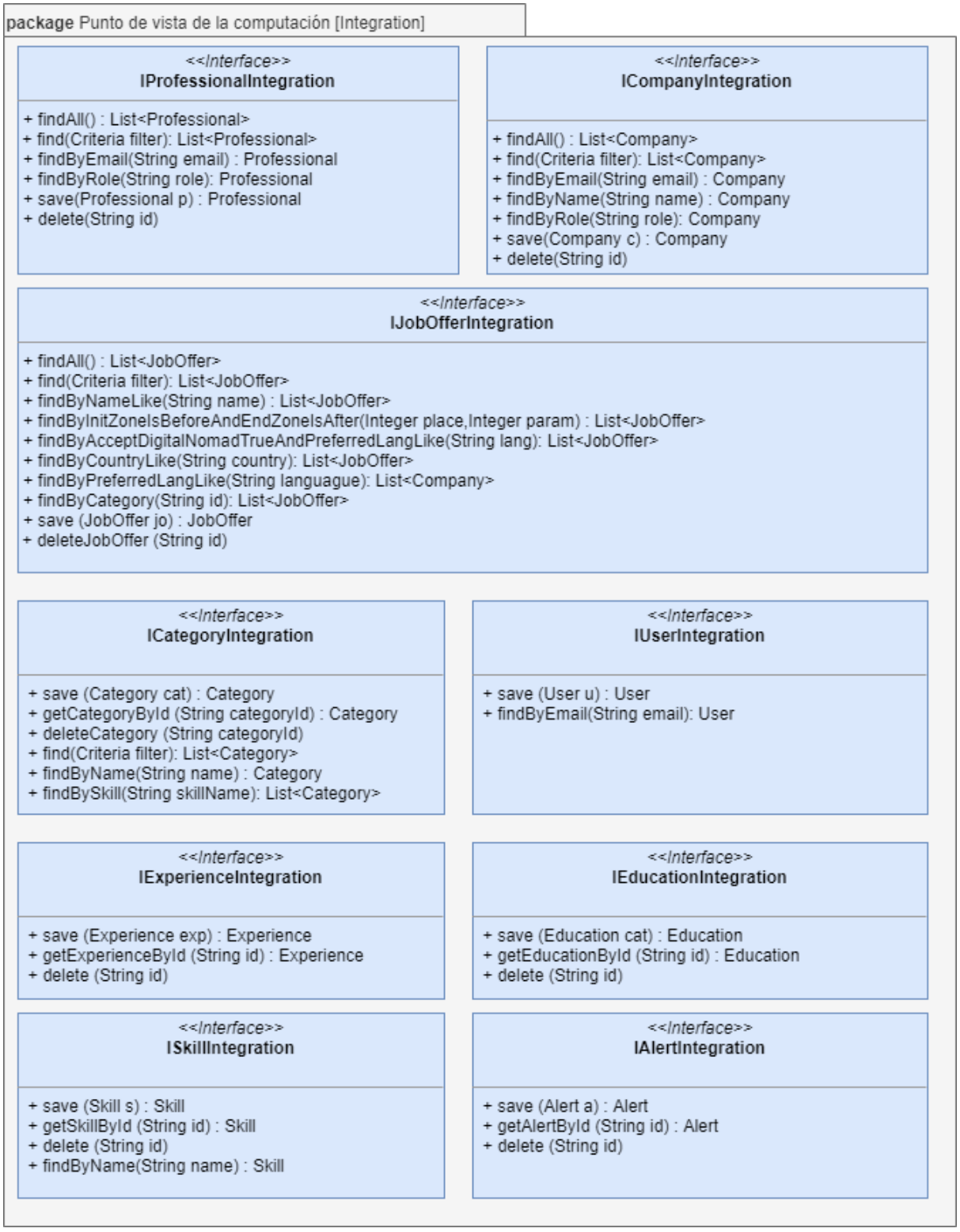


ILUSTRACIÓN 13 – DIAGRAMA CAPA INTEGRACIÓN

3.3.5. Refinamiento a partir del punto de vista de la computación

3.3.5.1. Capa de End Points

Desarrollaremos un sistema basado en tres capas como hemos visto en el diagrama de la computación, este sistema seguirá la arquitectura MVC (Modelo – Vista – Controlador) con la excepción de que en nuestra aplicación, en lugar de vistas tendremos los end points que publican la información que ofrece nuestra api REST.

A continuación, hacemos una aproximación por componentes a la capa de end points (presentación).

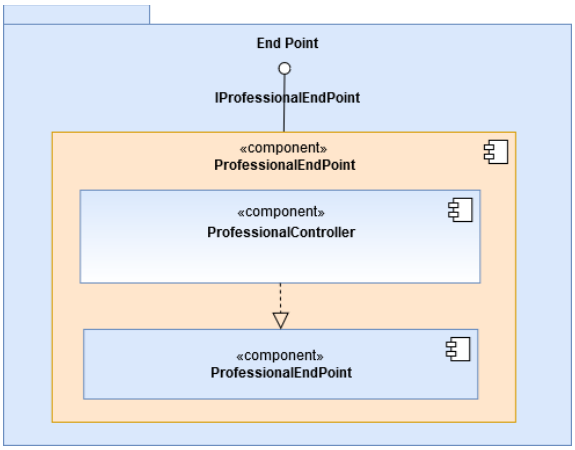


ILUSTRACIÓN 14 – PROFESSIONAL ENDPOINT

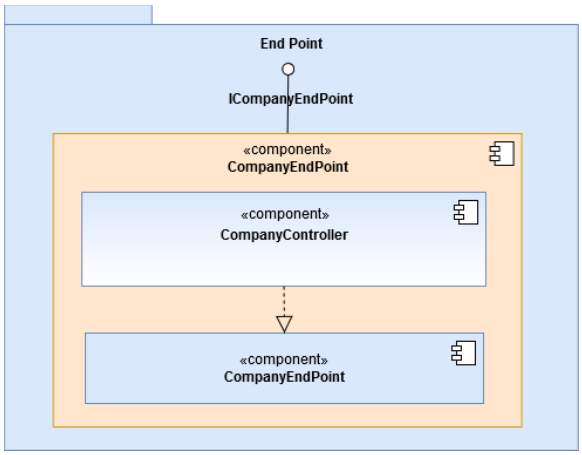


ILUSTRACIÓN 15 – COMPANY ENDPOINT

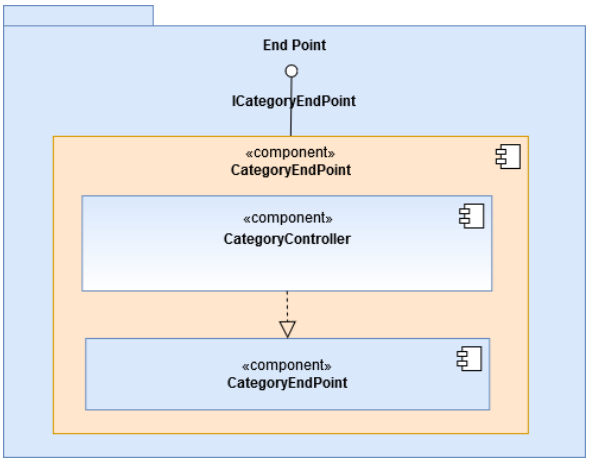


ILUSTRACIÓN 16 – CATEGORY ENDPOINT

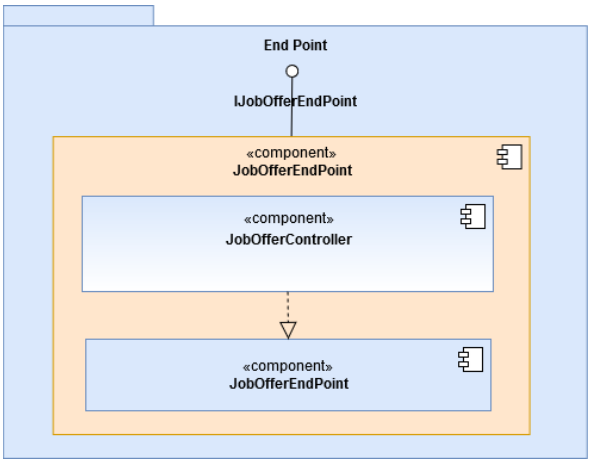


ILUSTRACIÓN 17 -JOBOFFER ENDPOINT

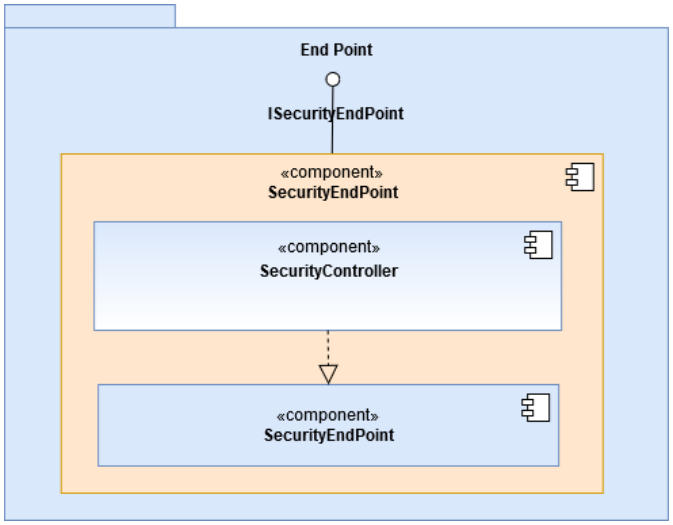


ILUSTRACIÓN 18 – SECURITY ENDPOINT

Partiendo del refinamiento inicial visto anteriormente, decidimos aplicar las siguientes pautas de diseño:

- Tendremos un único controlador por cada componente (MainController)
- Cada método del controller corresponde con un end point.

Luego tras aplicarlas, obtenemos un nuevo refinamiento:

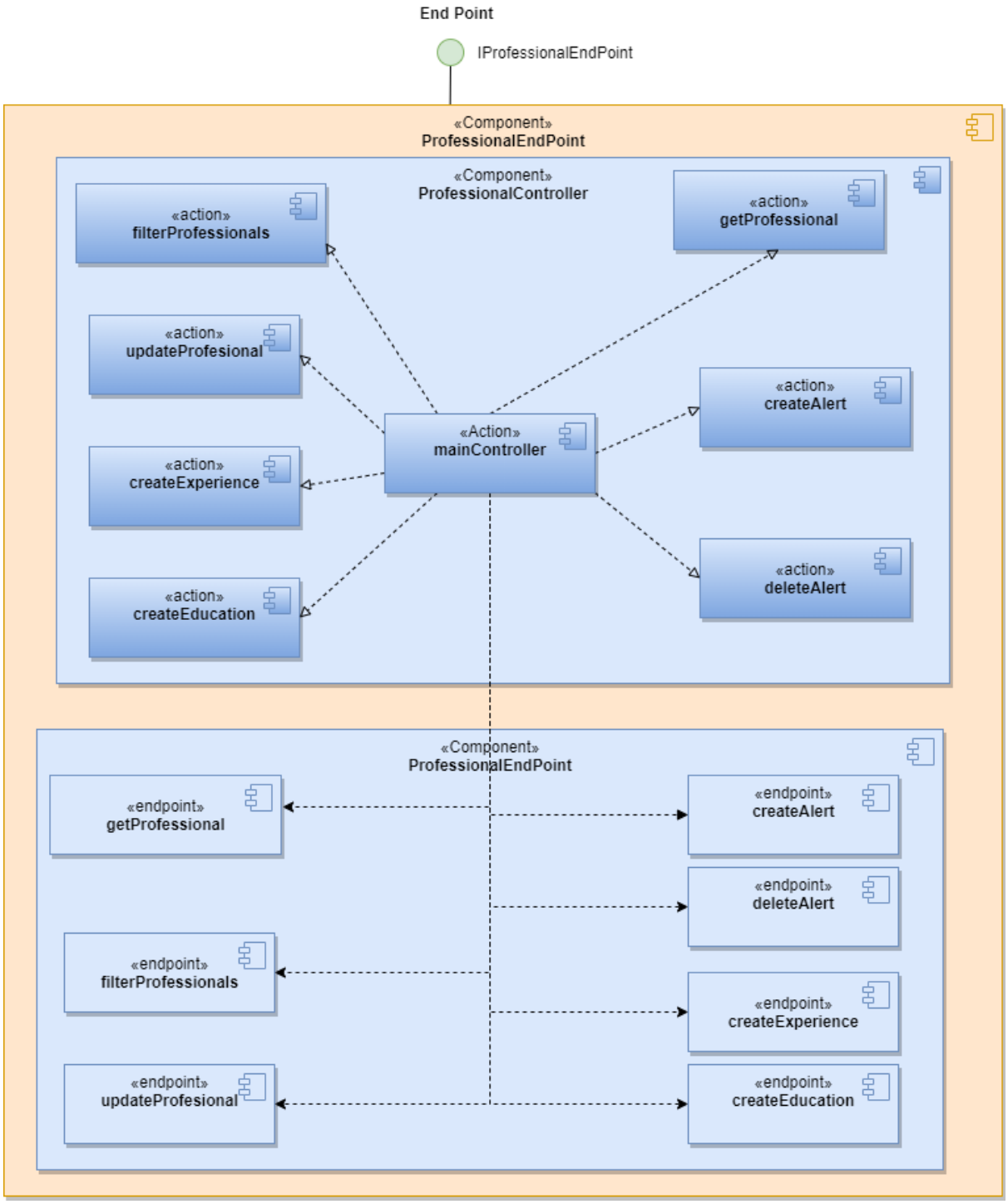


ILUSTRACIÓN 19 – PROFESSIONAL ENDPOINT CONTROLLER

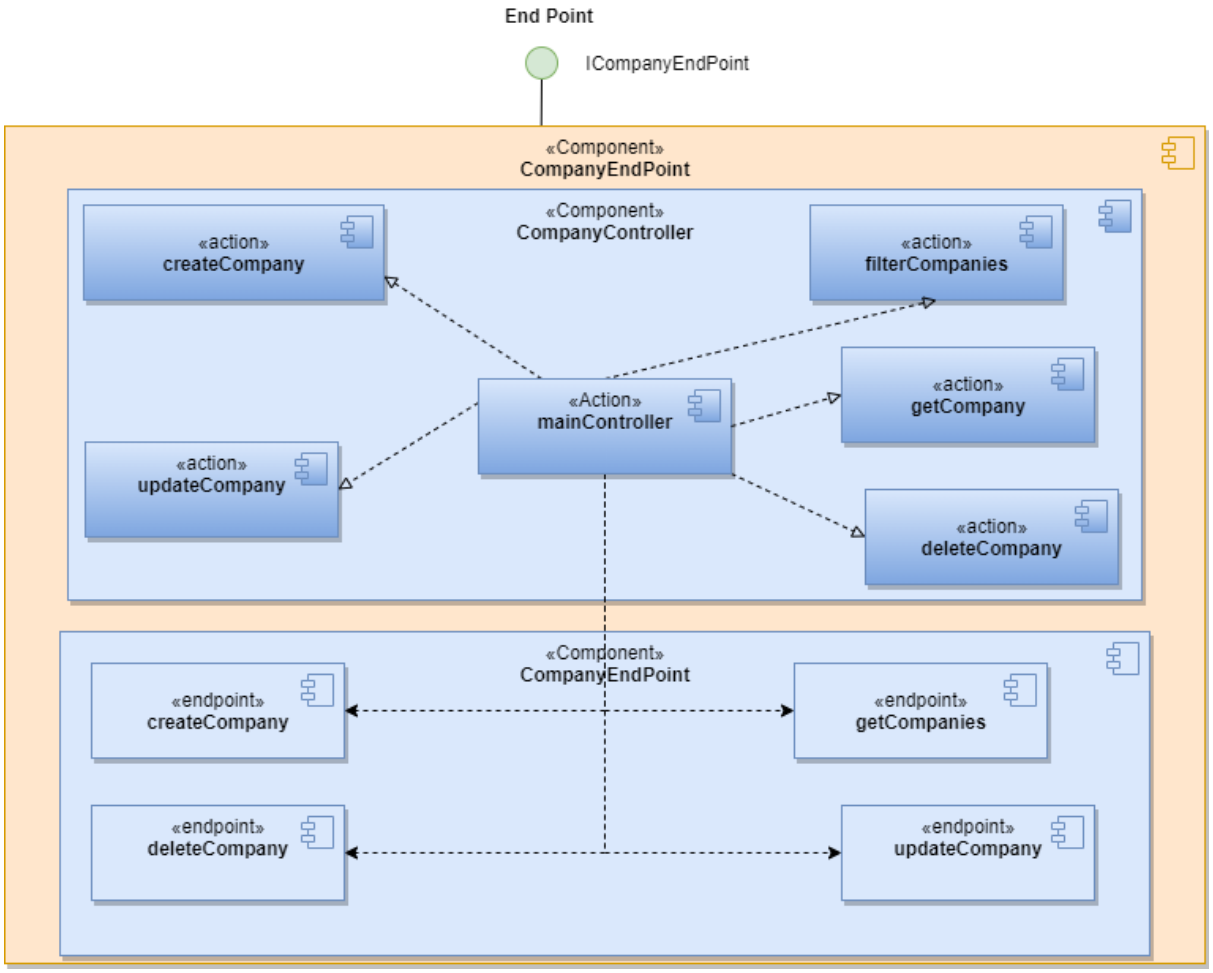


ILUSTRACIÓN 20 – COMPANY ENDPOINT CONTROLLER

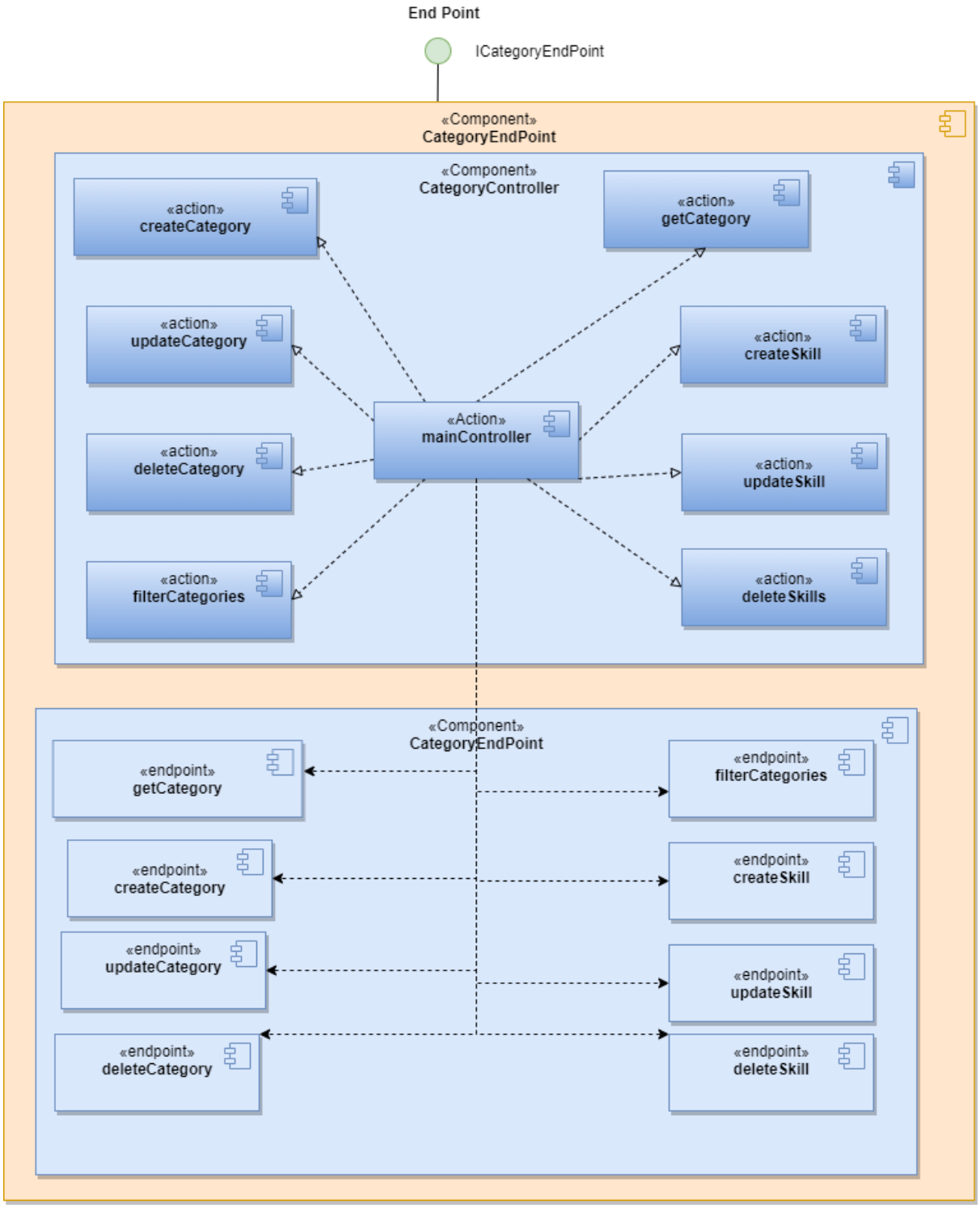


ILUSTRACIÓN 21 – CATEGORY ENDPOINT CONTROLLER

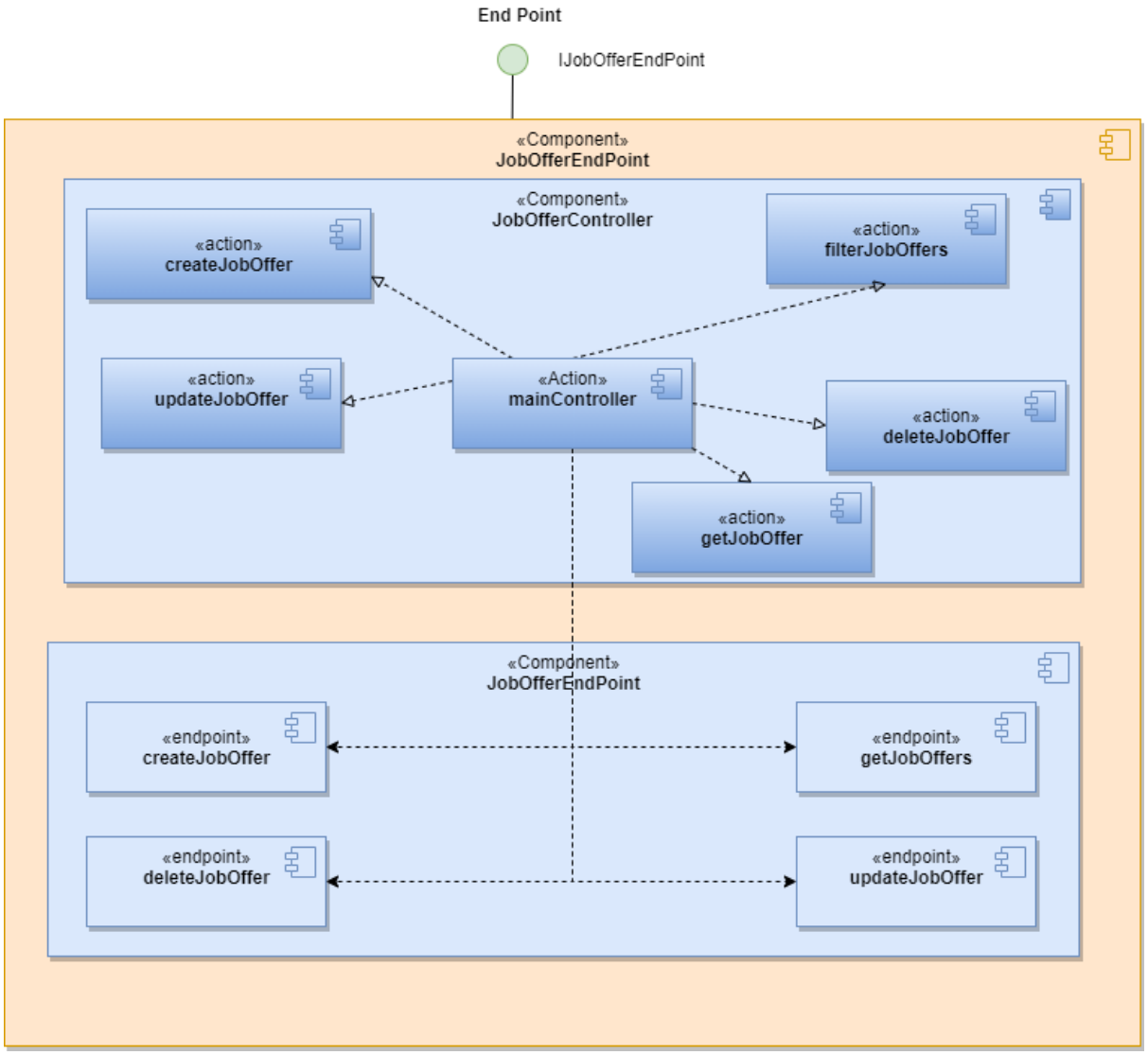


ILUSTRACIÓN 22 – JOBOFFER ENDPOINT CONTROLLER

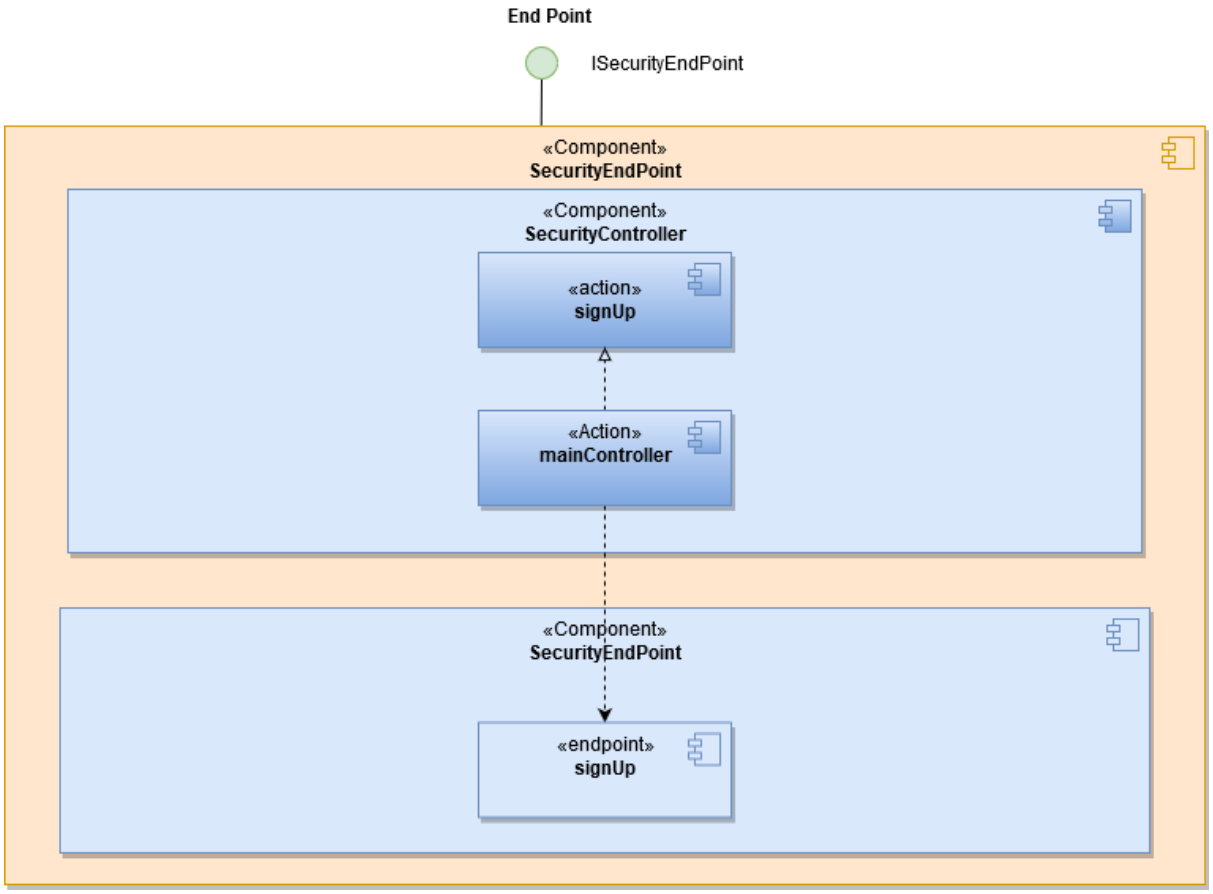


ILUSTRACIÓN 23 – AUTH ENDPOINT CONTROLLER

3.3.5.2. Capa de negocio

Dado que la capa de negocio seguirá los estándares habituales, no necesitamos refinar más a este nivel, quedando los componentes como sigue:

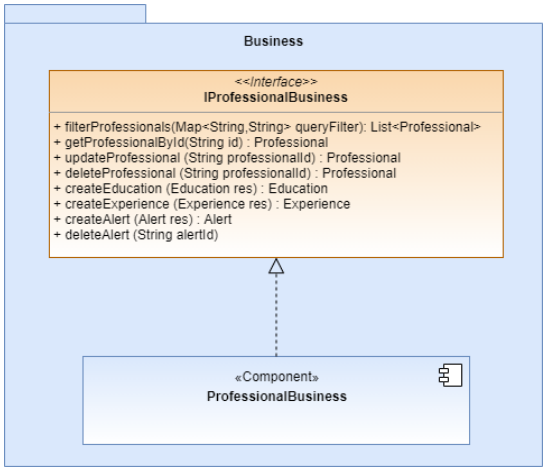


ILUSTRACIÓN 24 – PROFESSIONAL BUSINESS

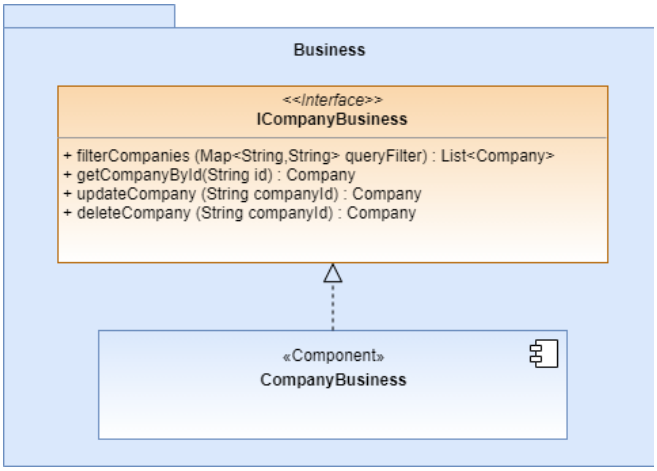


ILUSTRACIÓN 25 – COMPANY BUSINESS

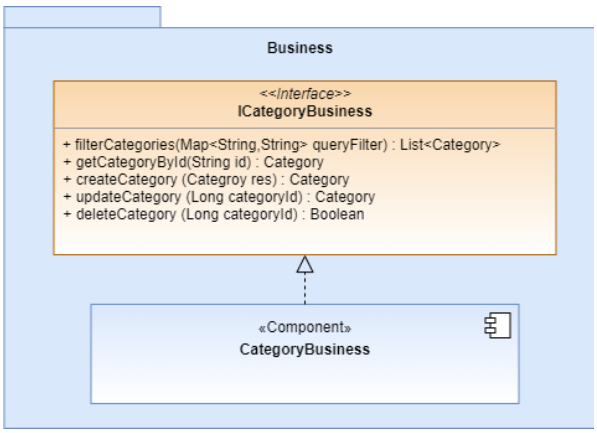


ILUSTRACIÓN 26 – CATEGORY BUSINESS

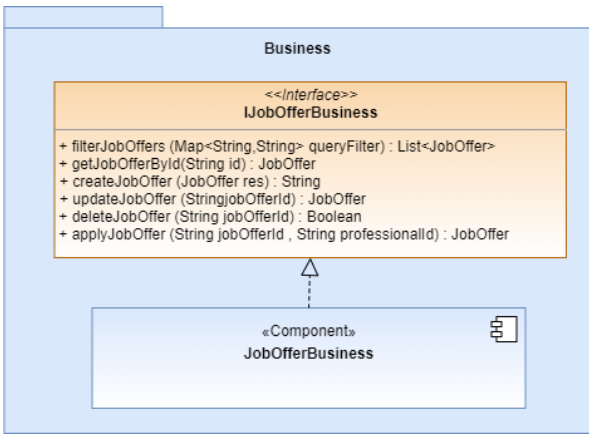


ILUSTRACIÓN 27 – JOBOFFER BUSINESS

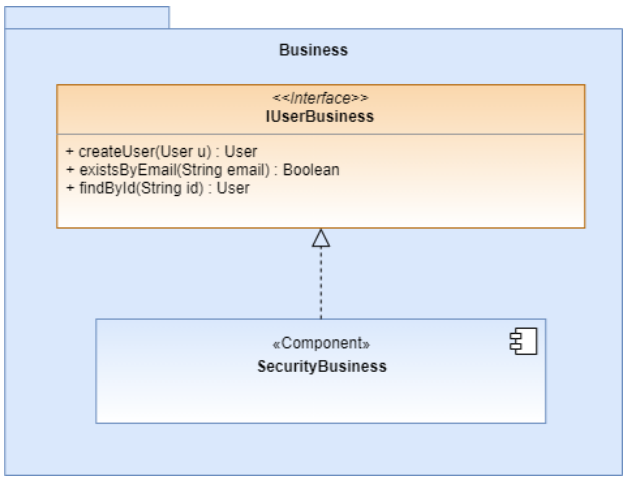


ILUSTRACIÓN 28 – SECURITY BUSINESS

3.3.5.3. Capa de integración

En la capa de integración trabajaremos con spring-data, que nos ofrece una integración transparente con mongoDB, luego tampoco será necesario aplicar más refinamientos a este nivel, quedando los componentes como sigue:

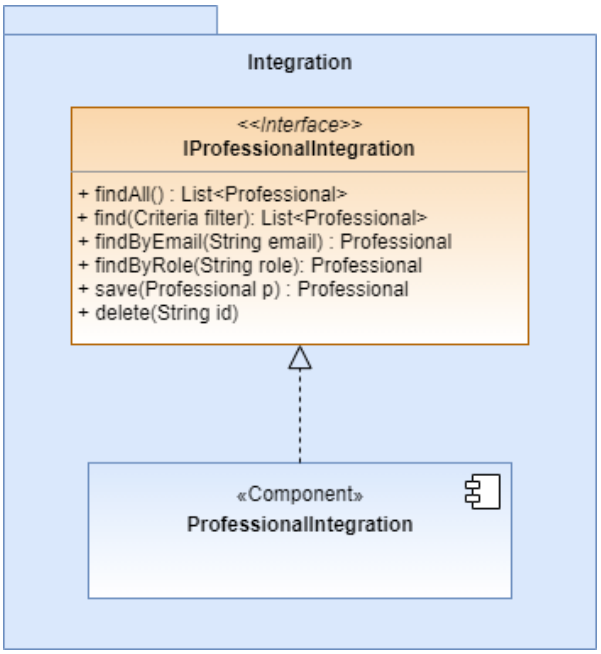


ILUSTRACIÓN 29 – PROFESSIONAL INTEGRATION

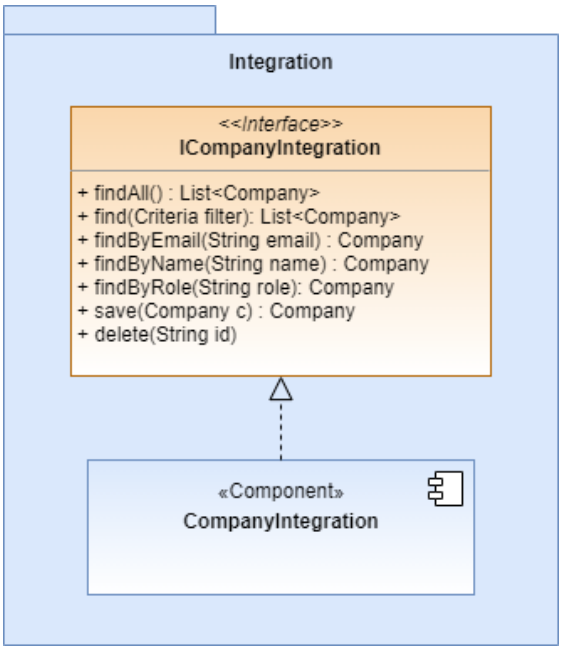


ILUSTRACIÓN 30 – COMPANY INTEGRATION

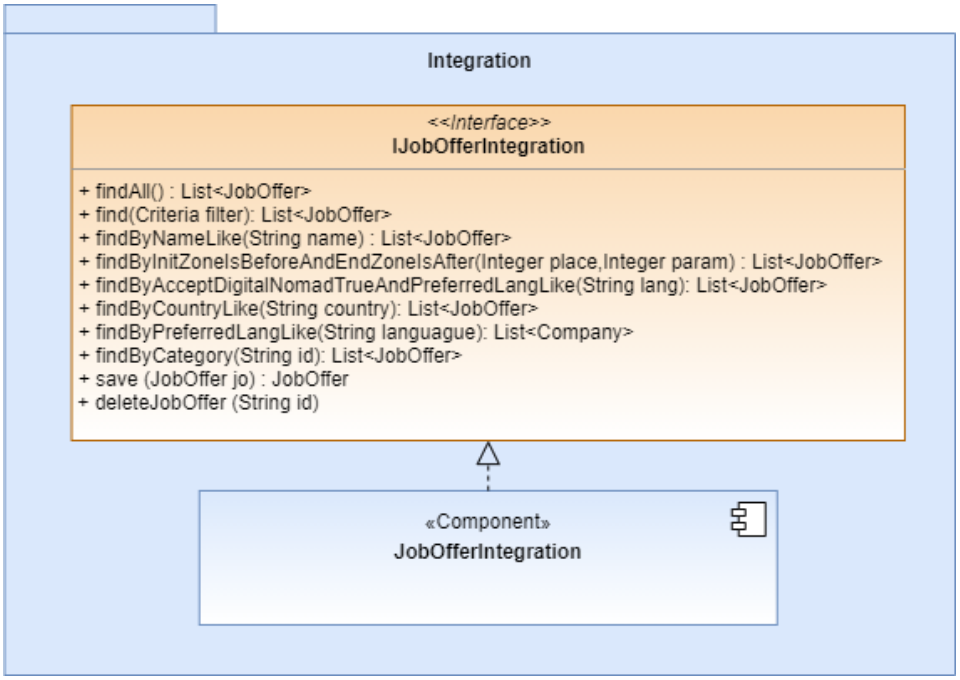


ILUSTRACIÓN 31 – JOBOFFER INTEGRATION

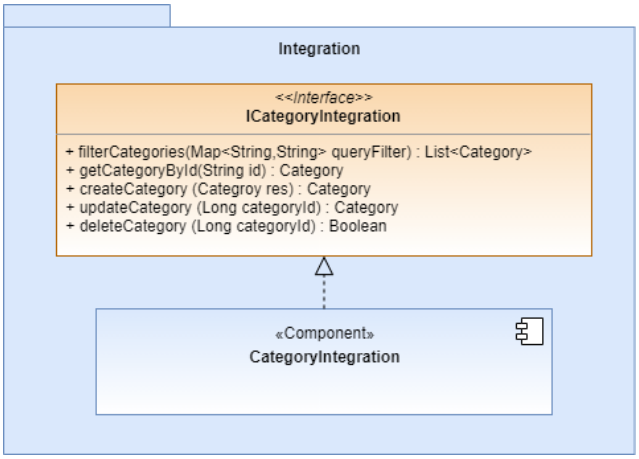


ILUSTRACIÓN 32 – CATEGORY INTEGRATION

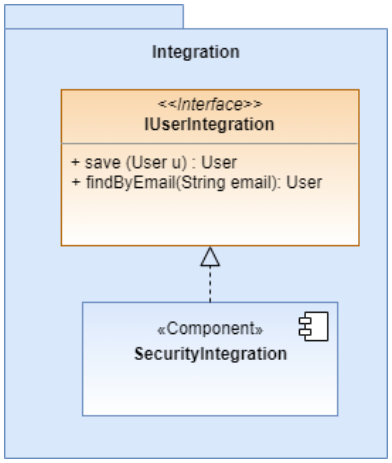


ILUSTRACIÓN 33 – USER INTEGRATION

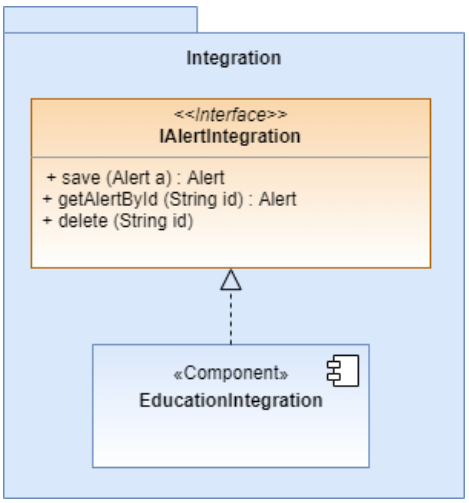


ILUSTRACIÓN 34 – ALERT INTEGRATION

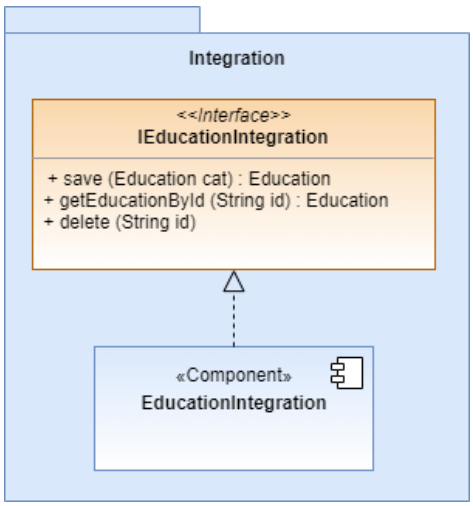


ILUSTRACIÓN 35 – EDUCATION INTEGRATION

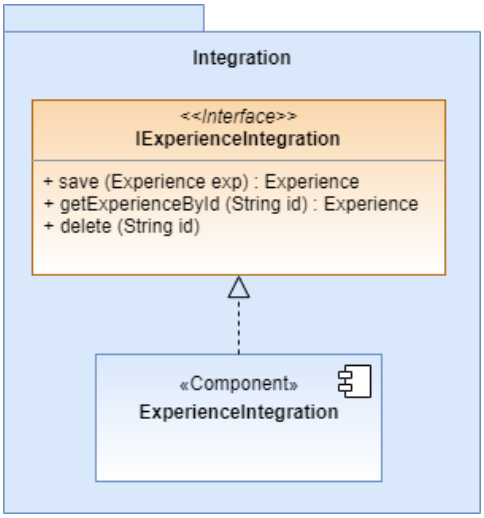


ILUSTRACIÓN 36 – EXPERIENCE INTEGRATION

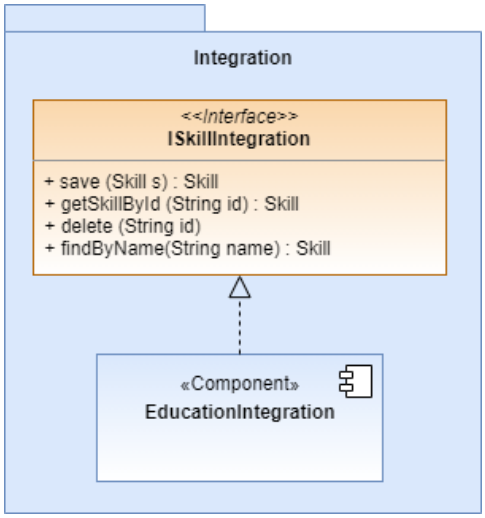


ILUSTRACIÓN 37 – SKILL INTEGRATION

3.3.6. Diagrama de refinamiento por componentes

3.3.6.1. End Point

Aplicando los refinamientos adecuados al nivel de los end points y aplicando el perfil REST a los diagramas, obtenemos los siguientes componentes, donde desgranamos con más detalles los métodos de las interfaces que publicarán cada recurso y el método REST al que corresponde cada uno.

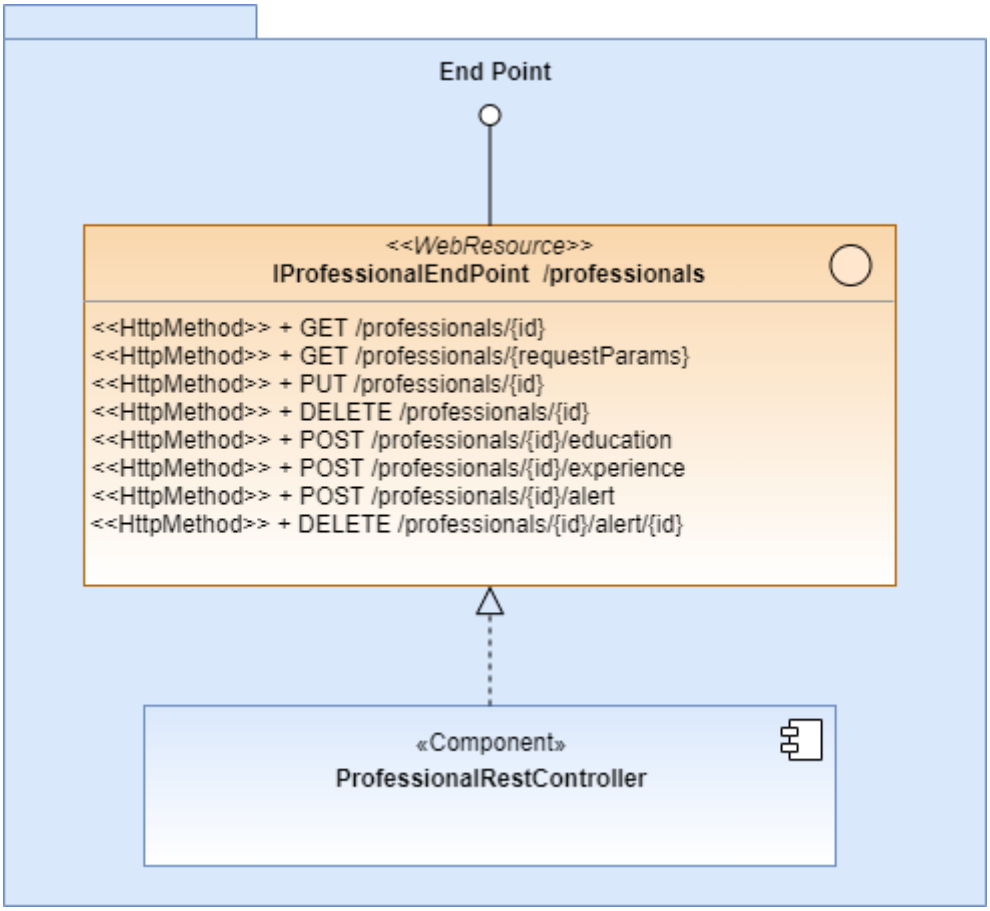


ILUSTRACIÓN 38 – PROFESSIONAL ENDPOINT DETALLE

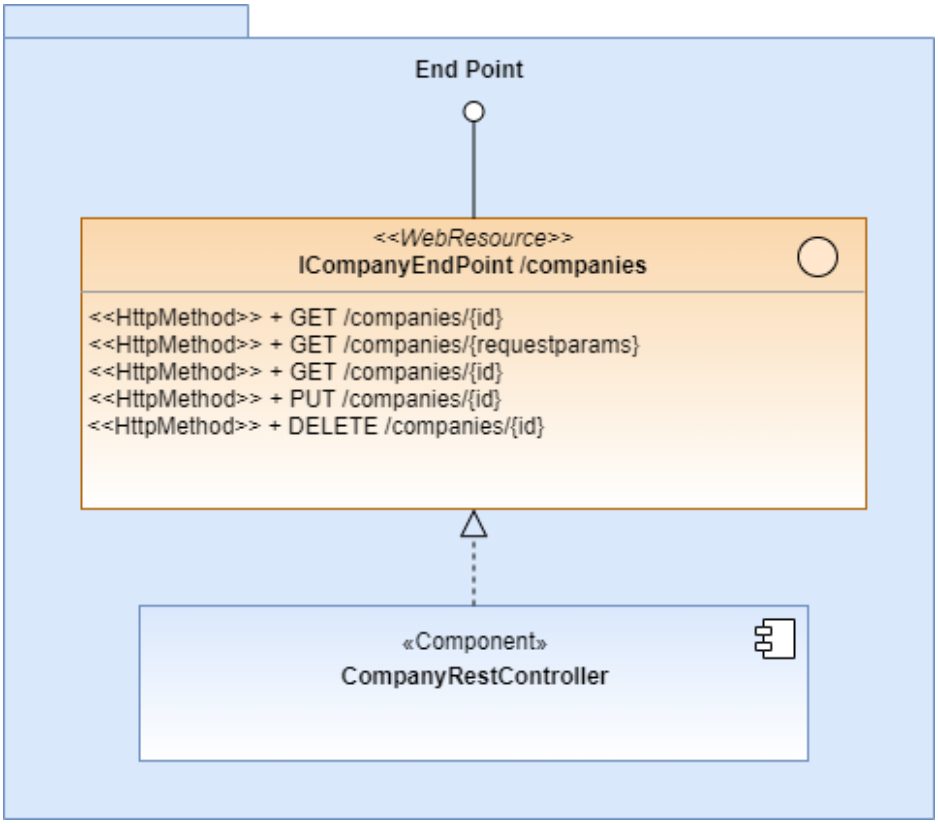


ILUSTRACIÓN 39 – COMPANY ENDPOINT DETALLE

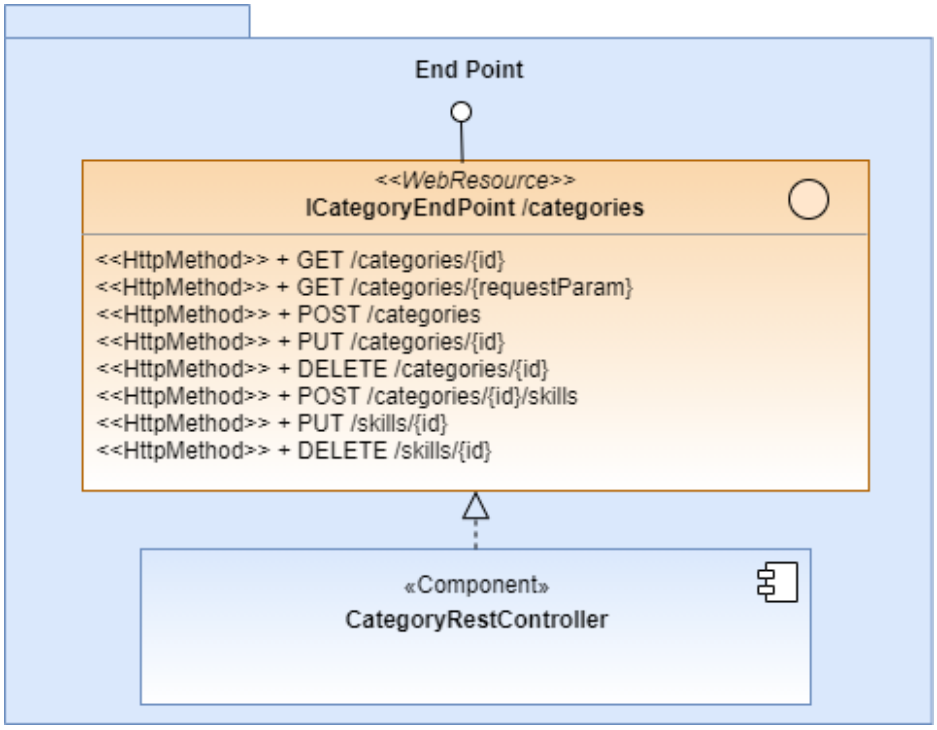


ILUSTRACIÓN 40 – CATEGORY ENDPOINT DETALLE

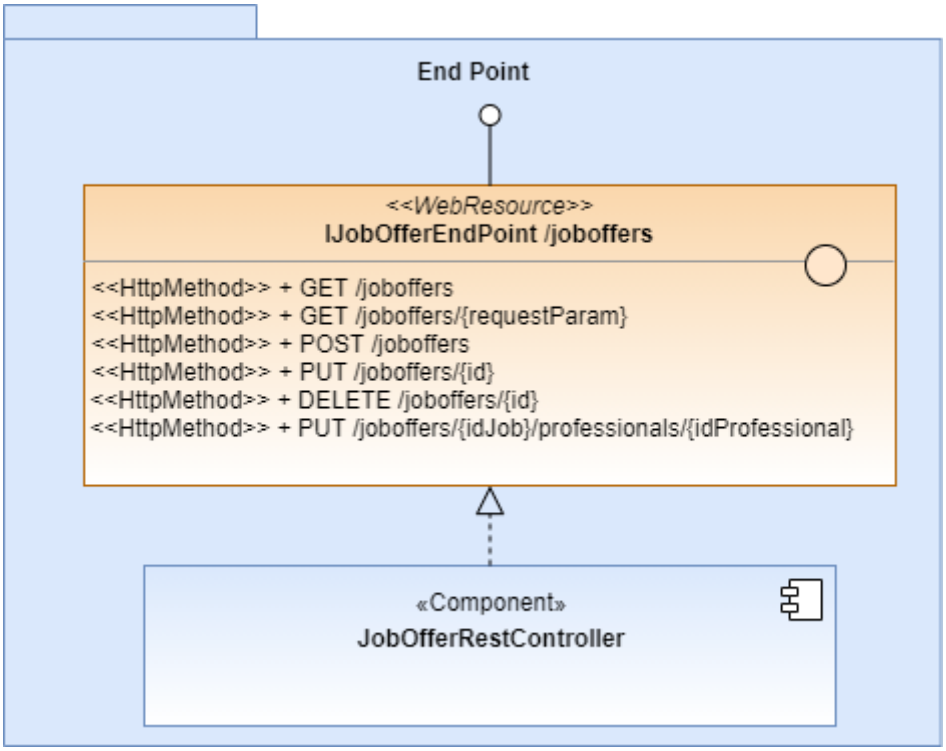


ILUSTRACIÓN 41 – JOBOFFER ENDPOINT DETALLE

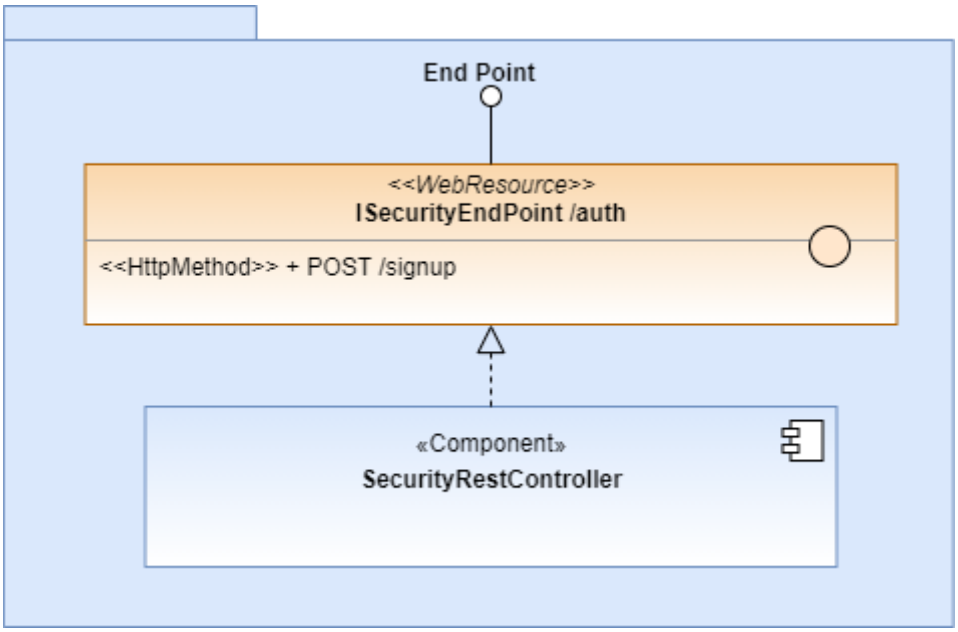


ILUSTRACIÓN 42 – SECURITY ENDPOINT DETALLE

Y a continuación, desgranamos más detalladamente las partes que intervienen para el correcto funcionamiento de un end point determinado orientado a servicio rest.

Debemos tener en cuenta, en los refinamientos aquí desarrollados, que todos seguirán el esquema indicado a continuación, pero que se ha omitido de los refinamientos para evitar diagramas excesivamente complejos donde se perdiera claridad. Toda la seguridad vendrá gestionada por OAuth 2.0 y en el diagrama a continuación se muestra a grosso modo su funcionamiento.

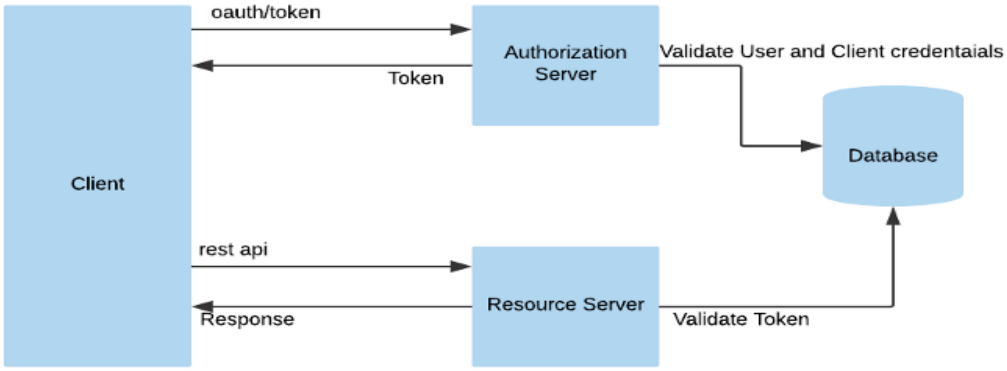


ILUSTRACIÓN 43 – FLUJO AUTENTICACIÓN OAUTH 2.0

Consideramos también a modo de ejemplo, desgranar en un diagrama detallado el flujo de una petición REST, en la que consultamos ofertas de empleos, creamos una oferta de empleo y borramos una oferta de empleo, secuencia omitida de los diagramas de refinamiento por los motivos expuestos en el punto anterior.

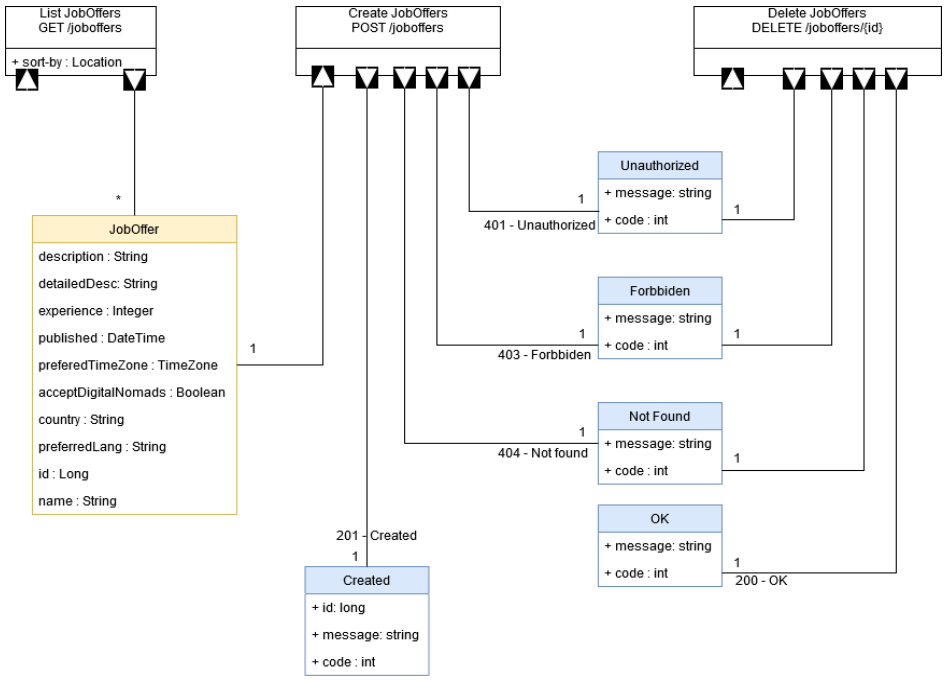


ILUSTRACIÓN 44 – ESQUEMA FLUJO PETICIÓN REST

En base a los diagramas estudiados hasta ahora y teniendo en cuenta la tecnología a aplicar, podemos refinar un poco más y desgranar con más detalle cada rest controller.

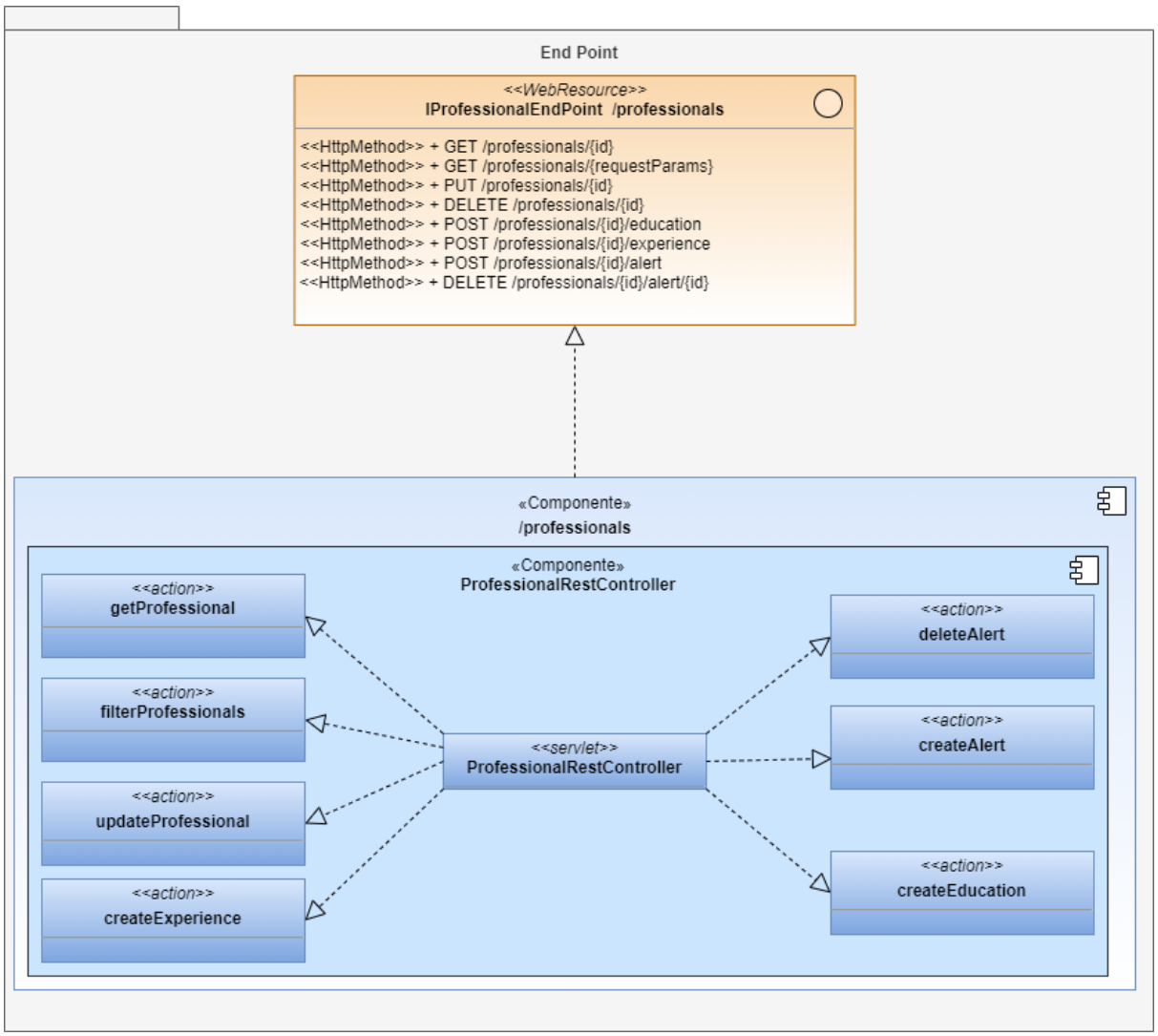


ILUSTRACIÓN 45 – PROFESSIONAL ENDPOINT REST CONTROLLER DETALLE

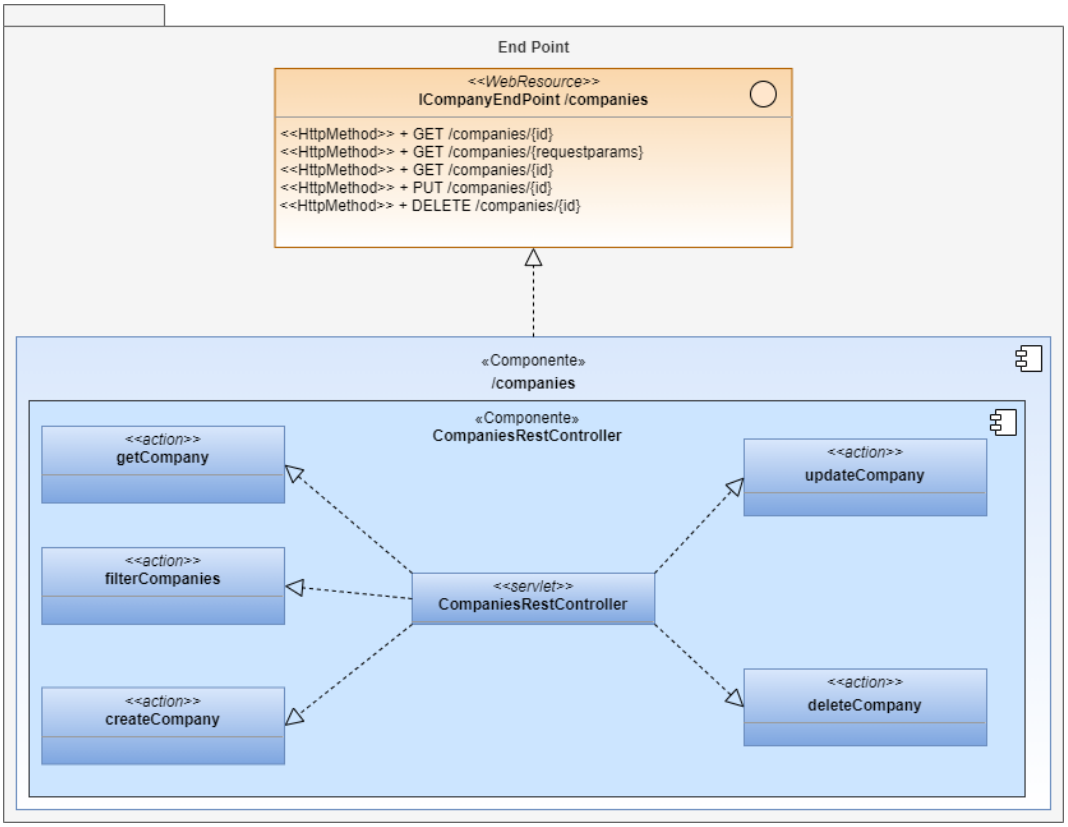


ILUSTRACIÓN 46 – COMPANY ENDPOINT REST CONTROLLER DETALLE

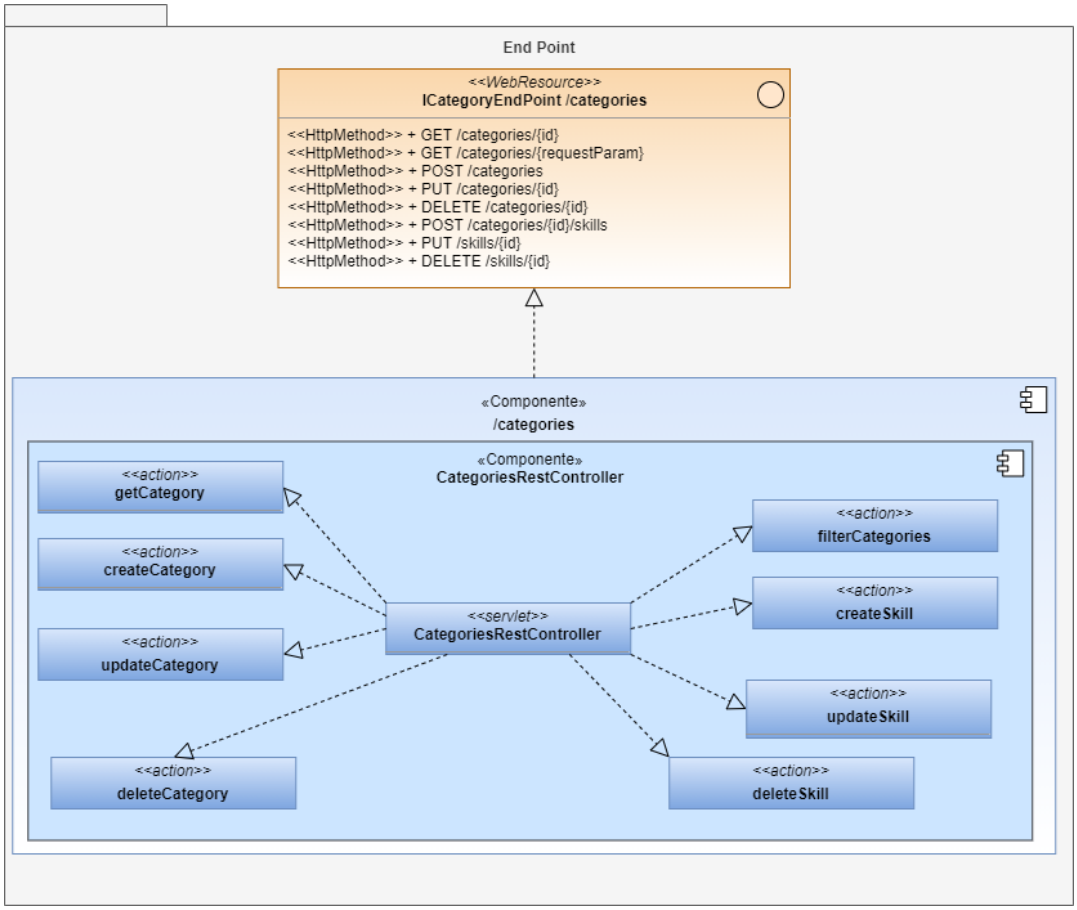


ILUSTRACIÓN 47 – CATEGORY ENDPOINT REST CONTROLLER DETALLE

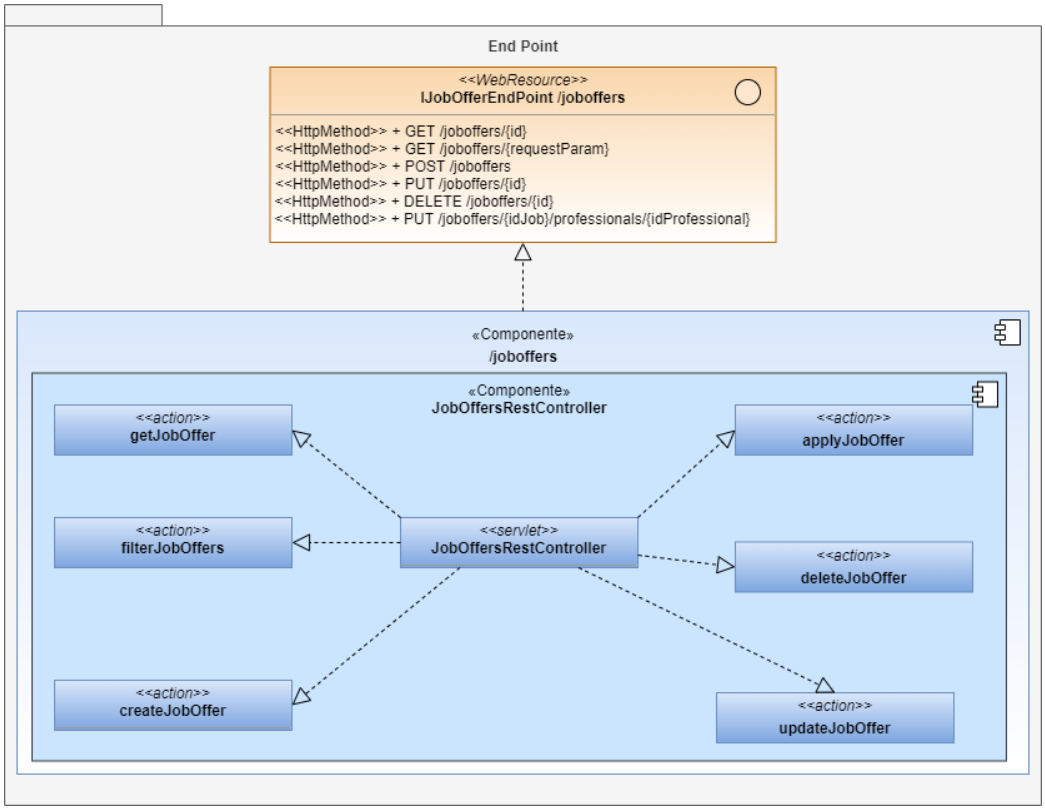


ILUSTRACIÓN 48 – JOBOFFER ENDPOINT REST CONTROLLER DETALLE

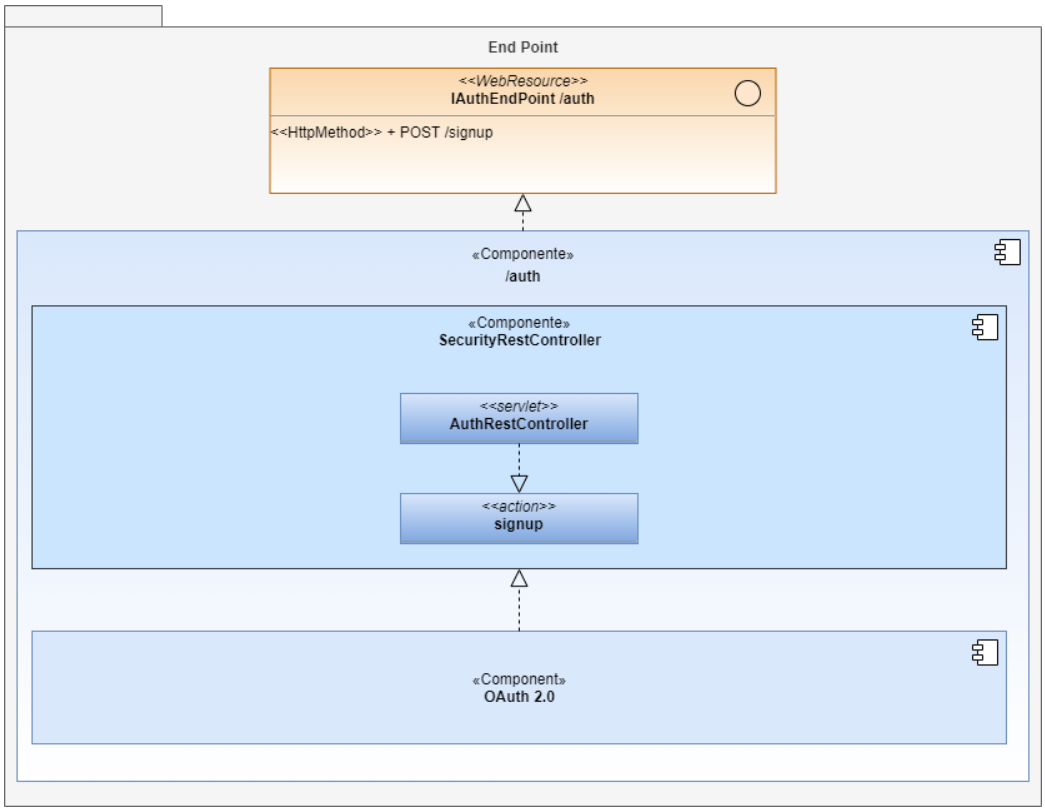


ILUSTRACIÓN 49 – AUTH ENDPOINT REST CONTROLLER DETALLE

4. Implementación

El detalle de la implementación completa del producto obtenido puede verse detalladamente en el capítulo de "Pruebas" del presente documento.

Todas las herramientas y tecnologías utilizadas en el presente trabajo son software libre.

4.1. Implementación de los endpoints

Para la implementación de los end points he desarrollado un conjunto de controller rest de spring, para dar servicio a cada componente definido en el presente documento en el apartado de diseño del sistema.

Cada controller rest ha sido anotado con la url "/api/wwj" y cada método del controller con la operación REST que permiten y el nombre en plural del recurso que publican, siguiendo las convenciones de buenas prácticas establecidas por la comunidad de desarrolladores (8).

Todos los end points han sido documentados aprovechando la integración de Spring con Open API (9)

4.2. Implementación del negocio

Para la implementación de la capa de negocio, he desarrollado los componentes necesarios y configurado los mismos como @Service.

4.3. Implementación de la integración

Para la capa de integración, he delegado en la integración que ofrece spring-data para trabajar con MongoDB, de forma que la mayoría de las operaciones las hace el repositorio que ofrece spring-data sin necesidad de hacer desarrollos o implementaciones adicionales.

En el caso de las búsquedas, utilizando la potencia a más bajo nivel que ofrece MongoTemplate, he implementado queries más específicas de forma que se pudiera filtrar de forma dinámica según los parámetros recibidos para la búsqueda.

4.4. Implementación de la seguridad

En cuanto a la seguridad del sistema, he utilizado la potencia de spring-security integrada con OAuth 2.0 como ya he indicado en el apartado del diseño.

A nivel de método he utilizado spring-security para impedir que perfiles no autorizados ejecuten operaciones que no corresponden.

La integración con Auth 2.0 permite registrar tantos clientes como sea necesario para que puedan hacer uso de la API y protege los recursos de accesos no deseados.

4.5. Notas al proceso de desarrollo

Una vez abordado el desarrollo, basado en TDD y tras llevar a cabo los primeros tests unitarios para el alta de usuarios, el login y el logout, se detecta la necesidad de llevar a cabo cambios relevantes sobre el diseño inicial en cuanto a las urls disponibles para tal fin que paso a detallar:

- Las URL's especificadas en el US-001, POST sobre /professionals y en el US-005, POST sobre /companies resultan redundantes e innecesarias, por tanto, que el POST a la url /auth/signup (end point definido para AuthController) satisface los requisitos de estas historias de usuario, de modo que en la versión que se entrega se eliminan los POST sobre /professionals y /companies para creación de usuarios con estos perfiles.

- Detecto así mismo la necesidad de implementar un método adicional en el servicio de usuarios que valide correctamente la unicidad del email a registrar y en caso de que exista devuelva una respuesta adecuada al caso.

- Así mismo y tras un conocimiento más detallado del sistema de seguridad que ofrece OAuth 2.0, se descarta la url /logout definida para AuthController y se opta por dejar esta funcionalidad en manos del tiempo de validez del token retornado por OAuth. Con el objetivo de facilitar las pruebas del sistema se establece un tiempo de validez del jwt token de 1000 segundos si bien lo ideal en un entorno de producción es reducirlo a unos 100 segundos y explotar el refresh token para dar continuidad a la autorización de acceso al recurso.

- Por el mismo motivo se elimina la url /login ya que OAuth y Spring Security cubrirán este requisito.

- La url raíz de la API a nivel general se define como **/api/wwj**, si bien se deja acceso libre y anónimo al root del proyecto para que swagger pueda funcionar correctamente sin requerir autorización y se deja acceso libre al GET de /jobOffers con la finalidad de que la búsqueda de ofertas de empleo esté accesible a nivel general y se deja acceso libre a /signup para permiter el registro de usuarios.

- Continuando con los tests unitarios y contrastando con la documentación elaborada en el diseño, se detecta la falta de algunos métodos fundamentales para el cumplimiento de requisitos fundamentales de la idea de este proyecto en la documentación, tal como la búsqueda de ofertas y la capa de integración. Así mismo encuentro errores menores en la definición de las urls que se subsanan en el desarrollo.

- Se necesitan más códigos de respuesta de los especificados en el diseño inicial que se detallan en la documentación de las pruebas realizadas.

- Los métodos de la capa de integración se reducen considerablemente al delegar en spring-data con lo que la capa de integración se simplifica, dejando en la

capa de negocio como se había previsto inicialmente la lógica adicional que se necesite en determinados casos.

- Para operaciones en las que es necesario recuperar el usuario logado, recurro al contexto de seguridad de spring, que me devuelve el nombre de usuario y a partir de ahí enviar al negocio todos los datos necesarios para la lógica.

- El desarrollo con MongoDB, tecnología totalmente desconocida para mí, ha resultado más difícil de lo esperado, puesto que el cambio de filosofía de bases de datos relacionales a no relacionales me ha costado asimilarlo más de lo que esperaba si bien creo que he conseguido solventarlo, al menos funcionalmente aunque ya tengo ideas de mejora a este respecto.

- Las restricciones de unicidad del nombre de usuario (email en este caso) las he delegado en la lógica de negocio del registro ya que he tenido diversos problemas trabajando con los index de spring data y mongodb.

4.6. Tecnologías y Herramientas

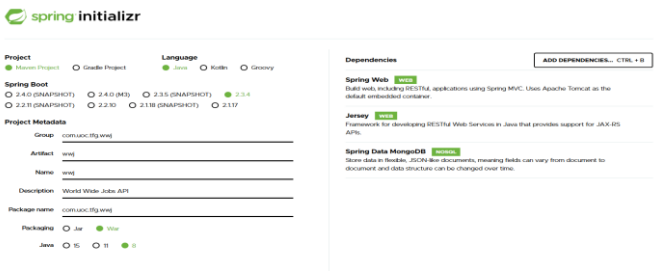
Eclipse IDE for Enterprise Java Developers

El entorno elegido es Eclipse IDE, por el conocimiento previo de la herramienta, porque es un IDE ampliamente extendido y muy extensible, tiene buen soporte de la comunidad dentro del abanico de herramientas 100% gratuitas.

Spring-Boot

Spring Boot es un framework de Spring que simplifica tanto la creación como el despliegue de una aplicación, permitiendo al desarrollador centrarse en lo verdaderamente importante, crear el código, evitando algunos dolores de cabeza conocidos por todos los que habitualmente trabajan en el desarrollo de aplicaciones.

Desde la página / podemos en unos sencillos pasos crear un proyecto base listo para importar en nuestro eclipse y comenzar a trabajar.



Persistencia

Tras debatirme entre las bases de datos relacionales tradicionales y las bases de datos NoSQL, he optado por estas últimas, concretamente por MongoDB, puesto que el sistema de almacenamiento de datos debe ser muy escalable y además MongoDB incluye algunas características que podrían facilitar el trabajo de los datos relacionados con franjas horarias. Además de lo mencionado, Spring Data ofrece una buena integración con MongoDB haciendo el trabajo con este sistema de persistencia muy transparente al desarrollador.

Maven

Ampliamente usada por la comunidad y muy potente, he elegido maven para la gestión de dependencias del proyecto, automatización de builds y ejecución de test previos.

Control de versiones

Cada vez más extendido y aceptado por la comunidad, por su facilitad en el trabajo con ramas y por su naturaleza distribuida, que garantiza que los cambios no se pierden si el servidor cae, he elegido Git y crearé en mi cuenta de BitBucket un repositorio para este proyecto.

Para la gestión del repositorio usaré como cliente SourceTree, ya que es una herramienta sencilla de usar, con una curva de aprendizaje rápida y totalmente gratuita.

Herramientas Test y Documentación

Para el desarrollo principal he usado JUnit, como herramientas para la especificación de pruebas que debe superar el código ya que el desarrollo de este proyecto será orientado a tests (TDD)

Para las pruebas manuales durante el desarrollo, utilizaré Postman y para la documentación de la API utilizaré Swagger, que es una herramienta sencilla y cada vez más usada por los profesionales del sector para la documentación de APIs.

Herramienta de planificación y gestión

Para la gestión de sprints y la evolución del proyecto, he usado trello como herramienta sencilla y gratuita, que en este caso cubre suficientemente las necesidades para este trabajo.

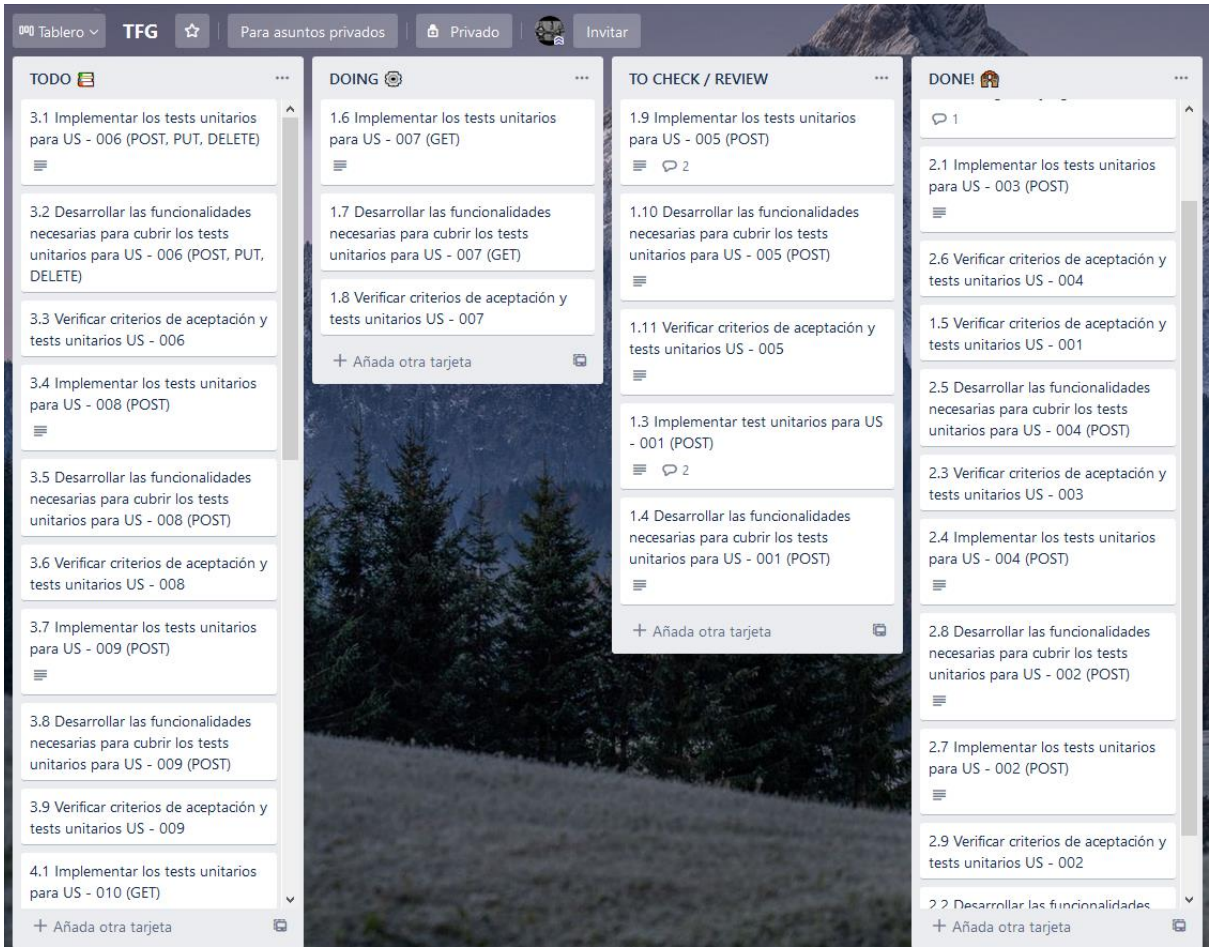


ILUSTRACIÓN 50 – VISTA TRELLO PROYECTO

5. Pruebas

A continuación, se presentan las pruebas desarrolladas inicialmente y las llevadas a cabo posteriormente con postman para cada caso de uso a todos los niveles de la arquitectura de la aplicación. He podido llevar a cabo algunas pruebas con swagger que se encuentra accesible y documentado en http://localhost:8080/swagger-ui.html, si bien la mayoría de la pruebas por rapidez y conocimiento de la herramienta las he llevado a cabo con Postman.

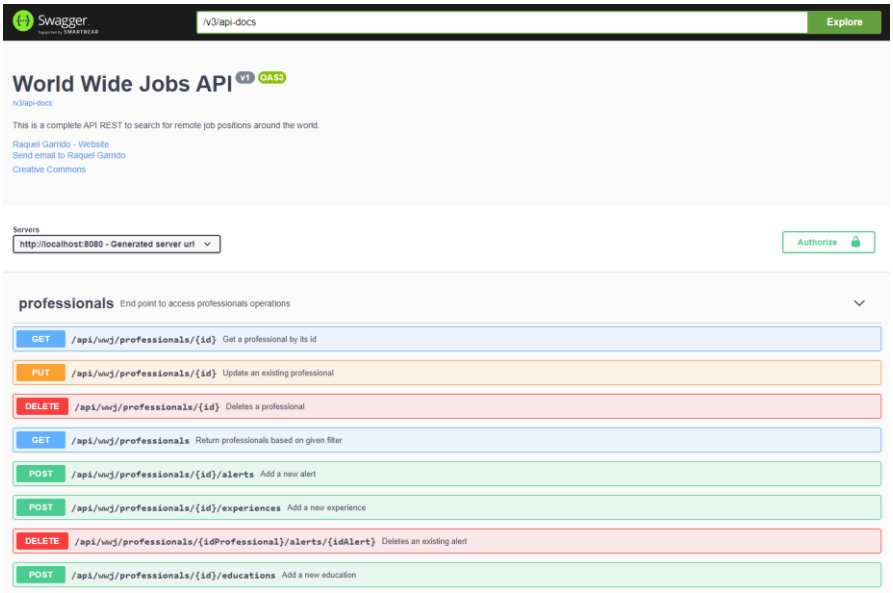


ILUSTRACIÓN 51 – VISTA SWAGGER

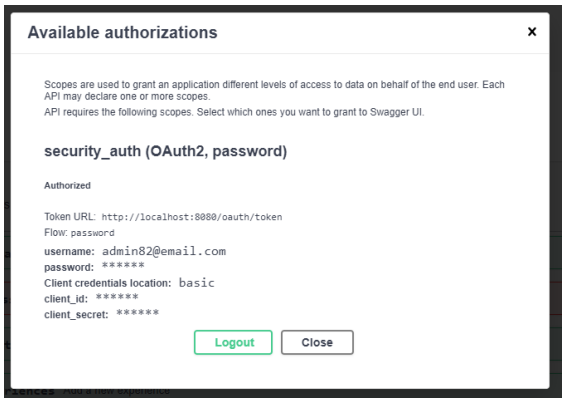


ILUSTRACIÓN 52 – SWAGGER AUTH

A grandes rasgos, he estructurado los tests de la siguiente forma:

- A nivel de repositorio, verifico que todos los métodos que se utilizarán de la capa de datos responden según lo esperado.
- A nivel de servicio, verifico la funcionalidad de la lógica "mockeando" la capa de datos.
- A nivel de recursos o controllers, verifico la funcionalidad de cada url, la respuesta con parámetros habituales y excepcionales y los requisitos de seguridad de cada método.
- A nivel de integración, compruebo que todos los componentes encajan y responden según lo esperado, amplio las pruebas sobre la seguridad del sistema a nivel de metodos e implemento tests más a fondo sobre las búsquedas ya que en ellas se basa la idea principal de este proyecto.

- Con Postman, valido de nuevo toda la funcionalidad, generando datos más "reales" en la base de datos y ejecutando toda variedad de búsquedas que se me ocurren.

En el segundo apartado del anexo se detallan y documentan a fondo toda la batería de pruebas llevadas a cabo, tanto en el desarrollo guiado por tests como en las pruebas finales una vez implementada la aplicación.

6. Mejoras y evolución del proyecto

El objetivo del presente trabajo es ofrecer una api orientada a la publicación y búsqueda de ofertas de trabajo en modalidad de teletrabajo o remoto.

Dado el limitado espacio de tiempo para desarrollar el trabajo, he dejado fuera del alcanza opciones que me parecían muy interesantes, como por ejemplo:

- Explotar el tipo de dato GEOJsonPoint que ofrece MongoDB para poder definir áreas acotadas de búsqueda a nivel global, más allá de las franjas horarias, para permitir la búsqueda por ejemplo, en diversas zonas o países concretos independientemente de la zona horaria. Esto sería muy útil para los nómadas digitales que planean vivir en distintas zonas a lo largo de un periodo de tiempo.

- Integrar elasticsearch en el proyecto, puesto que en un entorno real de producción el volumen de ofertas y usuarios activos exigiría una capa especializada para la gestión de búsquedas y el análisis de los datos.

- Aislar la implementación de Oauth 2.0. En este proyecto se ha implementado Oauth en el mismo proyecto que lo explota, desarrollando una solución todo en uno, pero lo ideal sería implementar OAuth 2.0 en otro servidor de forma que se aisle la gestión de los posibles clientes a registrar y la lógica de los accesos del proyecto que contiene los recursos reales a los que accede.

- Implementar un sistema de avisos, configurable por el usuario, que permita recibir en el correo cada cierto tiempo un resumen de las ofertas publicadas en la plataforma coincidentes con las alertas creadas por el usuario.

- Así mismo, ampliar este sistema de avisos a la creación de notificaciones en la nube para permitir la recepción de avisos en clientes móviles a través de sistemas push o de naturaleza similar.

- Desarrollar un cliente integrado con un servicio de mapas, en cuyo caso habría que estudiar el coste económico de los servicios actuales en el mercado, Google Maps por ejemplo tiene una tarifa establecida por cada petición realizada a su api, lo que implica hacer un estudio de eficiencia en las consultas a apis de terceros para evitar sobrecostes.

7. Conclusiones

A pesar de que tengo experiencia en el sector del desarrollo de aplicaciones Java, analizar, diseñar y desarrollar un proyecto completo desde cero ha sido todo un reto.

En los diversos proyectos en los que he participado a lo largo de mi carrera profesional, he tenido la ocasión de intervenir en distintas partes del ciclo de desarrollo, pero nunca había tenido la oportunidad de intervenir en todas las fases del proyecto en un solo proyecto.

Este trabajo me ha permitido ejercer roles de analista, arquitectura, desarrolladora y gestión de proyecto a lo largo de todo el trabajo, derivando en una experiencia muy enriquecedora a nivel académico y personal.

Para llevarlo a cabo con éxito, además de mi experiencia profesional previa, he explotado los conocimientos previamente adquiridos a lo largo de mis estudios académicos, tales como:

- Análisis y Diseño con Patrones, que me ha permitido reconocer, identificar y detectar los más adecuados en cada caso.
- Gestión de proyectos, de cara a la planificación temporal de los sprints y el enfoque ágil que he querido darle a este trabajo.
- Ingeniería de Requisitos e Ingeniería del Software, que me han permitido elaborar el análisis y diseño con garantías, así como definir los elementos necesarios a nivel de arquitectura del sistema.
- Diseño de bases de datos, me ha servido como base para en esta recta final de mi formación académica, atreverme con sistemas más actuales como NOSql y tener una visión diferente de la gestión de los datos.

8. Glosario

- IDE: Acrónimo en inglés de Entorno Integrado de Desarrollo, herramienta que facilita el desarrollo integrando en la misma todas las herramientas necesarias
- API: Acrónimo para Interfaz de Programación de Aplicaciones, una API presenta las operaciones permitidas para que dos componentes software se comuniquen entre sí.
- REST: Acrónimo de Representational State Transfer, es un estilo de arquitectura software stateless que utiliza el protocolo HTTP para el intercambio de información entre distintos sistemas en formatos muy específicos (json o xml)
- Framework: Es un conjunto de librerías, utilidades, guías, prácticas o herramientas software que facilitan la resolución e implementación de una aplicación.
- SCRUM: Es un proceso definido de trabajo para entornos cambiantes, englobado dentro de las metodologías de gestión ágiles y que define un conjunto de prácticas, normas y roles para garantizar entregas de valor en cortos periodos de tiempo.
- SPRINT: Es el periodo de tiempo que se define hasta alcanzar el siguiente entregable de valor para el cliente, cada sprint se compone de una serie de tareas a ejecutar extraídas del backlog.

- Backlog: Es una lista de tareas ordenadas por prioridad, dichas tareas en su conjunto engloban todo el desarrollo del proyecto.
- TDD: Acrónimo en inglés de Desarrollo guiado por pruebas, es un método de desarrollo basado en elaborar primero los tests que verifican los requerimientos a cumplir para posteriormente ir desarrollando las implementaciones necesarias para que los tests se cumplan.

9. Bibliografía

1. Peek, S. (2020, 18 marzo). *Communication Technology and Inclusion Will Shape the Future of Remote Work*. Business News Daily. https://www.businessnewsdaily.com/8156-future-of-remote-work.html

2. Navarro, D. (2020, 11 mayo). *El teletrabajo, una perspectiva global: evolución y actualidad del trabajo en remoto*. RRHHDigital. http://www.rrhhdigital.com/secciones/actualidad/141722/El-teletrabajo-una-perspectiva-global-evolucion-y-actualidad-del-trabajo-en-remoto

3. *index | TIOBE - The Software Quality Company*. (s. f.). TIOBE. https://www.tiobe.com/tiobe-index/

4. Jeremy. (2020, 17 julio). *Writing Great User Stories For Developing APIs*. Jeremy Jarrell. https://www.jeremyjarrell.com/user-stories-apis/

5. Todd Fredrich, Pearson eCollege. (2018). *HTTP Status Codes*. Wikipedia. https://www.restapitutorial.com/httpstatuscodes.html

6. Kwangchul Shin, K. S., Chulhyun Hwang, C. H., & Hoekyung Jung, H. J. (2017). *NoSQL Database Design Using UML Conceptual Data Model* (ISSN 0973-4562 Volume 12). Research India Publications. https://www.ripublication.com/ijaer17/ijaerv12n5_12.pdf

7. Das, S. (2020, 27 junio). *Securing Spring Boot REST API with JSON Web Token and JDBC Token Store*. Medium. https://medium.com/@dassum/securing-spring-boot-rest-api-with-json-web-token-and-jdbc-token-store-67558a7d6c29

8. A. (2020, December 22). REST Resource Naming Guide - REST API Tutorial. REST API Tutorial |. https://restfulapi.net/resource-naming/

9. Home. (2020, 12 noviembre). OpenAPI Initiative. https://www.openapis.org/

10. Anexos

10.1. Instalación y ejecución

Para poner en funcionamiento el sistema desarrollado será necesario instalar las siguientes herramientas en nuestro equipo:

- jdk1.8.0_241
- MongoDB 4.4.1 Community Edition (actualmente en la web se encuentra disponible la 4.4.2) https://www.mongodb.com/try/download/community
- Eclipse IDE for Enterprise Java Developers. Version: 2019-12 (4.14.0)
- Maven : Aunque se puede instalar aparte, en este caso he optado por la versión embebida que ofrece eclipse.
- MongoDBCompass 1.24.6 para cargar datos de prueba, comprobar los resultados en la base de datos y gestionar los esquemas desde una interfaz más amigable.
- La configuración de la base de datos se encuentra en los ficheros application.properties del proyecto, se ha creado una base de datos real para la ejecución del proyecto y una base de datos de test en caso de ser necesaria para pruebas.

  mongodb://myUserAdmin:Cheto%40123@localhost:27017/admin

  WorlWideJobs
  WorlWideJobsTests

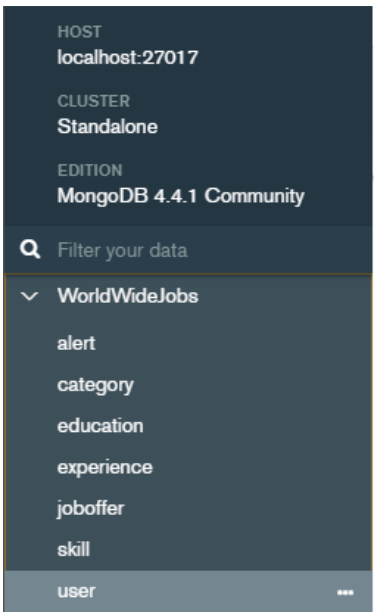


ILUSTRACIÓN 53 – DB CONFIG

- Postman https://www.postman.com/
- Clonaremos el proyecto alojado en https://Asheara@bitbucket.org/Asheara/worldwidejobs.git
- Lo importamos en nuestro eclipse : File > Import > Existing Maven Projects y desde el explorador de Windows seleccionamos la carpeta donde hemos clonado el proyecto

  Dentro del proyecto, encontraremos diversos ficheros en formato json en src/test/resources/data que utiliza el sistema para ejecutar las pruebas unitarias y de integración. Para ejecutar los tests sobre eclipse, nos situamos sobre la carpeta src/test/java y desplegando el menú contextual de esa carpeta seleccionamos Run as > Junit Test

  Para completar los tests con éxito, en algunos niveles hago uso de una utilidad que he implementado que lee los ficheros json con los datos de prueba y los carga en la base de datos en memoria que utiliza spring-boot y finalmente destruye los registros creados al finalizar la ejecución, por este motivo es importante que los ficheros con los datos de prueba se encuentren en la carpeta indicada.

  Los ficheros de datos que utiliza la implementación de los tests no son válidos para importar en la base de datos real puesto que ya cuentan con un id aleatorio y mongo le asigna a cada registro un nuevo id (o yo al menos no he llegado a averiguar como evitar este comportamiento), de modo que para pruebas reales se recomienda el juego de datos inicial de usuarios proporcionado en src/main/resources/data.

  Los datos para el servidor de OAuth 2.0 son:

  clientId: wwj-client
  client-secret: secret

  Se adjunta un juego inicial de usuarios (administrador, compañía y profesional) listo para importar con MongoDBCompass con el objetivo de que se pueda comenzar a generar datos desde postman de forma inmediata, cuyos usuarios/claves son:

  raquela82@gmail.com / profess@123

  47Deg@email.com / 47Deg@1982

  admin82@email.com / P4ss@82

  Las url's que expone el servicio se detallan más adelante en el apartado de implementación y en el de pruebas ejecutadas.

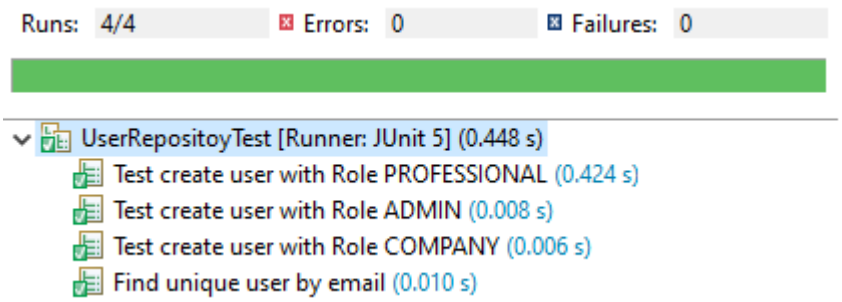
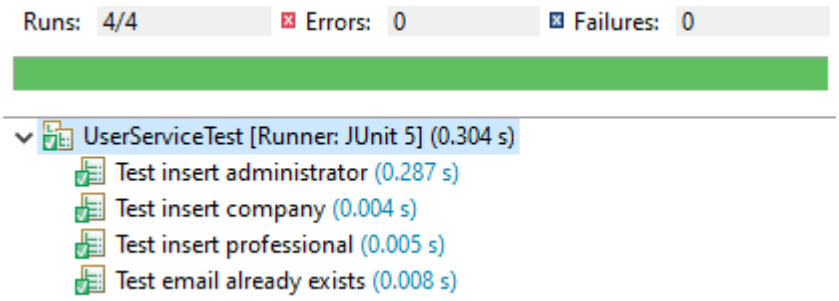
  La aplicación corre sobre el puerto habitual en http://localhost:8080 y la interfaz de swagger está disponible en http://localhost:8080/swagger-ui.html

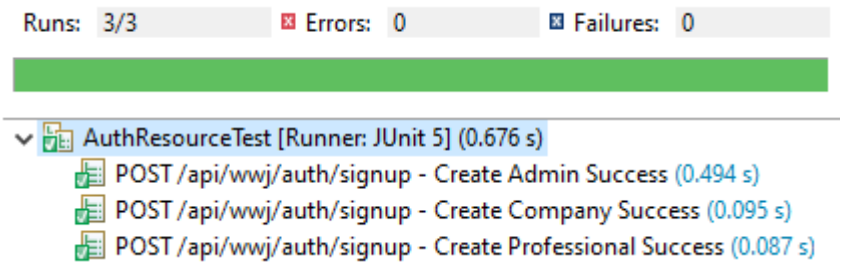
Las pruebas con swagger me han dado algunos problemas y resultados inesperados en algunos casos, por lo que recomiendo usar Postman en su lugar que ha sido más fiable durante todo el proceso.

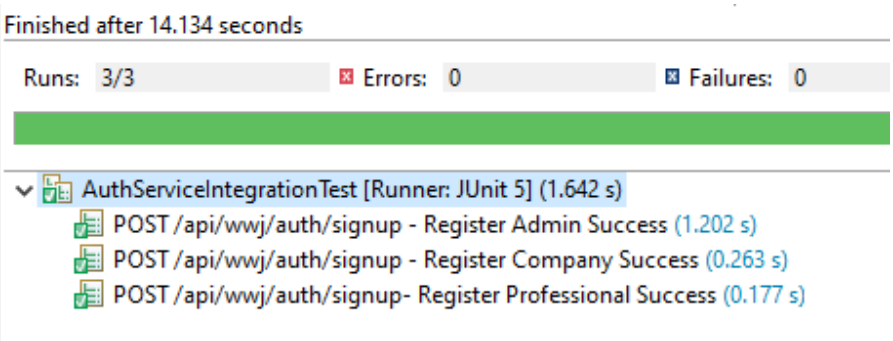
10.2. Test unitarios y pruebas del sistema

A continuación, se presenta un detalle de pruebas sobre los niveles indicados en el capítulo 5, por cada caso de uso especificado en el diseño, 13 en total.

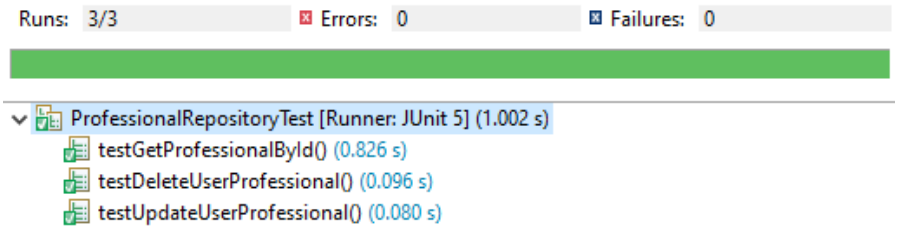
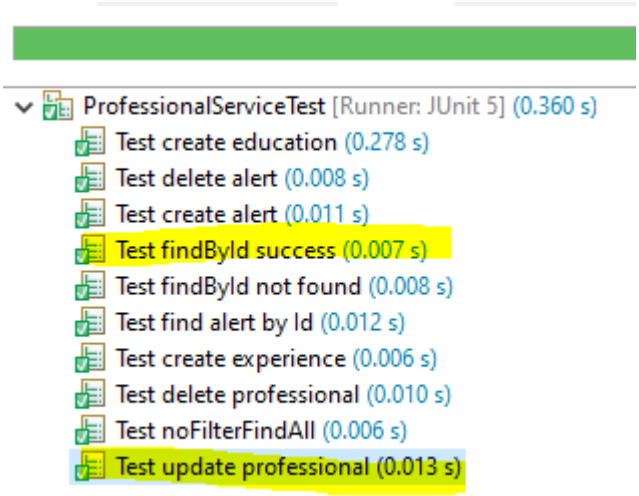
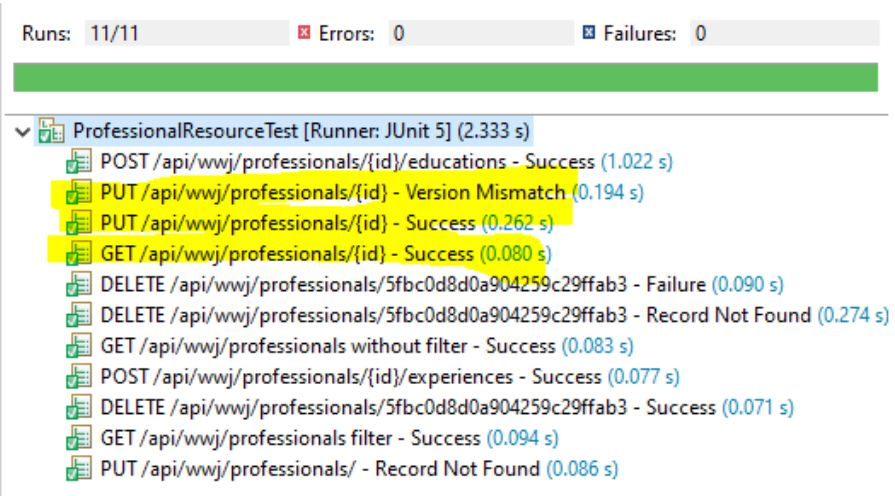
TABLA 17 – DETALLE TESTING HISTORIA DE USUARIO 001

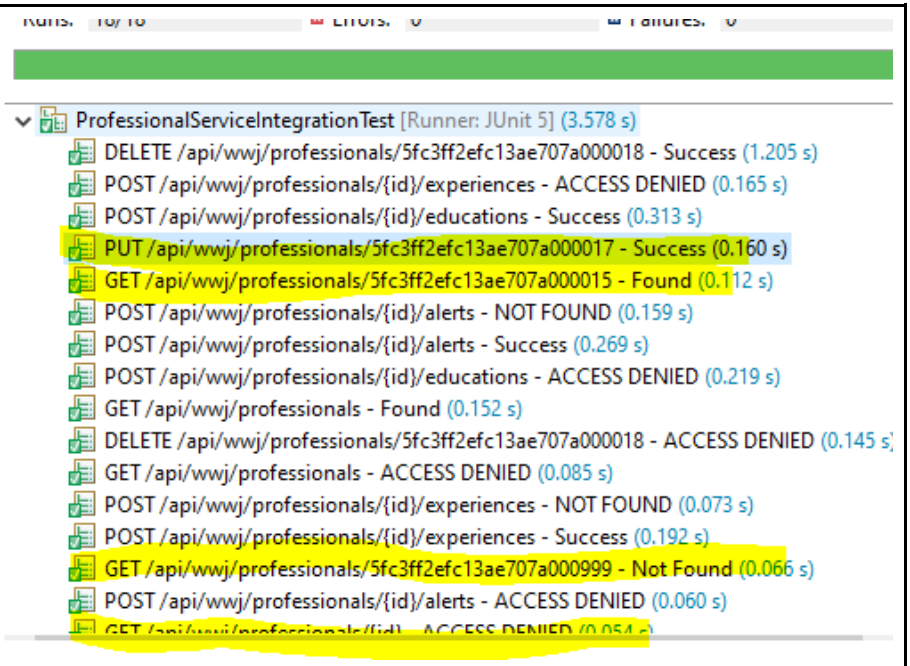
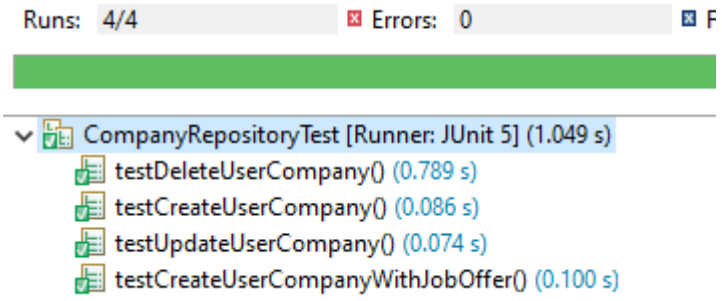
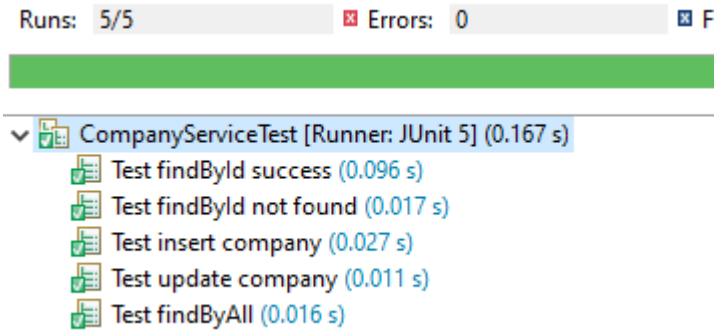
US - 001	Cómo profesional quiero registrarme en la plataforma para dar de alta mis datos personales y profesionales y estar visible en la plataforma.
Test End Point - Junit	
POST	/auth/signup
RepositoryTest	
	Verificamos que la capa de integración funciona correctamente a la hora de registrar usuarios. No se verifica el borrado puesto que se deja a la unidad de negocio asociada a la entidad final (profesional o empresa)
ServiceTest	

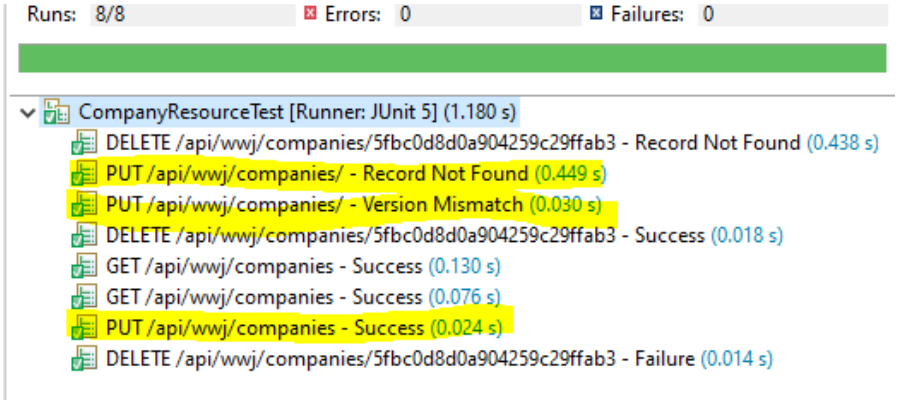
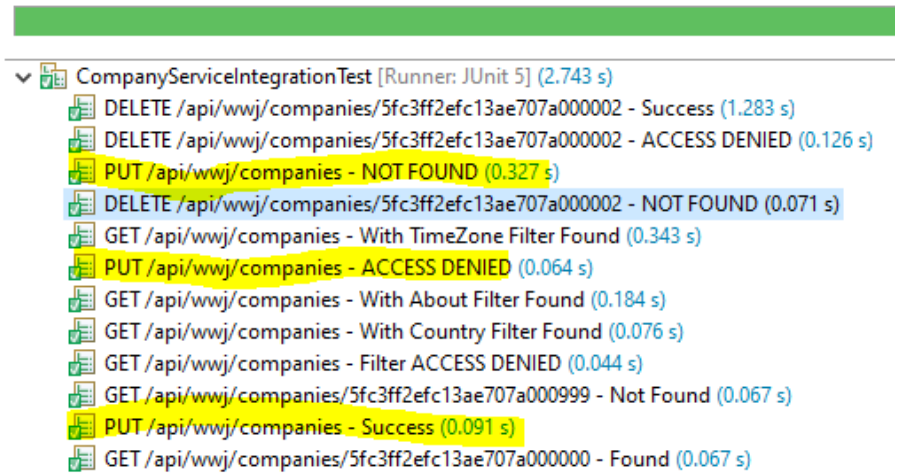
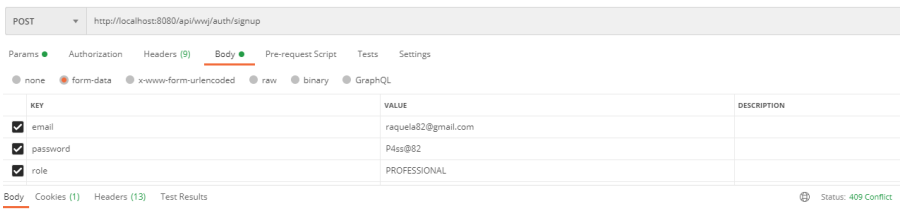
<table>
<tr><td></td><td>A nivel de servicio pretendo verificar la continuidad de los métodos necesarios. En esta capa de negocio además se incluye un método que apoyándose en el findByEmail testeado en el repositorio, responderá con true o false si el email ya existe, puesto que al utilizarse como nombre de usuario este debe ser único, decido implementar este control en el servicio antes de que la capa de datos deba responder a la restricción de unicidad.</td></tr>
<tr><td rowspan="2">ResourceTest</td><td> 
</td></tr>
<tr><td>Verificamos que efectivamente haciedo POST a la url de registro el usuario recibe la respuesta es la esperada.

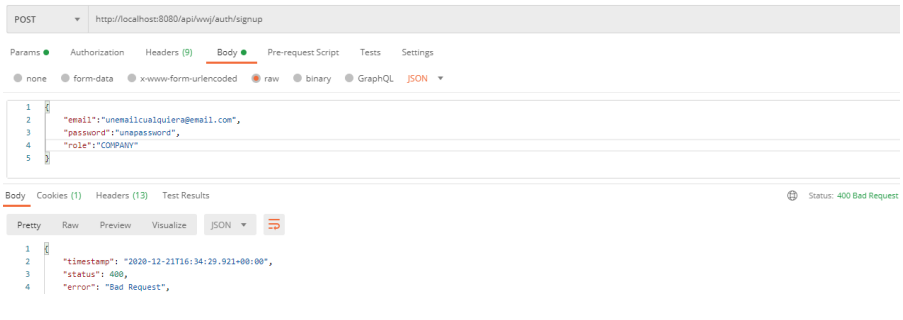
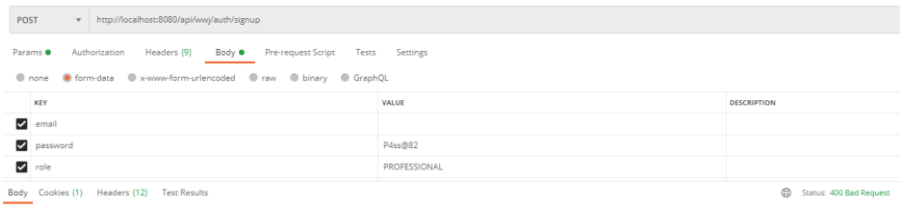
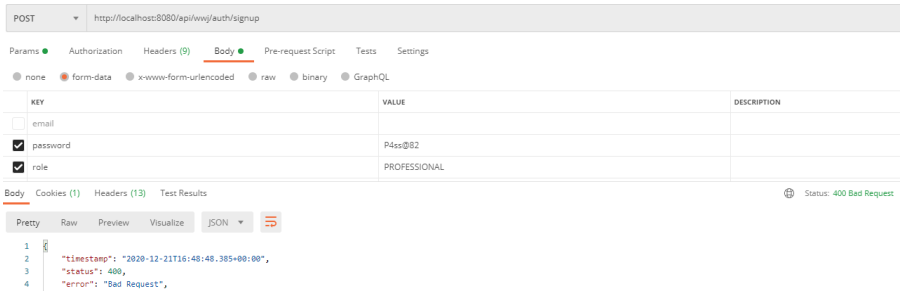
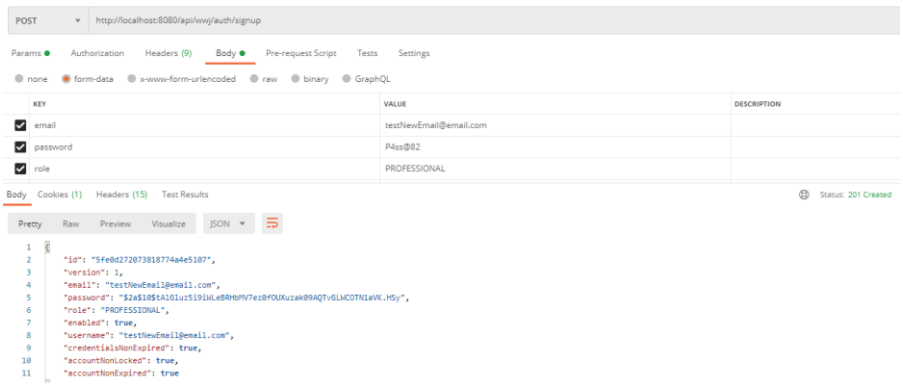
mock.perform(MockMvcRequestBuilders.post("/api/wwj/auth/signup").contentType(MediaType.APPLICATION_JSON)
.param("email", "raquela82@gmail.com").param("password", "somP4ss").param("role","COMPANY")).andExpect(status().isCreated()).andExpect(content().contentType(MediaType.APPLICATION_JSON)).andExpect(header().string(HttpHeaders.LOCATION, "/api/wwj/users/5fbc0d8d0a904259c29ffab3")).andExpect(jsonPath("$.email", is("raquela82@gmail.com")));</td></tr>
<tr><td>Integration</td><td> 

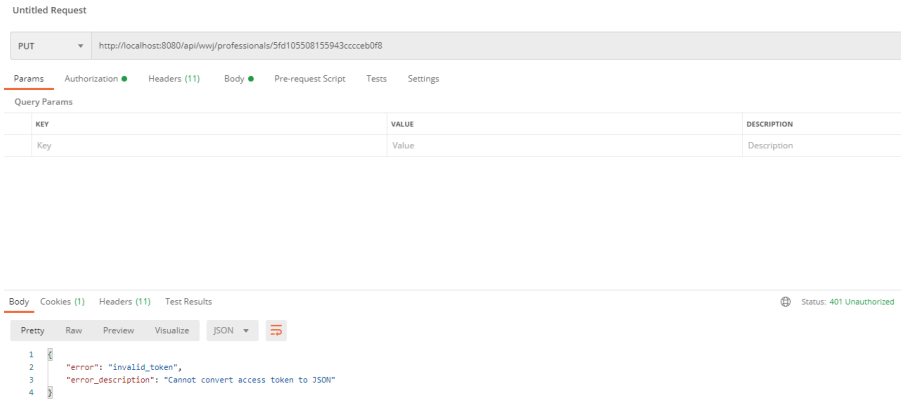
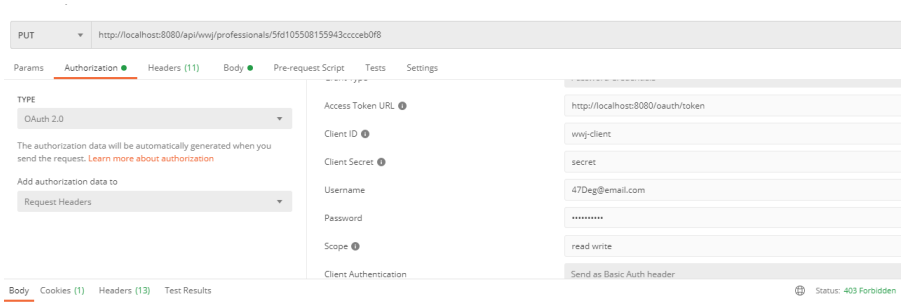
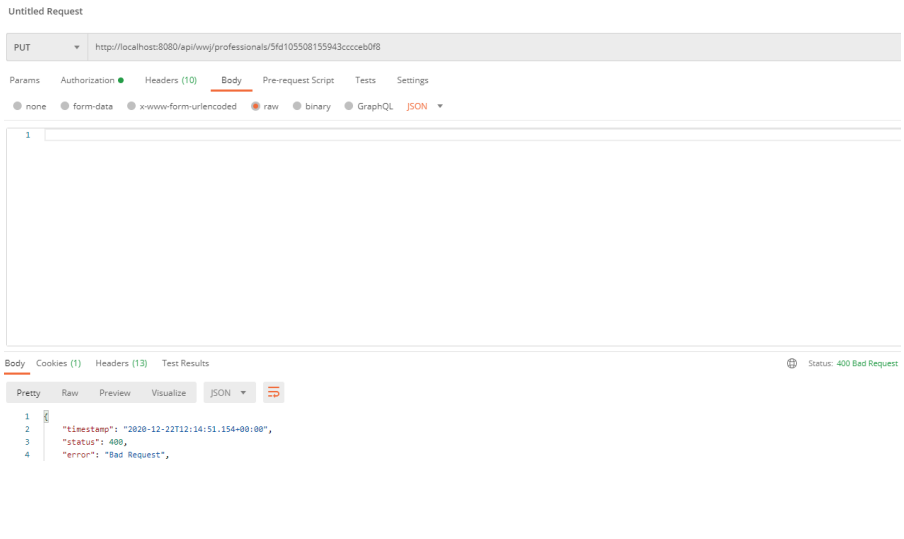
Validamos que la integración a todos los niveles como test final es correcta.</td></tr>
<tr><td>PUT (profesionales)</td><td>/professionals/{id}</td></tr>
</table>

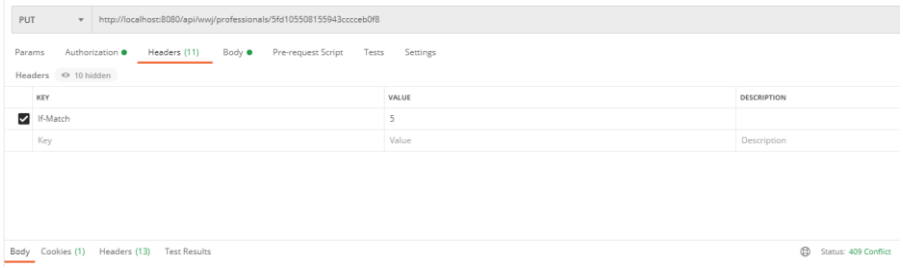
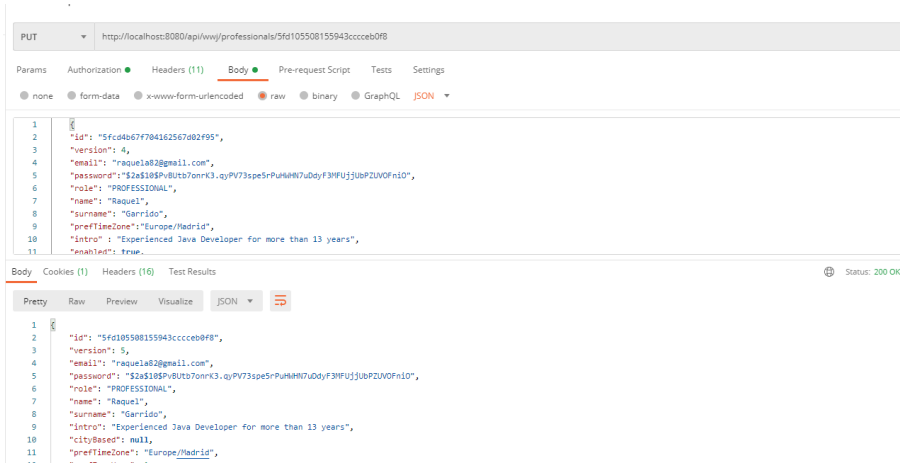
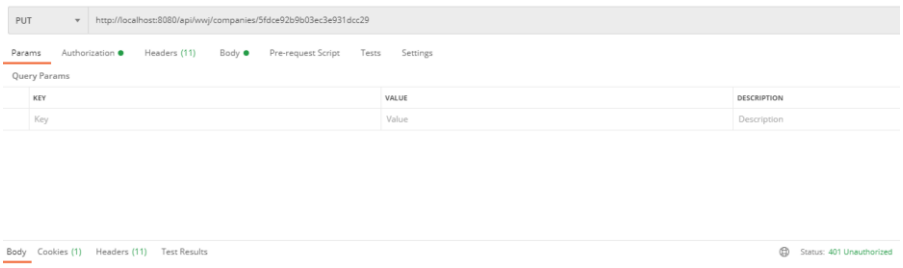
Repository	 Verificamos que lo métodos básicos a nivel de capa de datos que necesitaremos responden según lo esperado
Service	 Testeo en el servicio las operaciones que se utilizarán a lo largo del negocio
Resource	 Verifico a nivel de recursos que los tests pasan para los casos normales y las incidencias más habituales (errores de versionado y no encontrados)

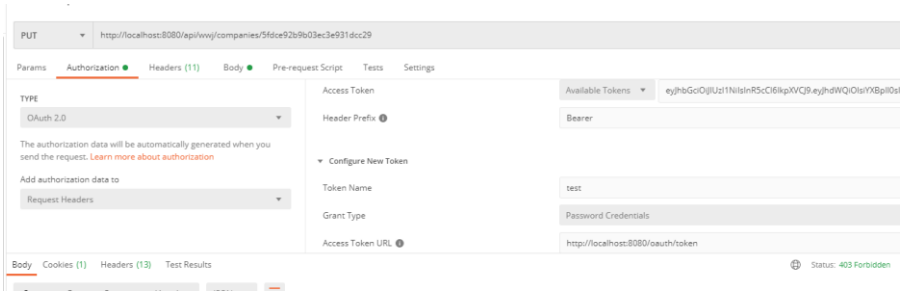
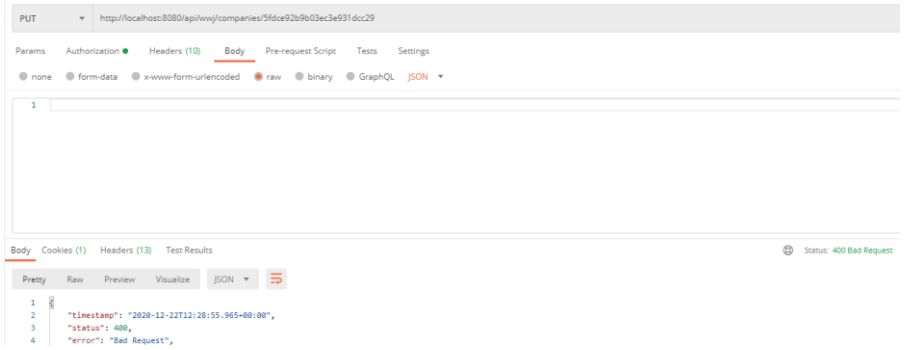
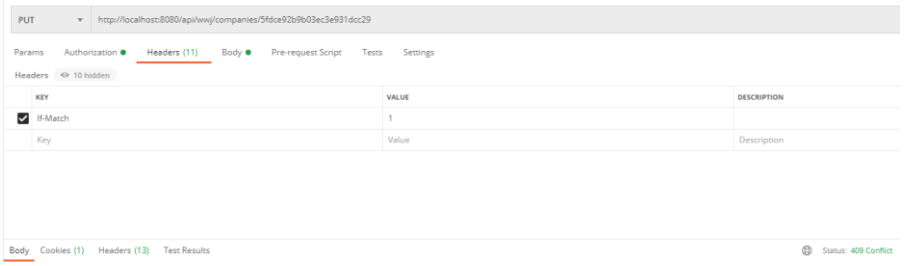
Integration	 Y por último testeamos en integración que todos los componentes encajan con seguridad incluída y compruebo accesos a nivel de método
PUT (empresas)	/companies/{id}
Repository	
Service	

Resource	
Integration	
Test End Point – Postman	
POST	/auth/signup
RESPONSE	
409 Error (email ya existe)	 Verificamos que efectivamente, si tratamos de registrar un usuario con un email que ya está registrado el sistema nos responde con un status code 409.

400 – La request no es correcta o acorde al formato esperado	 ---  En la captura anterior el sistema responde con un 400 controlado al recibir un parámetro requerido vacío. --- 
201 Created	
PUT (profesionales)	/professionals/{id}

401	 Sin autenticar
403	 Acceso denegado si autenticamos como empresa
400	

409 Conflict	 La cabecera de control de versión indica que hay un recurso más actualizado que el que envíamos nosotros
200 OK	 Comprobamos que con los permisos adecuados actualiza correctamente y el versionado del recurso es coherente.
PUT (companies)	/companies/{id}
401	 Si no me autentico

403	 Si me autentico como profesional
400	 Si la petición no se hace en el formato esperado
409	 Si la cabecera de versionado indica que el servidor tiene un recurso más actualizado que el nuestro.

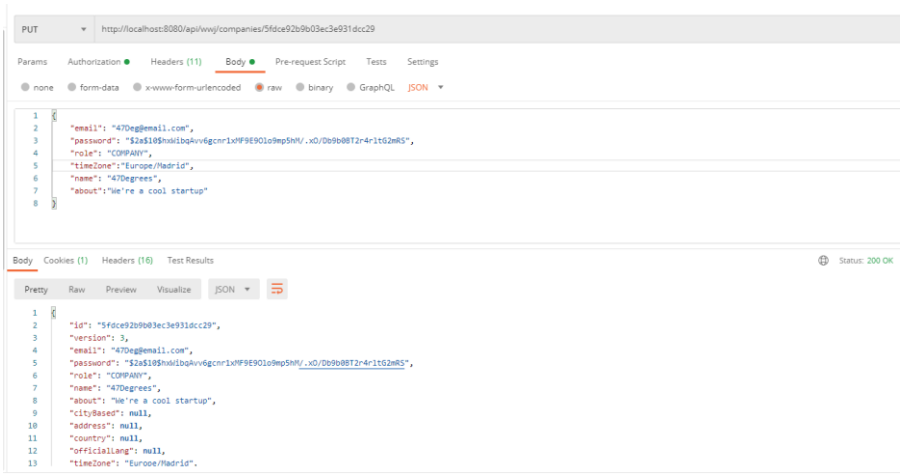
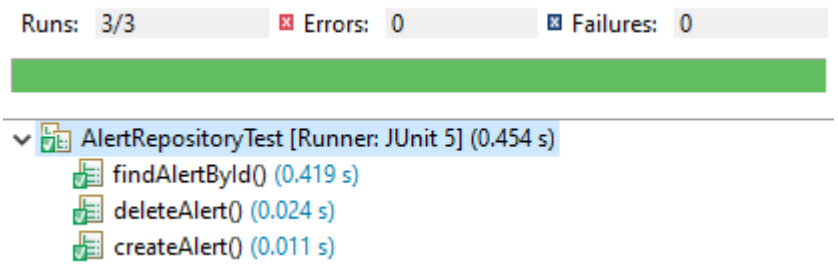
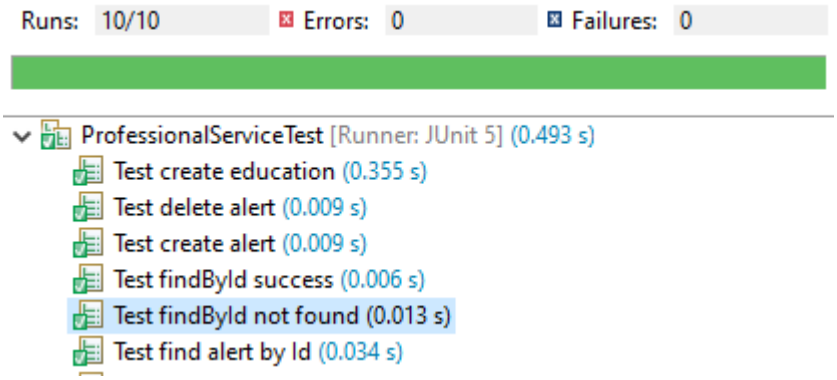
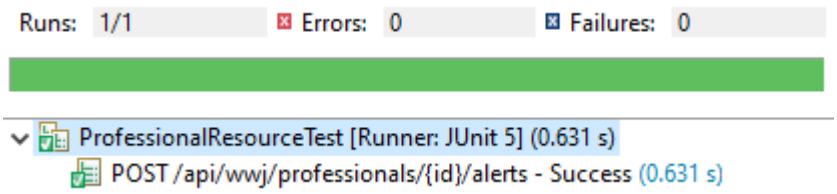
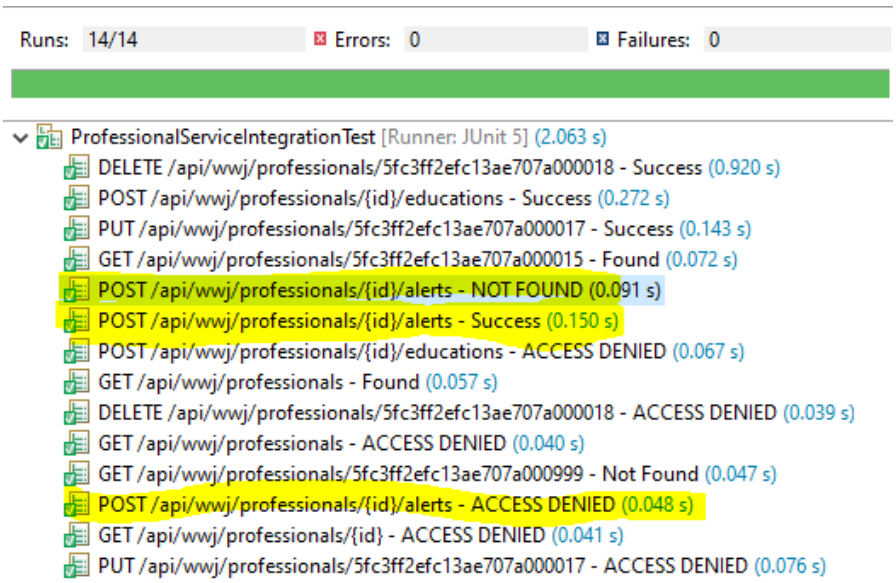
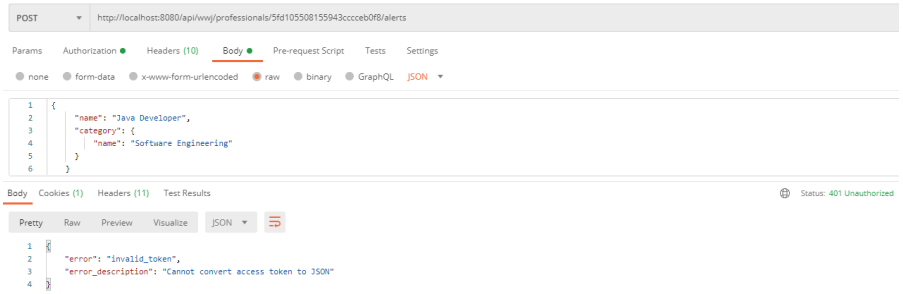
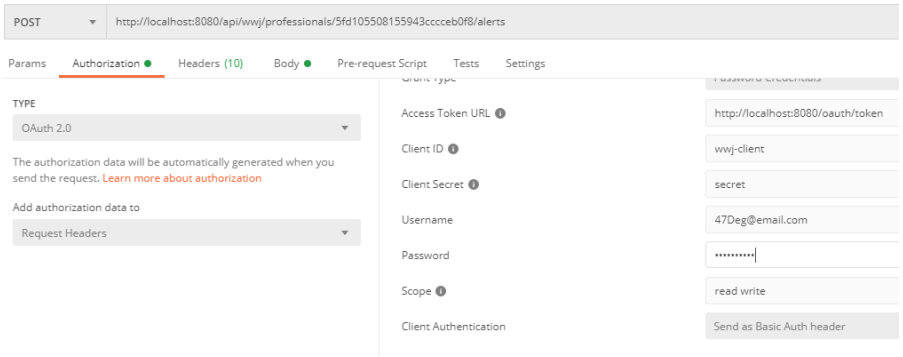
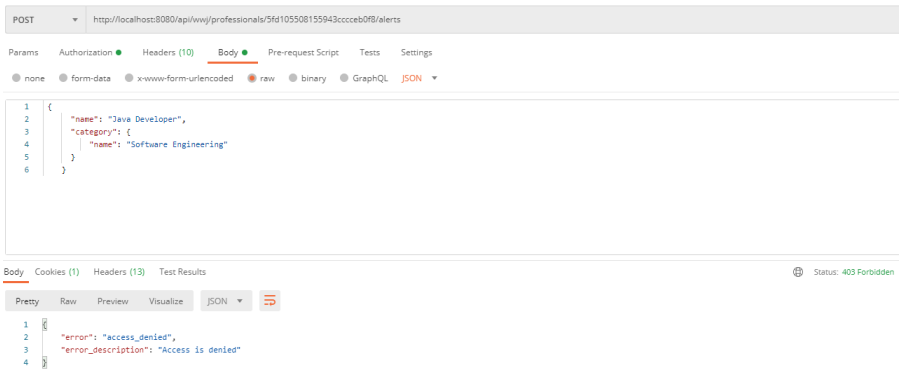
200 OK	 Autenticados como empresa y enviando una petición correcta

TABLA 18 – DETALLE TESTING HISTORIA DE USUARIO 002

US - 002	Cómo profesional quiero crear alertas sobre publicación de ofertas de trabajo en determinadas categorías y habilidades que poseo para estar al tanto de todas las posibilidades, incluídos los parámetros de zona horaria deseados e idioma preferido.
Test End Point - Junit	
POST	/professionals/{id}/alerts
RepositoryTest	
	Verificamos que la capa de integración funciona correctamente a la hora de crear y eliminar alertas, no se contempla la modificación de las mismas en esta primera versión.

ServiceTest	
	A nivel de servicio pretendo verificar la continuidad de los métodos necesarios tales como creación y borrado de alertas y la recuperación de alertas por identificador.
ResourceTests	
	Verificamos que efectivamente haciedo POST a la url de alertas con el profesional asociado el usuario recibe la respuesta es la esperada.
IntegrationTests	 En los tests de integración comprobamos las respuestas esperadas para los casos más habituales.

Test End Point - Postman	
POST	/professionals/{id}/alerts
401	 Si tratamos de crear una alerta sin haber enviado petición a Oauth previamente y autenticarnos, el sistema devuelve error unauthorized
403	 Si nos logamos como Company, el sistema devolverá un error de AccessDeniedException como observamos a continuación 

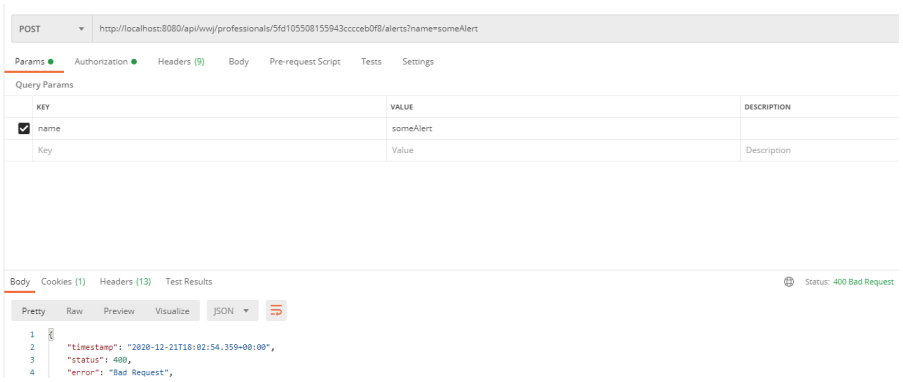
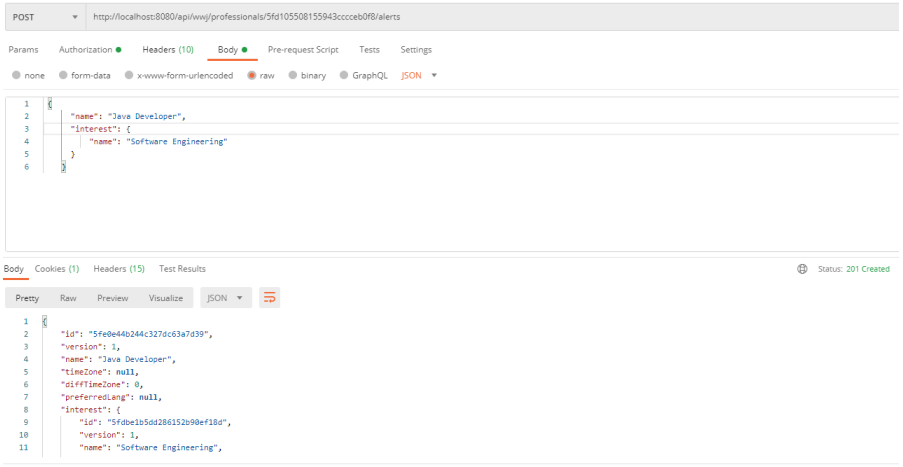
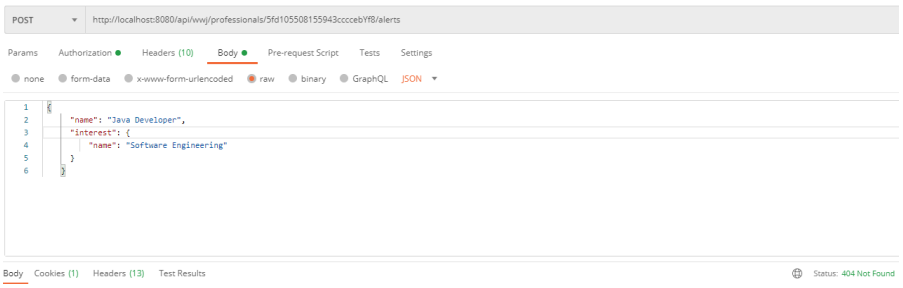
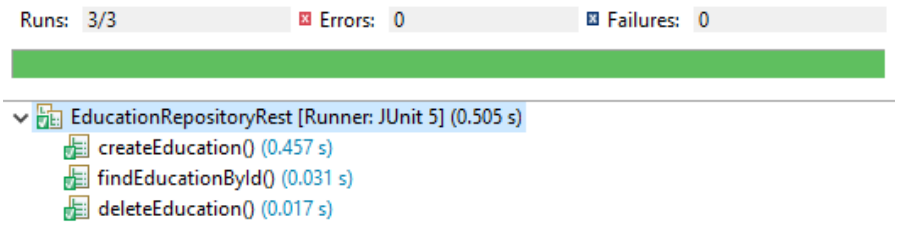
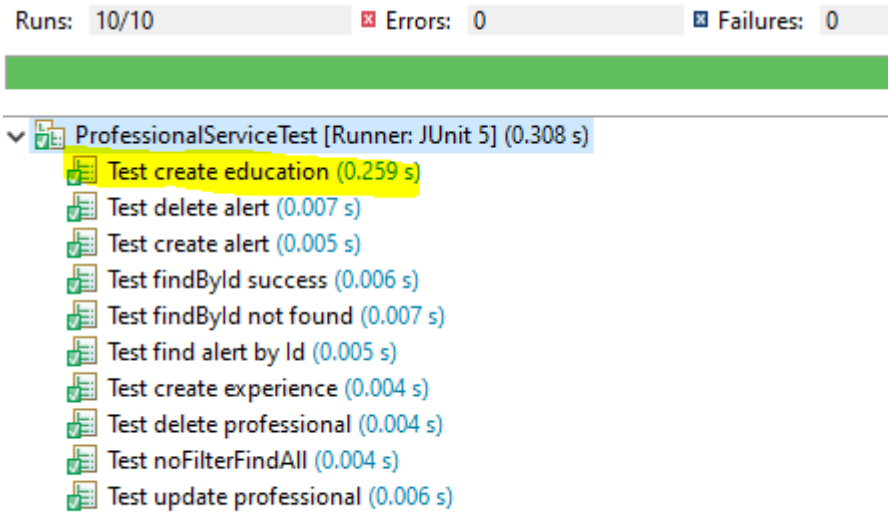
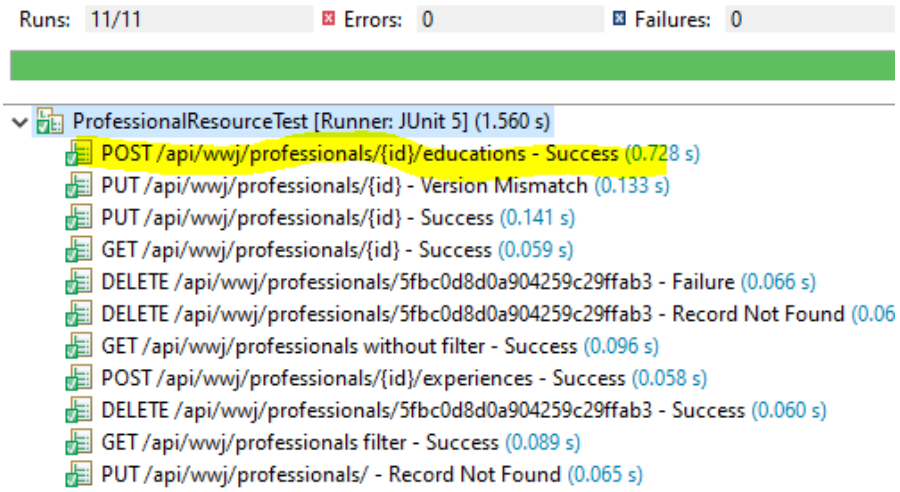
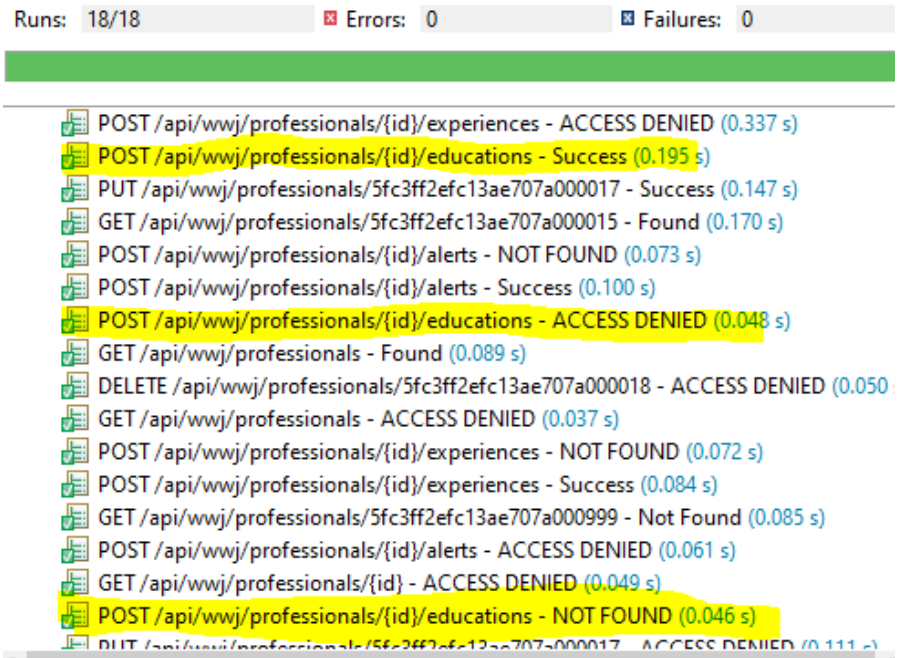
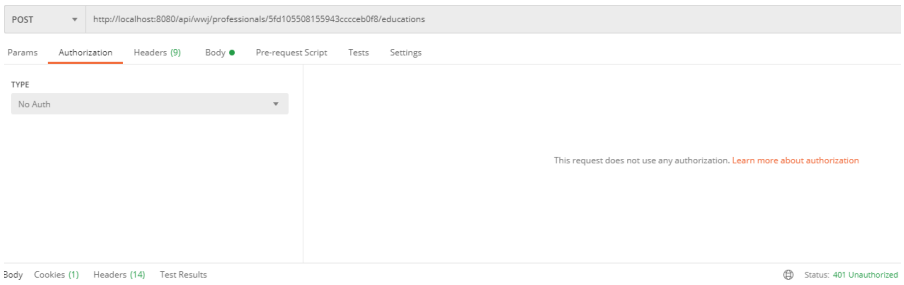
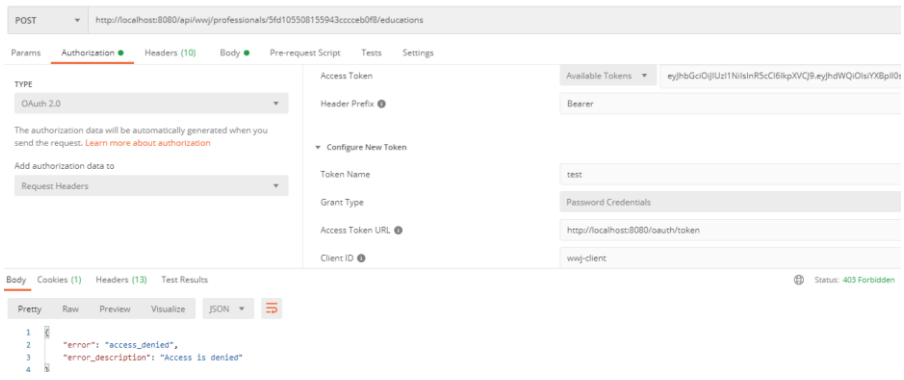
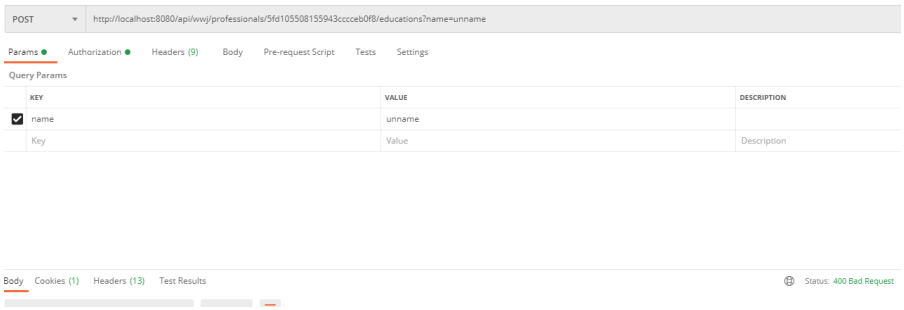
400 Bad Request	
201 Created	 Si ejecutamos la llamada autenticados como un profesional existente, nos devuelve un 201 OK y los datos de la alerta creada.
404 NOT FOUND	 Si el id del professional pasado como parámetro no se encuentra en el sistema

TABLA 19 – DETALLE TESTING HISTORIA DE USUARIO 003

US - 003	Cómo profesional quiero incluir mi formación académica en mi perfil para competir en el mercado laboral.
Test End Point - Junit	
POST	/professionals/{id}/educations
RepositoryTest	
	Verificamos que la capa de integración funciona correctamente a la hora de crear y eliminar educación, no se contempla la modificación de las mismas en esta primera versión.
ServiceTest	
	A nivel de servicio pretendo verificar la continuidad del requisito de usuario "Creación de formación académica"

ResourceTests	
	Verificamos que efectivamente haciedo POST a la url de educations con el profesional asociado el usuario recibe la respuesta es la esperada.
IntegrationTests	
	En los tests de integración comprobamos las respuestas esperadas para los casos más habituales.
Test End Point - Postman	
POST	/professionals/{id}/educations

401	 Si tratamos de crear una educación sin haber enviado petición a Oauth previamente y autenticarnos, el sistema devuelve error unauthorized
403	Si nos logamos como Company, el sistema devolverá un error de AccessDeniedException como observamos a continuación 
400 Bad Request	

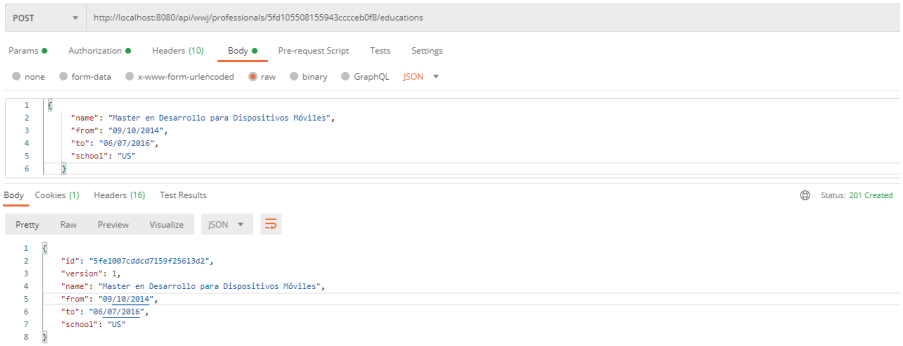
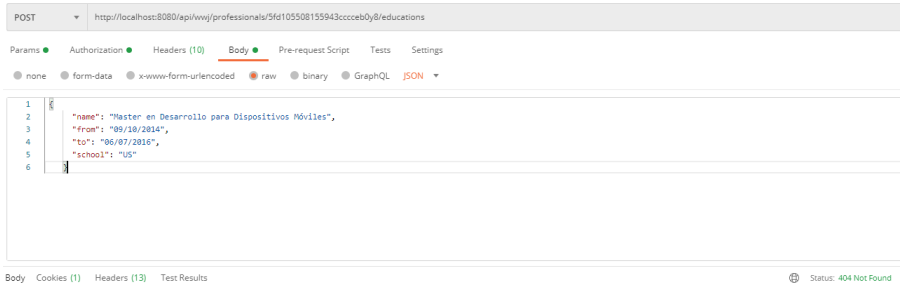
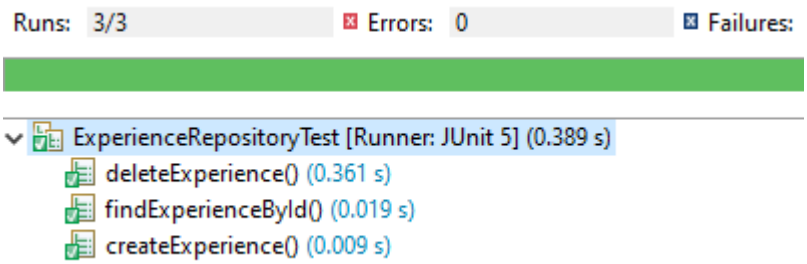
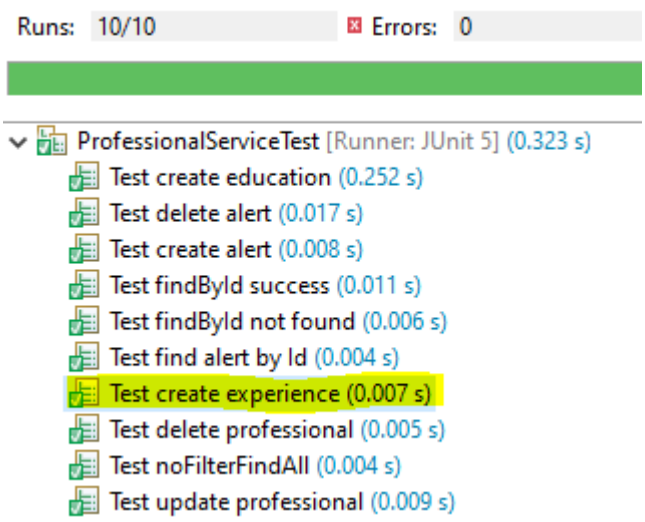
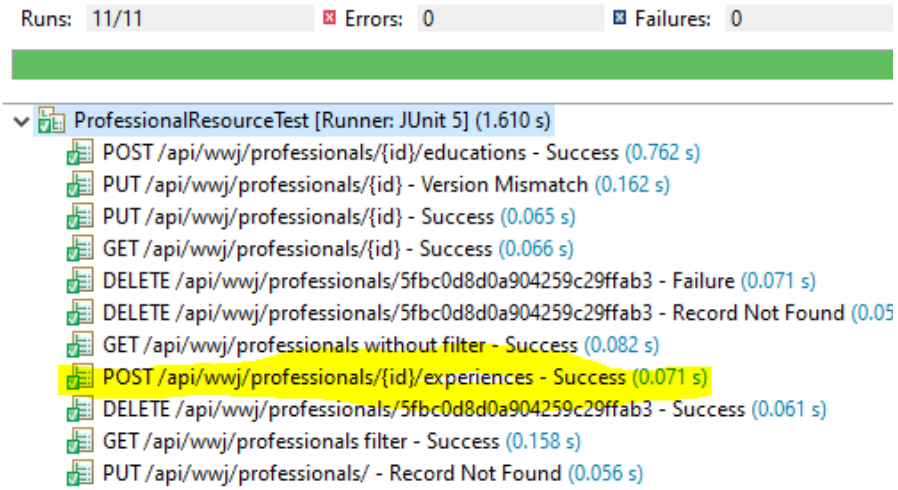
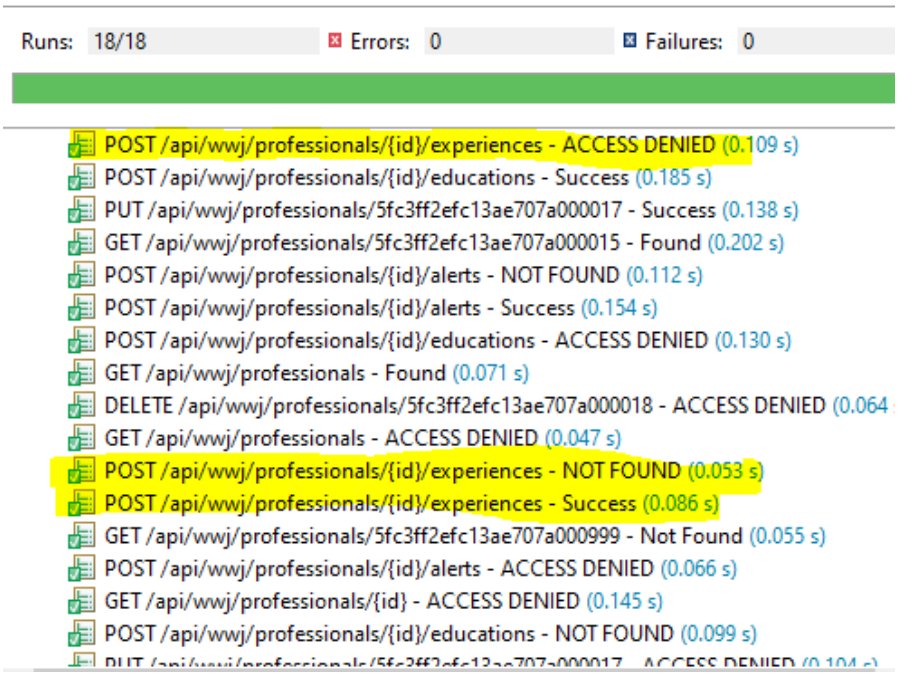
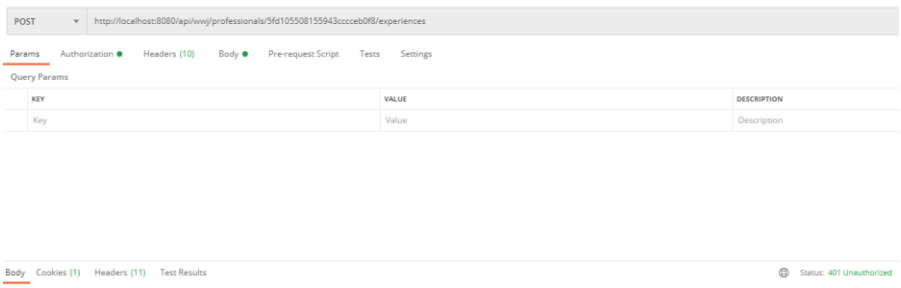
201 Created	 Si ejecutamos la llamada autenticados como un profesional existente, nos devuelve un 201 OK y los datos de la educación creada.
404 NOT FOUND	 Si el id del professional pasado como parámetro no se encuentra en el sistema

TABLA 20 – DETALLE TESTING HISTORIA DE USUARIO 004

US - 004	Cómo profesional quiero incluir mi experiencia profesional en mi perfil para competir en el mercado laboral.
Test End Point - Junit	
POST	/professionals/{id}/experiences
RepositoryTest	

	Verificamos que la capa de integración funciona correctamente a la hora de crear y eliminar experiencias, no se contempla la modificación de las mismas en esta primera versión.
ServiceTest	
	A nivel de servicio pretendo verificar la continuidad del requisito de usuario "Creación de experiencia profesional"
ResourceTests	
	Verificamos que efectivamente haciedo POST a la url de educaciones con el profesional asociado el usuario recibe la respuesta es la esperada.

IntegrationTests	 En los tests de integración comprobamos las respuestas esperadas para los casos más habituales.
Test End Point - Postman	
POST	/professionals/{id}/experiences
401	 Si tratamos de crear una experiencia sin haber enviado petición a Oauth previamente y autenticarnos, el sistema devuelve error unauthorized
403	Si nos logamos como Company, el sistema devolverá un error de AccessDeniedException como observamos a continuación

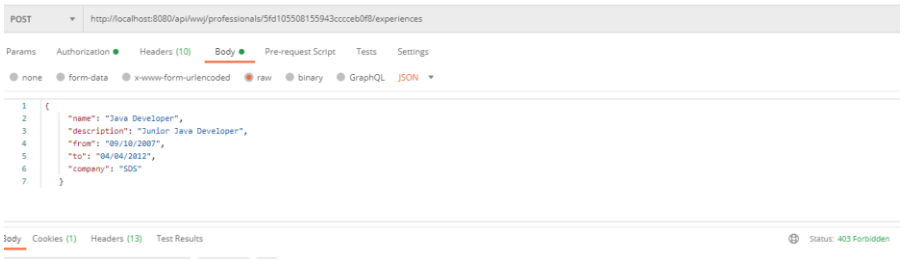
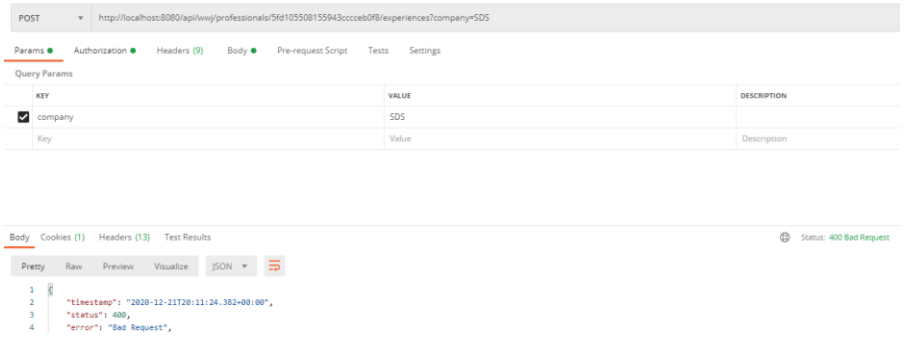
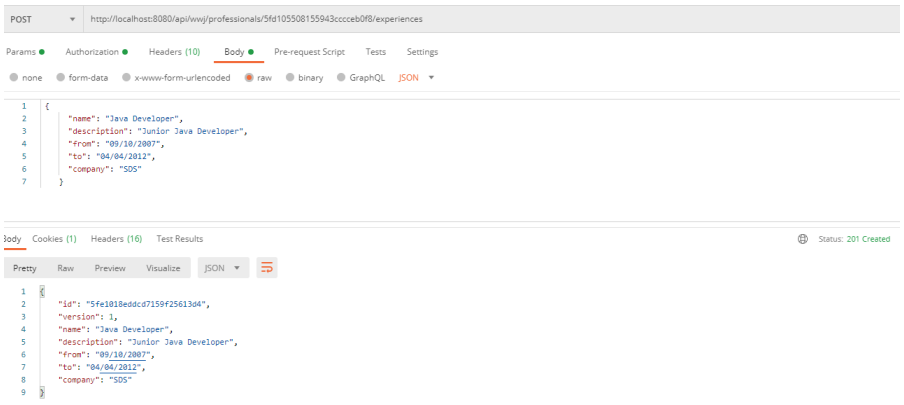
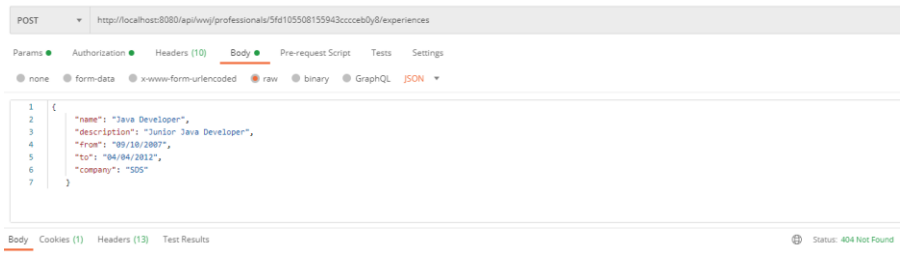
	
400 Bad Request	
201 Created	 Si ejecutamos la llamada autenticados como un profesional existente, nos devuelve un 201 OK y los datos de la alerta creada.
404 NOT FOUND	 Si el id del professional pasado como parámetro no se encuentra en el sistema

TABLA 21 – DETALLE TESTING HISTORIA DE USUARIO 005

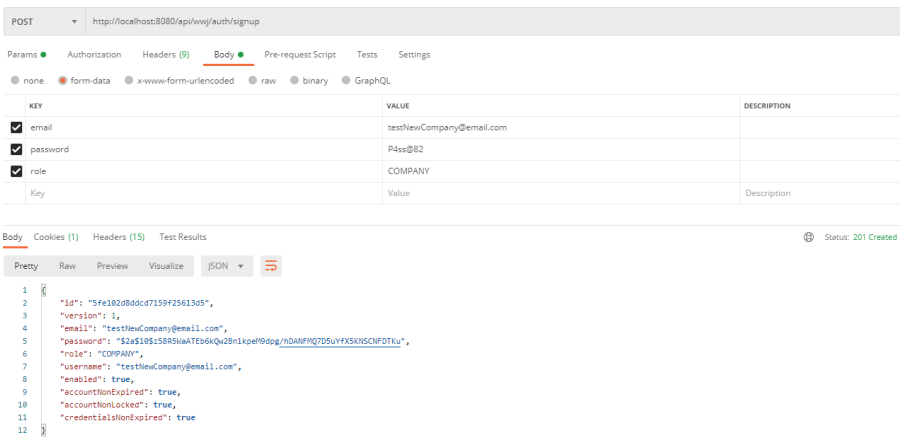
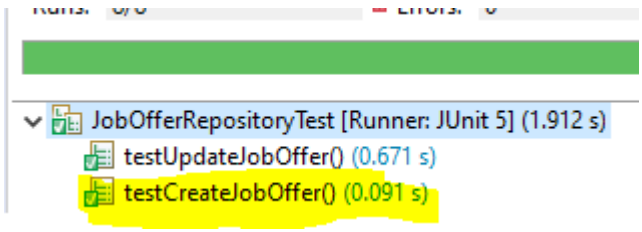
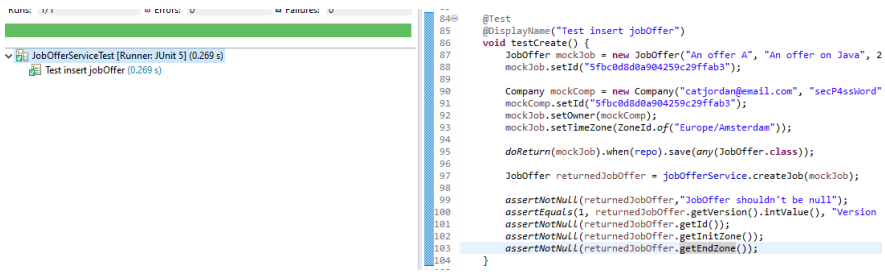
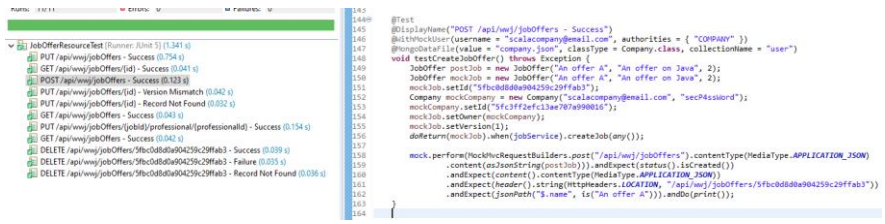
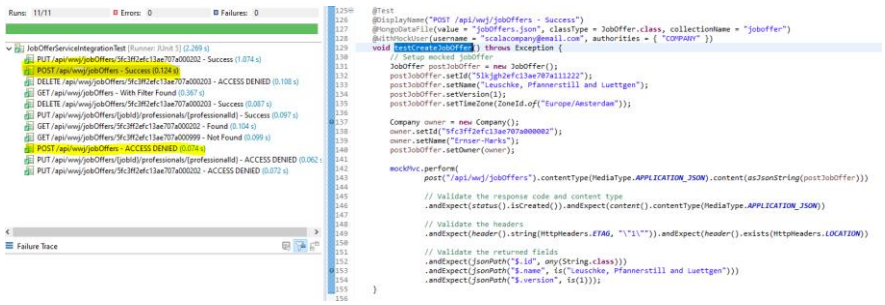
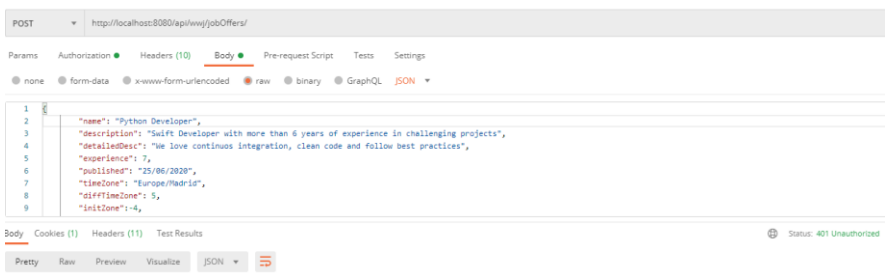
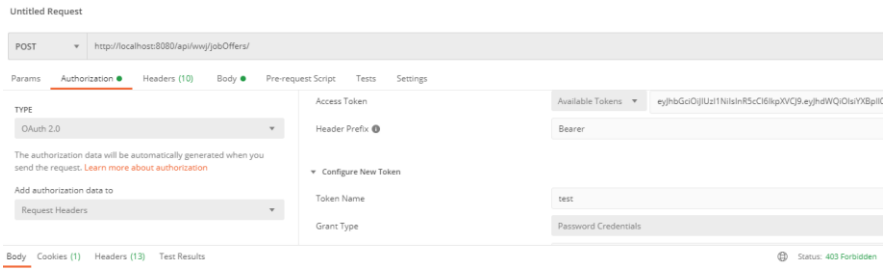
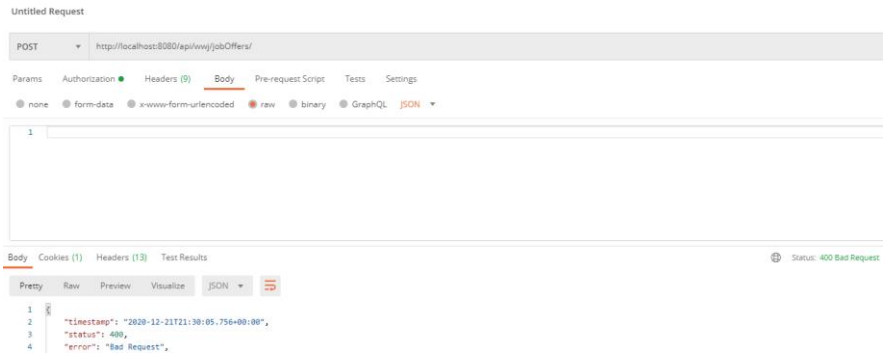
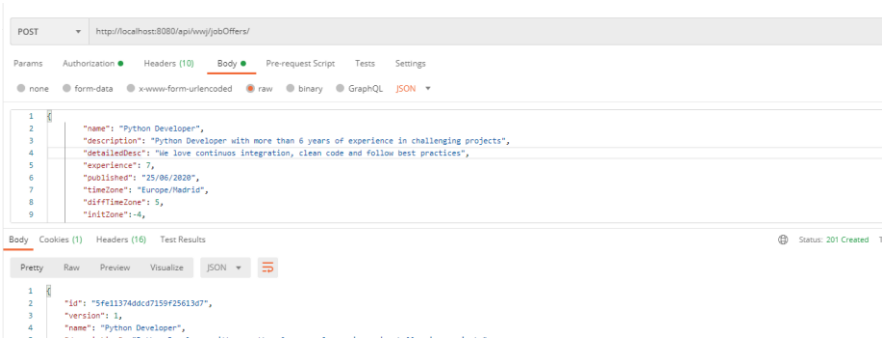
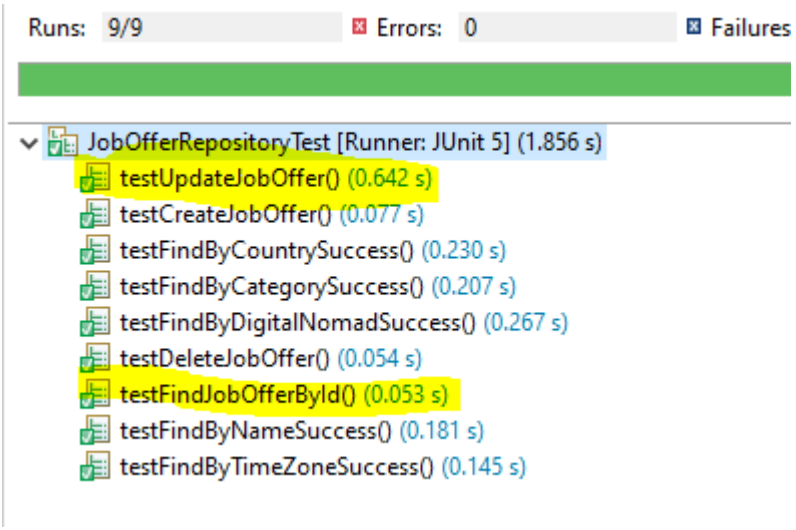
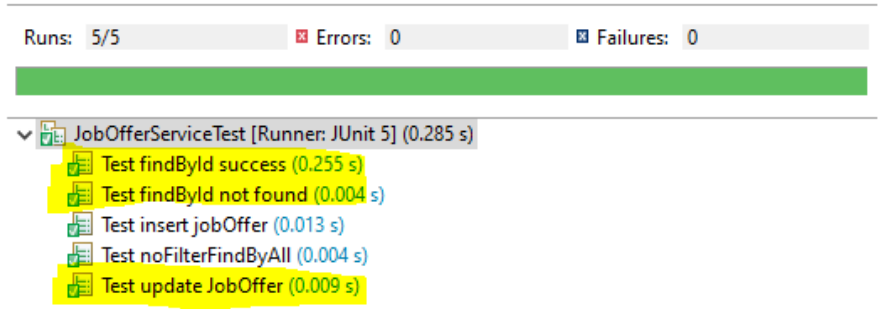
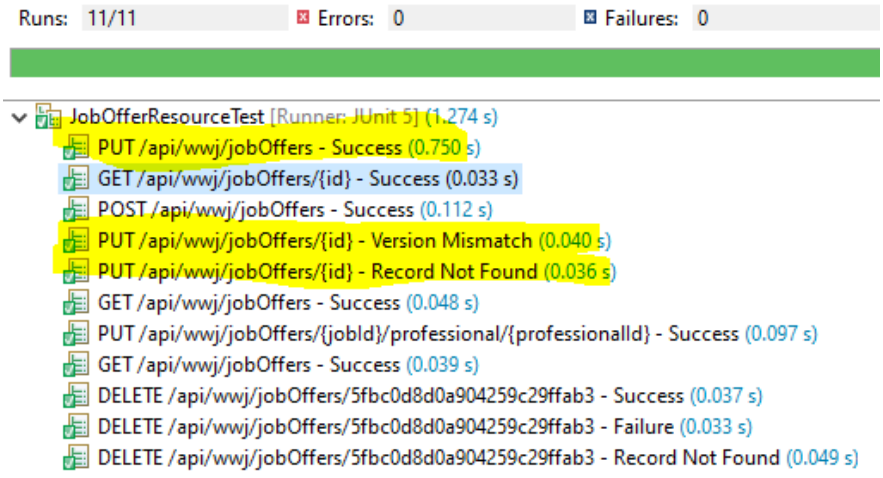
US - 005	Cómo profesional quiero registrarme en la plataforma para dar de alta mis datos personales y profesionales y estar visible en la plataforma.
Test End Point – Junit (Ver pruebas US-001)	
POST	/auth/signup
Test End Point - Postman	
POST	/auth/signup
201 Created	

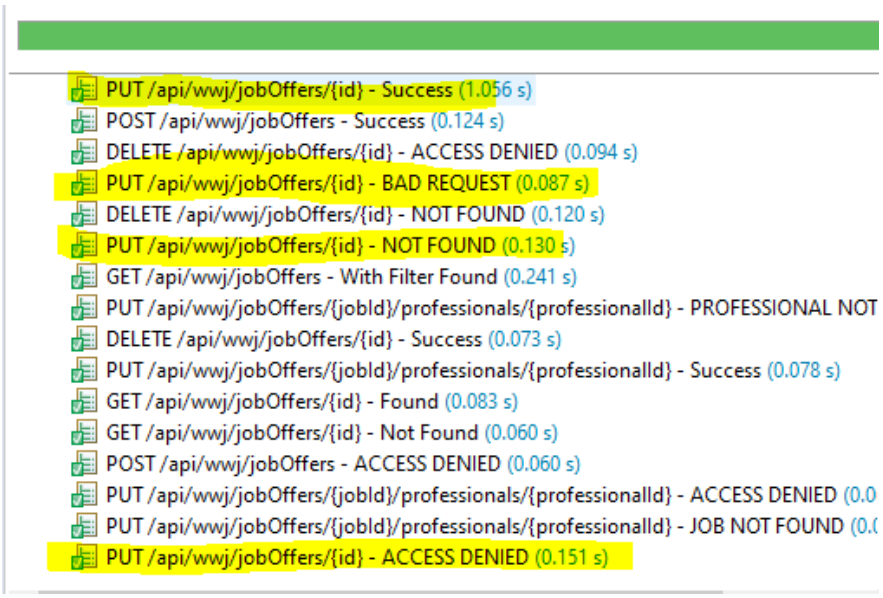
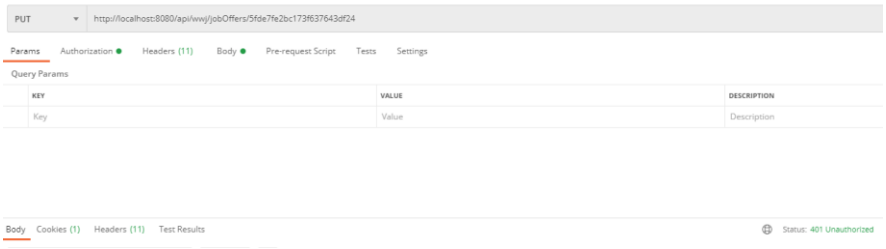
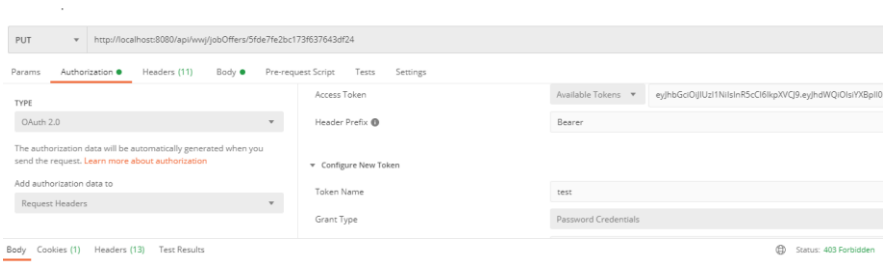
TABLA 22 – DETALLE TESTING HISTORIA DE USUARIO 006

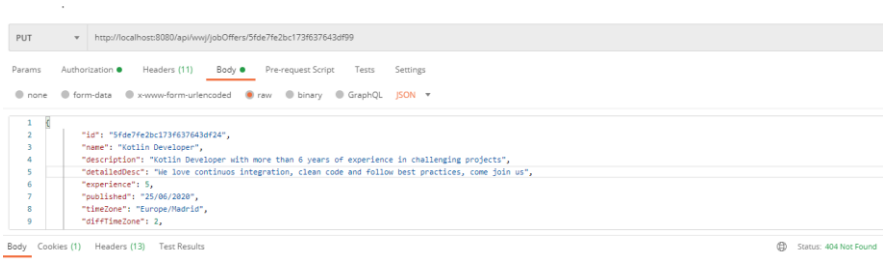
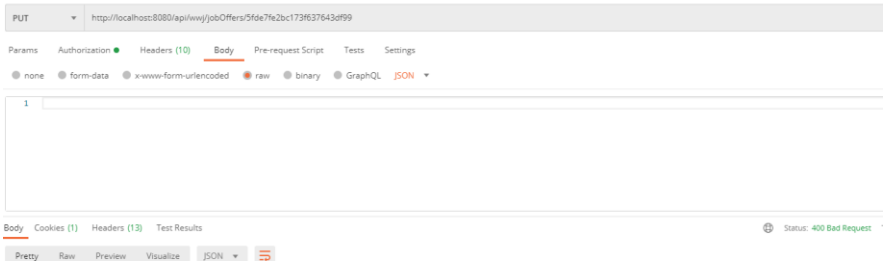
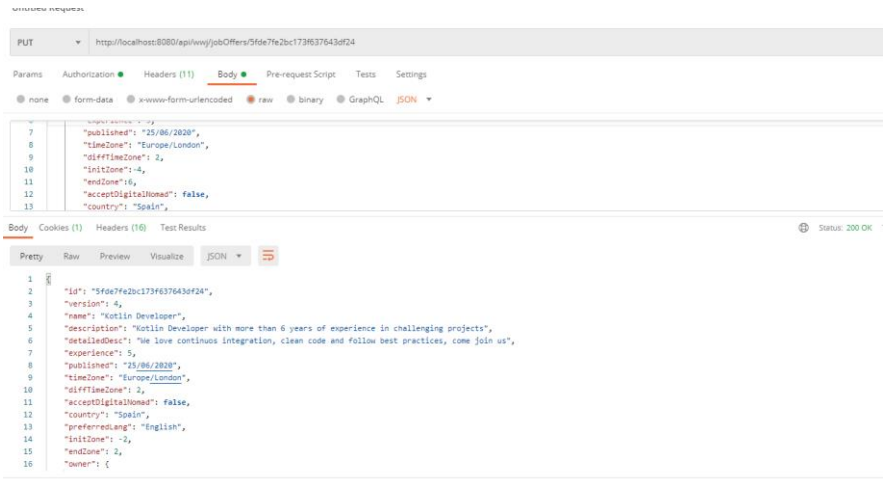
US - 006	Cómo empresa quiero gestionar mis ofertas en la plataforma para captar los mejores profesionales.
Test End Point - Junit	
POST	/jobOffers
Repository	

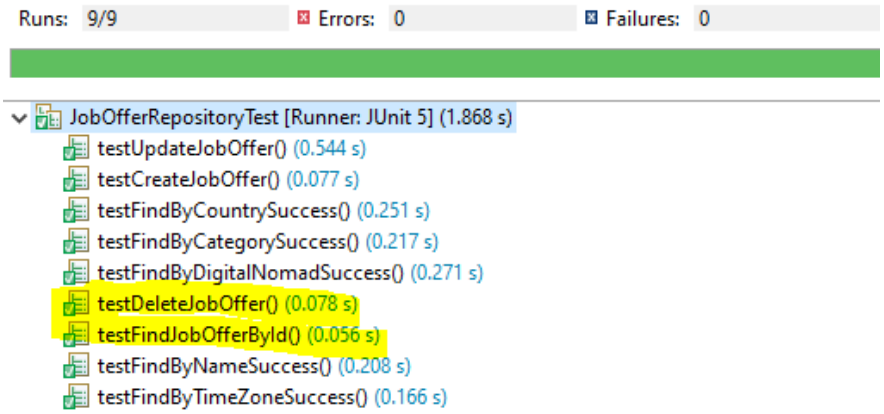
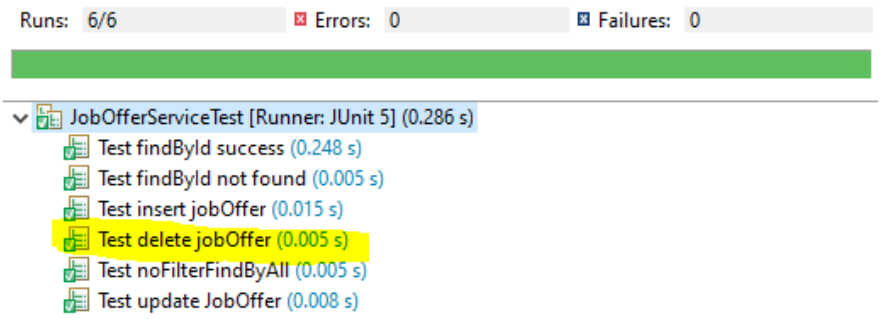
Service	 Se comprueba que además de la continuidad con la capa de persistencia, la lógica de servicio, que debe establecer parámetros como la franja horaria de inicio y fin en base a los datos aportados en la oferta se cumplimentan correctamente para que posteriormente las búsquedas sean correctas.
Resource	 Se verifica la continuidad a nivel de creación del recurso
Integration	 Se verifican las respuestas esperadas.
Test End Point - Postman	
401	

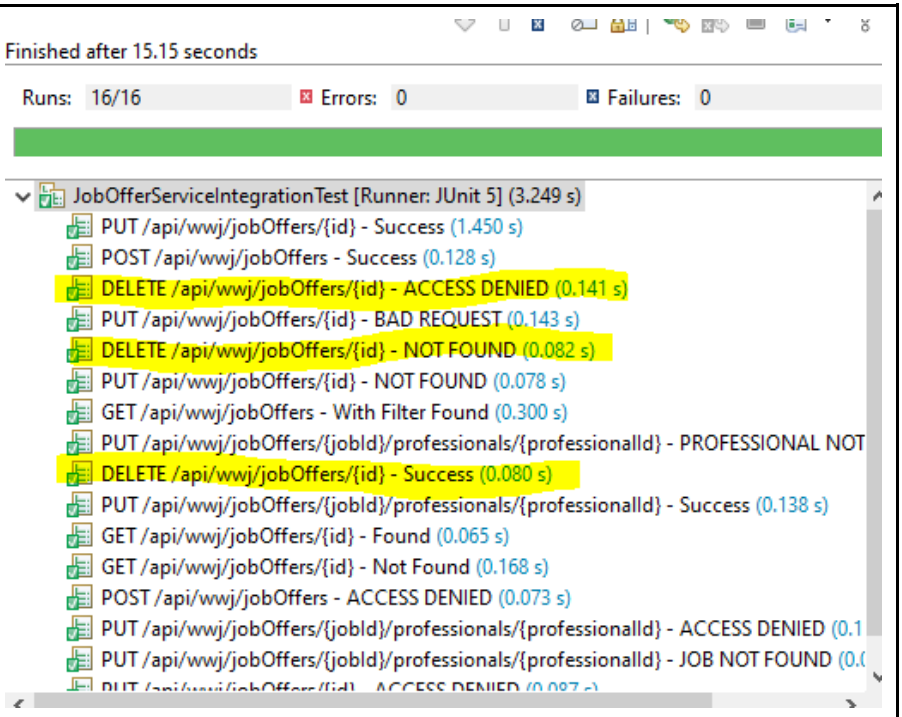
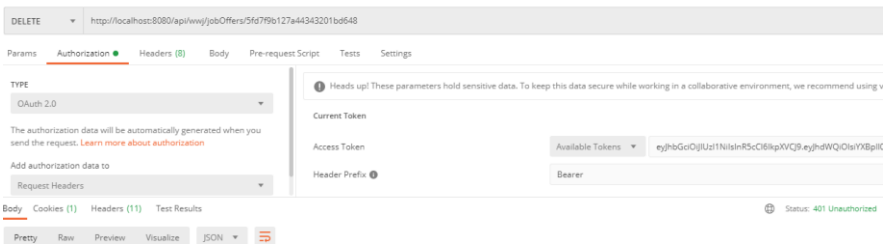
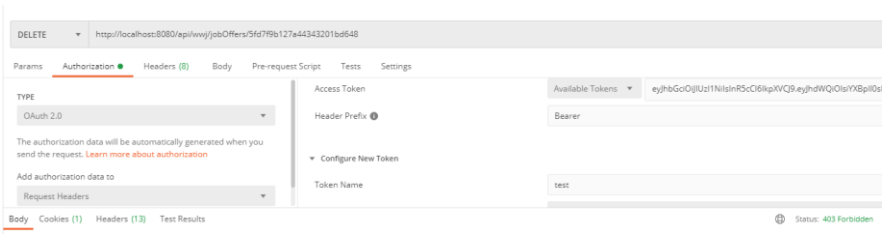
403	 Si me autentico como profesional e intento crear una oferta
400 Bad Request	
201 Created	 Si me autentico como empresa y creo una oferta con el formato de request correcto
PUT	/jobOffers/{id}

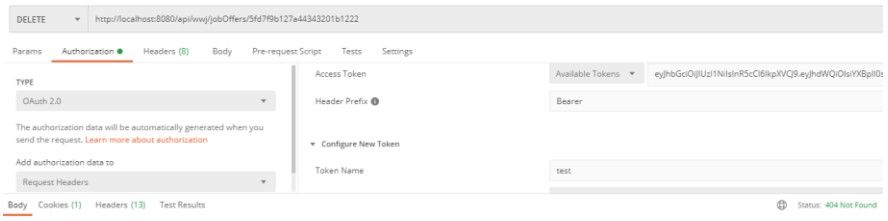
Repository	 Para cubrir esta funcionalidad necesitaremos recuperar la entidad por su ID para posteriormente actualizarla. Se han separado el update y create por funcionalidad si bien a nivel de repositorio la operación es la misma "save".
Service	 Se comprueba el servicio en caso de encontrar la entidad y en caso de no encontrarla y la actualización de la misma.
Resource	 Se verifican los casos más habituales, el correcto, el error de

	versionado y el recurso no encontrado
Integration	 Se verifican los casos esperados más habituales
Test End Point - Postman	
401	 Sin autenticar
403	 Autenticada como profesional

404	 Si pasamos un id de jobOffer inexistente
400	 Si enviamos una request inválida (sin body por ejemplo)
200 OK	 Comprobamos la correcta ejecución y reajuste de las franjas horarias según el nuevo time zone de la oferta
DELETE	/jobOffers/{id}

Repository	
Service	
Resource	 Se comprueban los casos más habituales de éxito, fallo y registro no encontrado

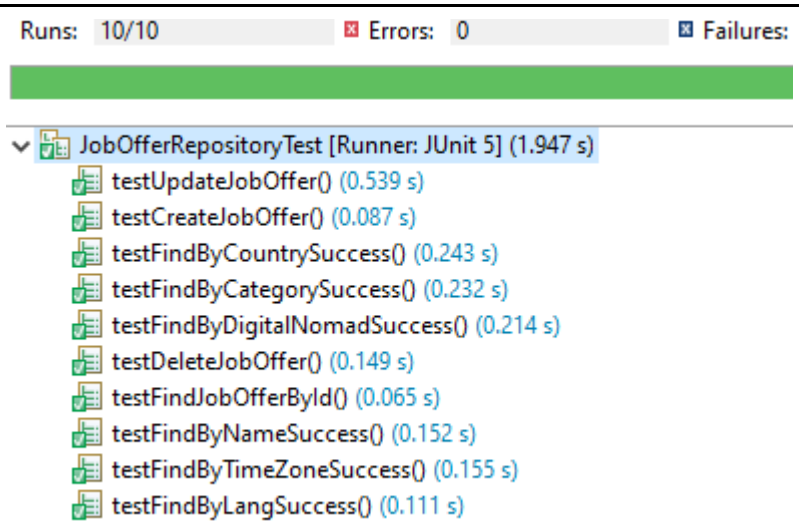
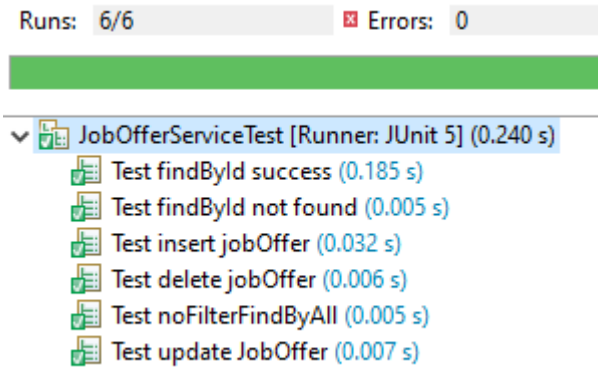
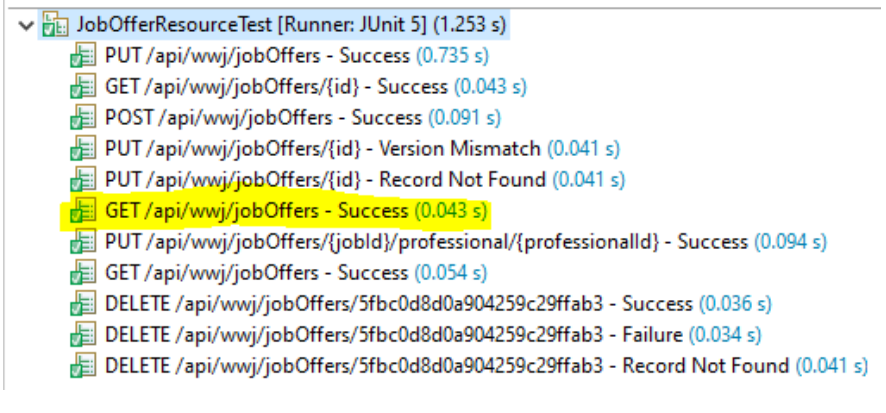
Integration	 Se comprueban las respuestas más esperadas.
Test End Point - Postman	
401	 Sin autenticar
403	 Autenticada como profesional

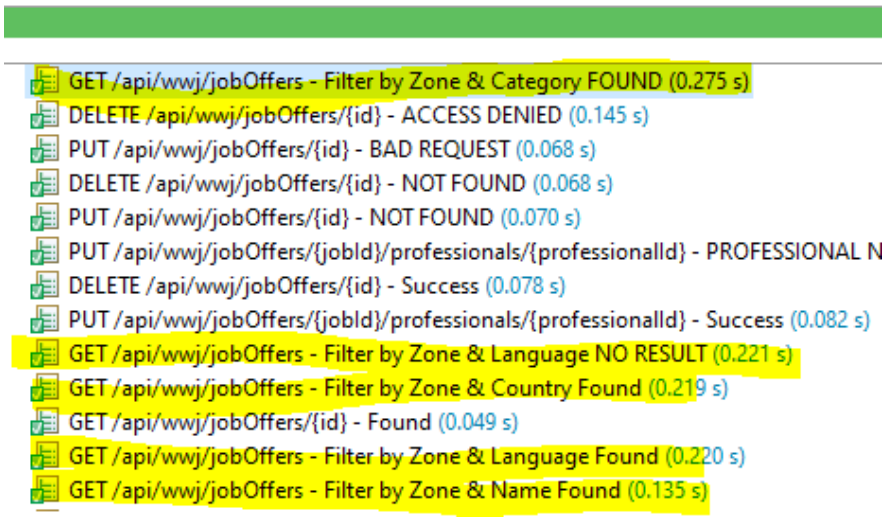
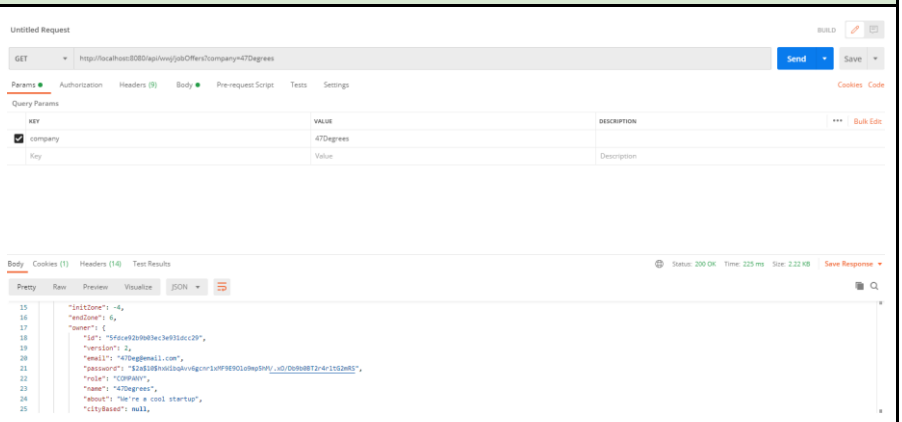
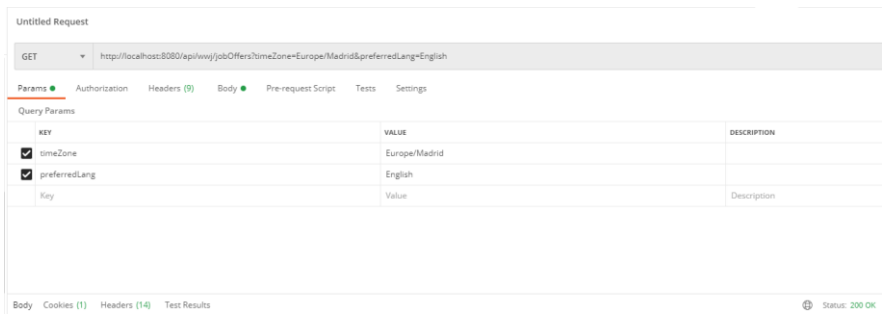
404	 Id de JobOffer inexistente
200 OK	 Cuando pasamos un id de oferta existente

Mención aparte merecen las búsquedas, en ellas se basa la idea principal de este proyecto. Se han implementado métodos de búsqueda específicos a nivel de repositorio que puedan actuar como métodos auxiliares de ser necesarios, si bien la búsqueda se basa en una implementación que combina diferentes criterios según los parámetros recibidos para la búsqueda utilizando mongotemplate. Si la búsqueda no recibe ningún parámetro de filtro devuelve todas las ofertas (findAll del repositorio).

TABLA 23 – DETALLE TESTING HISTORIA DE USUARIO 007

US - 007	Cómo usuario quiero realizar búsquedas de ofertas de empleo estableciendo franja horaria, idioma y/o ubicación para encontrar el trabajo deseado. Adicionalmente se ha implementado la búsqueda por categoría, empresa, pais, nombre, descripción y perfiles nómadas digitales.
Test End Point - Junit	
GET	/jobOffers

Repository	
Service	
Resource	

Integration	 En esta captura se presentan algunas de las combinaciones de filtros testeadas pero las posibilidades son grandes y con postman se puede probar cualquier combinación que se nos ocurra y contrastar con la BD real.
Test End Point - Postman	
200 OK	 Filtrado por compañía
200 OK	 Por zona e idioma

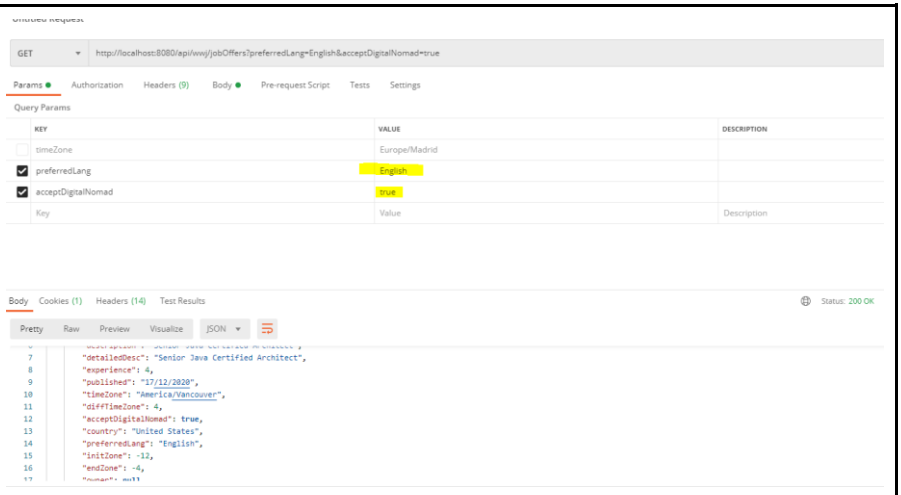
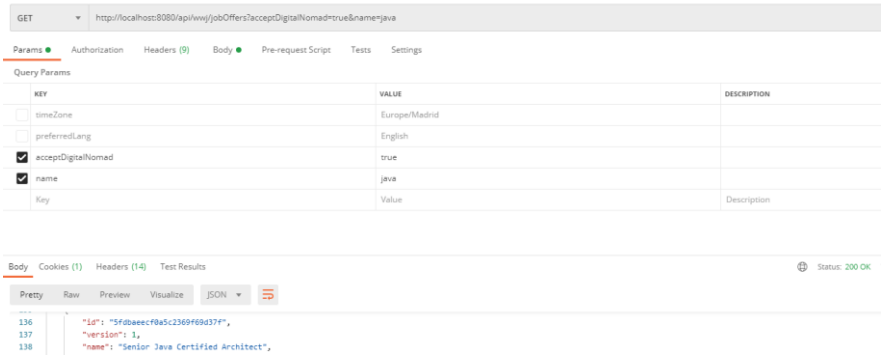
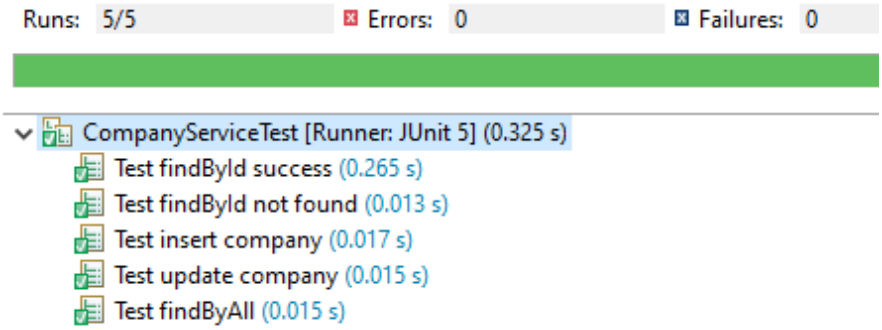
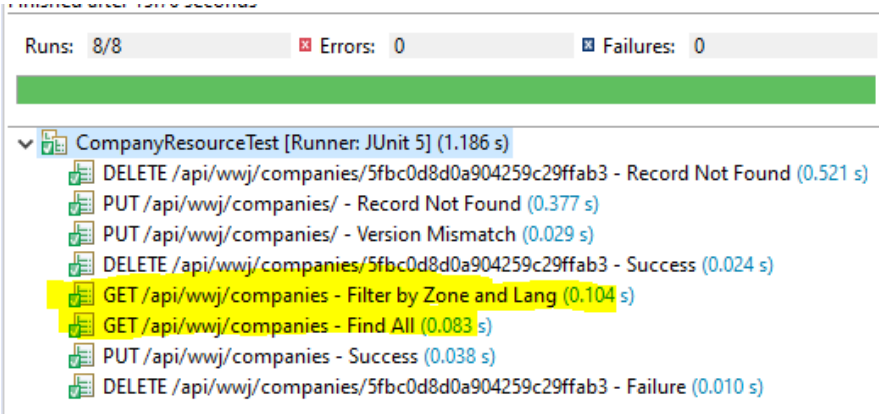
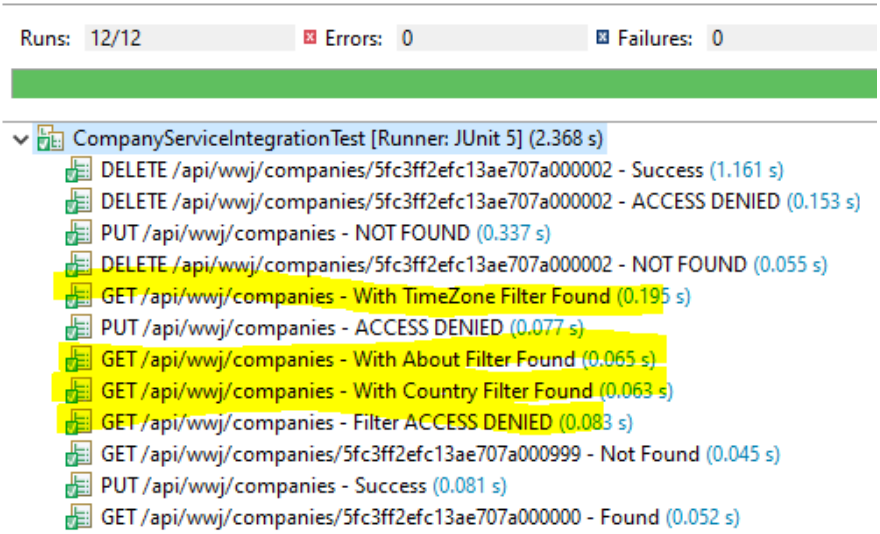
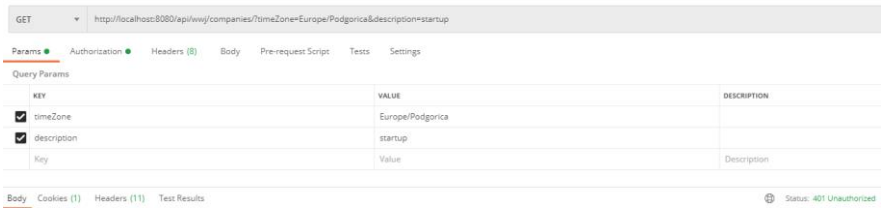
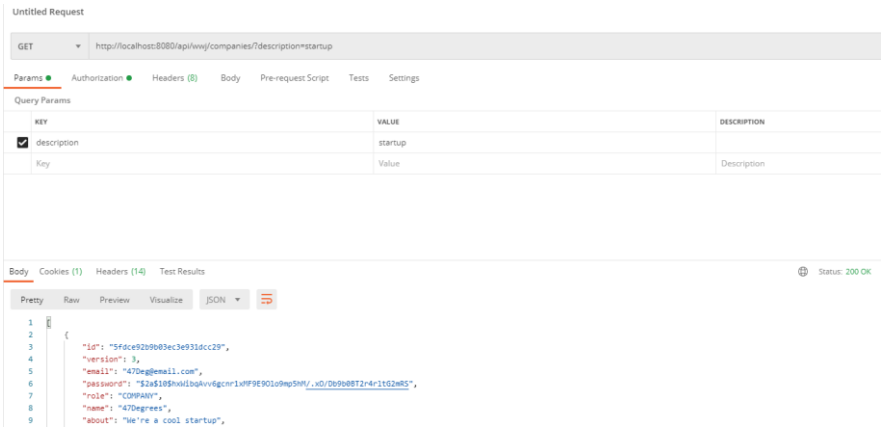
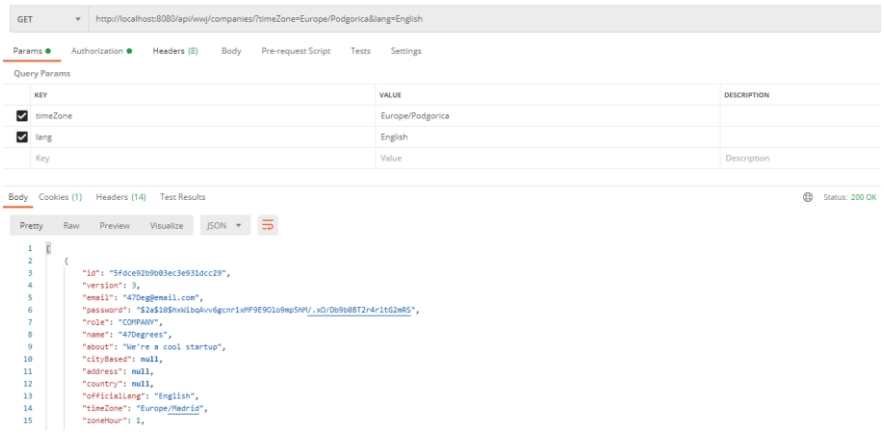
200 OK	 Por idioma y nómadas digitales
200 OK	 Por zona horaria y nombre de la oferta case insensitive

TABLA 24 – DETALLE TESTING HISTORIA DE USUARIO 008

US - 008	Cómo usuario quiero realizar búsquedas de empresas estableciendo la zona horaria, idioma o ubicación
Test End Point - Junit	
GET	/companies
Repository	A nivel de repositorio se verifican las pruebas en US-001 (update y get) puesto que el filtrado se implementa a nivel de servicio no se requieren mas tests a este nivel para US-008

Service	
Resource	
Integration	 En esta captura se presentan algunas de las combinaciones de filtros testeadas pero las posibilidades son grandes y con postman se puede probar cualquier combinación que se nos ocurra y contrastar con la BD real.
Test End Point - Postman	

401	
403	No se contempla el 403 porque todo usuario registrado puede ejecutar búsquedas de empresas
200 OK	 Filtrado por descripción
200 OK	 Por zona UTC+1 e idioma

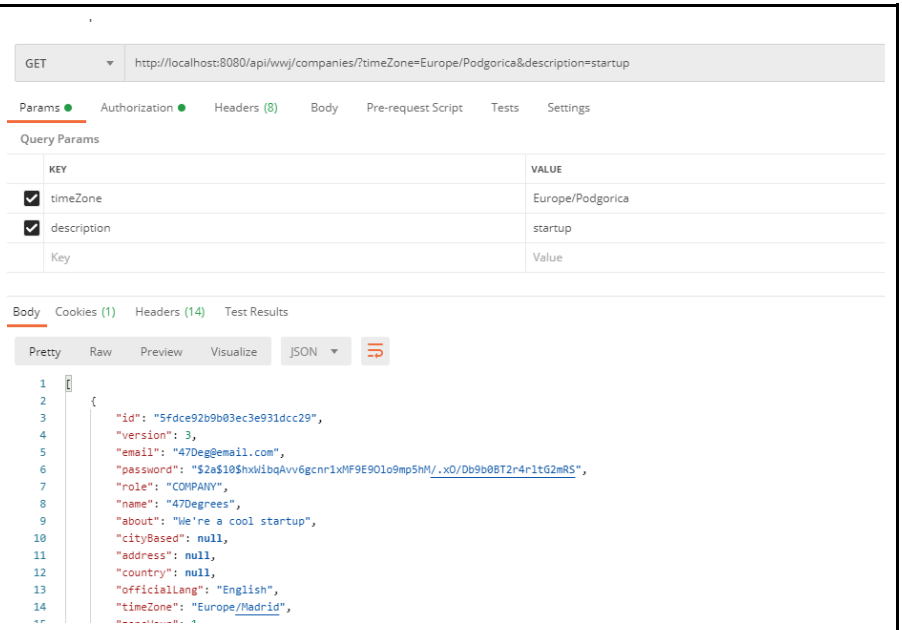
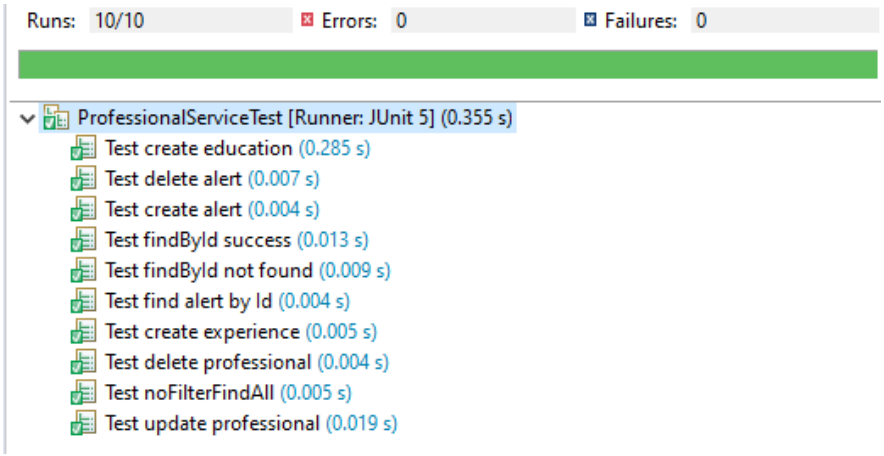
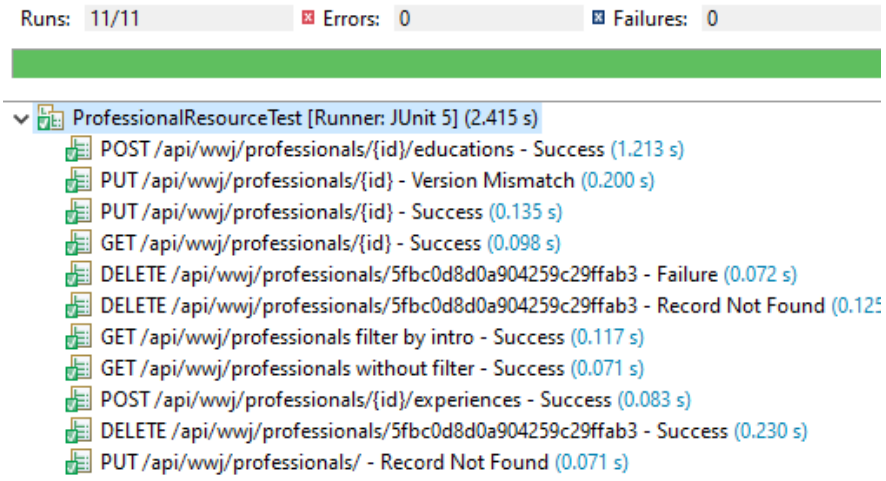
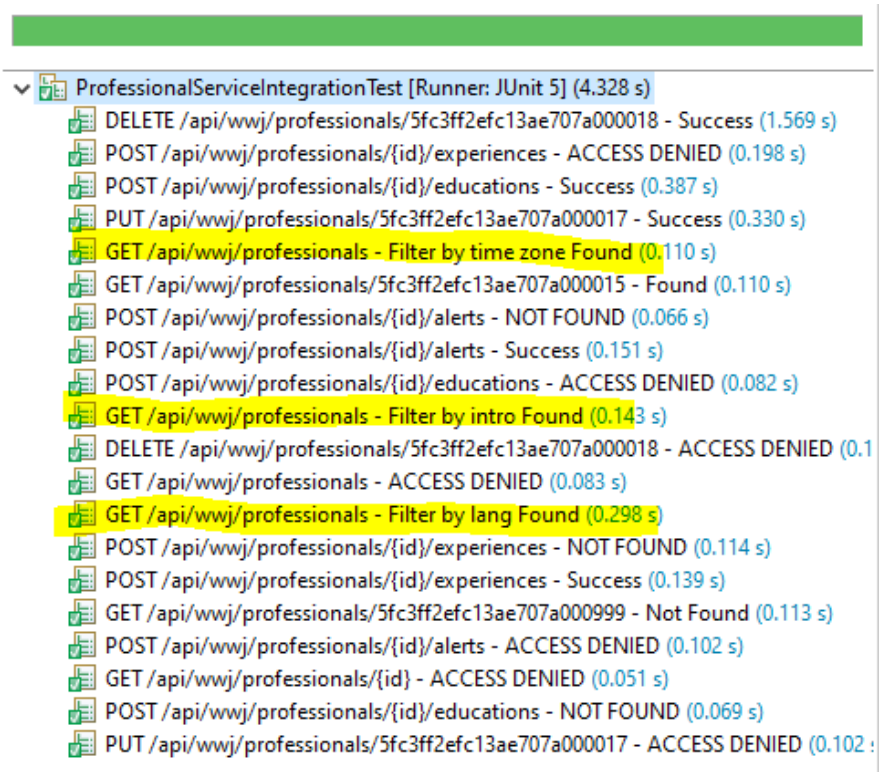
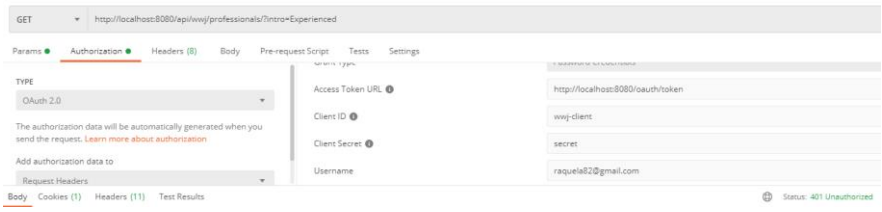
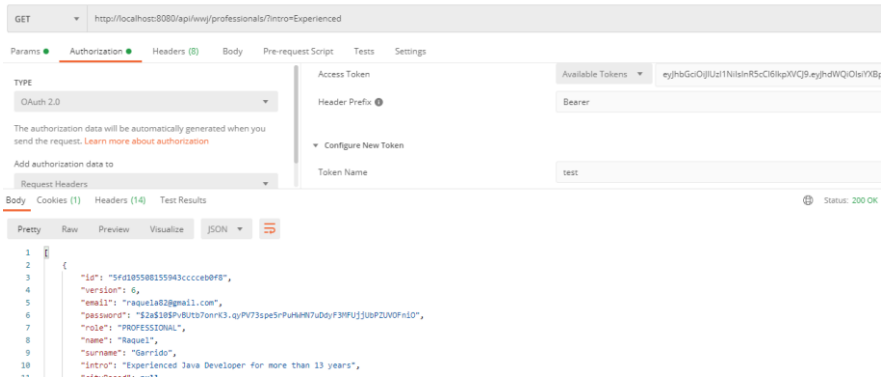
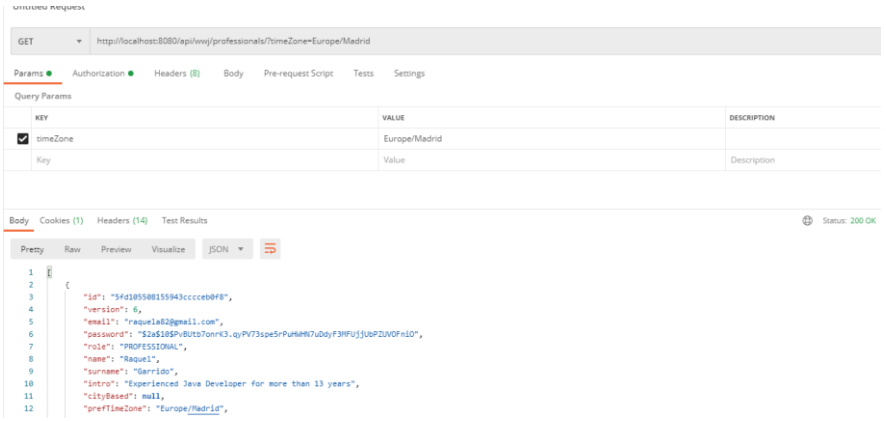
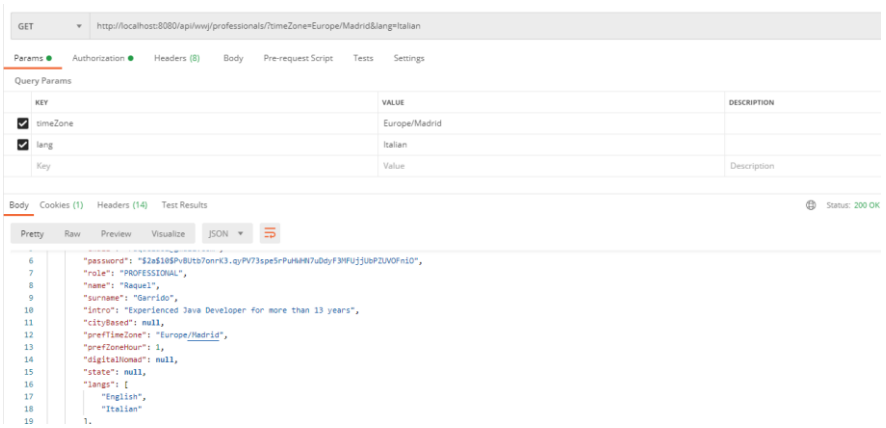
200 OK	 Por zona horaria y descripcion de la empresa case insensitive

TABLA 25 – DETALLE TESTING HISTORIA DE USUARIO 009

US - 009	Cómo usuario quiero realizar búsquedas de profesionales estableciendo los idiomas, la franja horaria y/o la ubicación para captar profesionales o comprobar las tendencias del sector.
Test End Point - Junit	
GET	/professionals
Repository	A nivel de repositorio se verifican las pruebas en US-001 (update y get) puesto que el filtrado se implementa a nivel de servicio no se requieren mas tests a este nivel para US-009

Service	 A nivel de servicio tenemos los métodos básicos que necesitamos
Resource	

Integration	 En esta captura se presentan algunas de las combinaciones de filtros testeadas pero las posibilidades son grandes y con postman se puede probar cualquier combinación que se nos ocurra y contrastar con la BD real.
Test End Point - Postman	
401	
403	No se contempla el 403 porque todo usuario autenticado puede realizar búsquedas de profesionales

200 OK	 Filtrado por introducción
200 OK	 Por idioma
200 OK	 Por idioma y zona

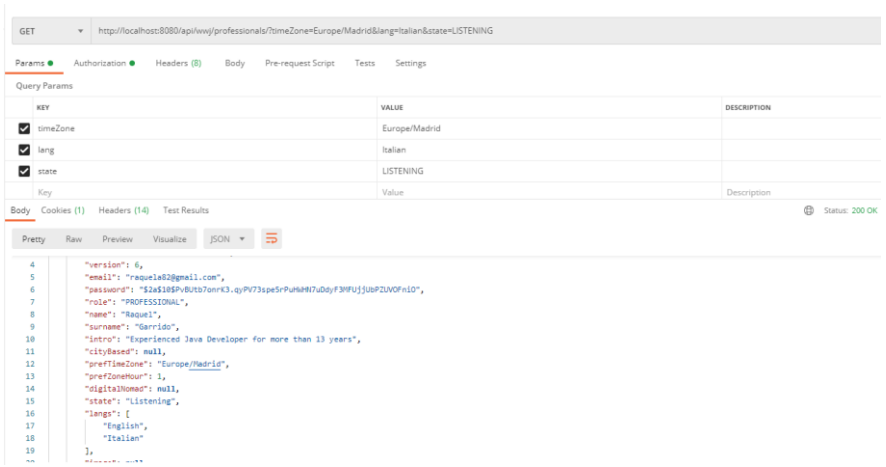
200 OK	 Por zona horaria, idioma y estado de disponiblidad

TABLA 26 – DETALLE TESTING HISTORIA DE USUARIO 010

US - 010	Cómo usuario quiero consultar todas las ofertas de una categoría determinada para estar al día
Test End Point - Junit	
GET	/jobOffers
Repository	A nivel de repositorio el conjunto de pruebas es el mismo que para US-007
Service	A nivel de servicio el conjunto de pruebas es el mismo que para US-007
Resource	A nivel de controller el conjunto de pruebas es el mismo que para US-007

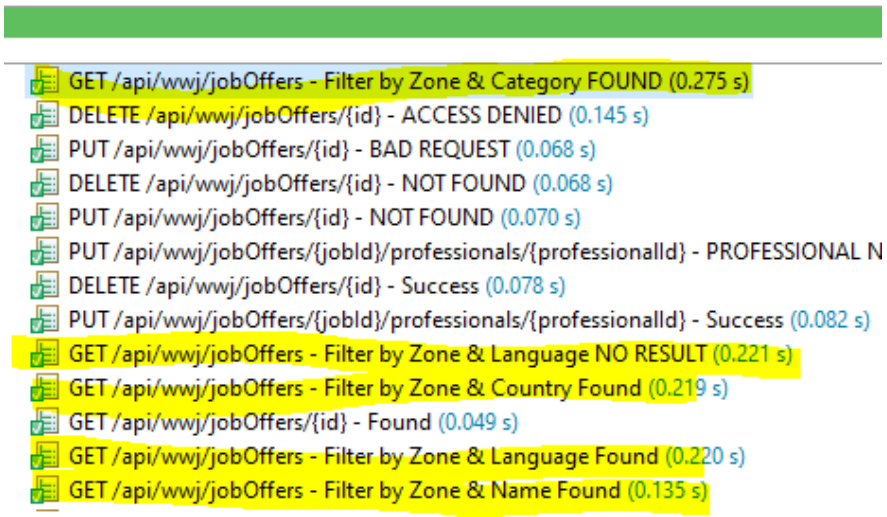
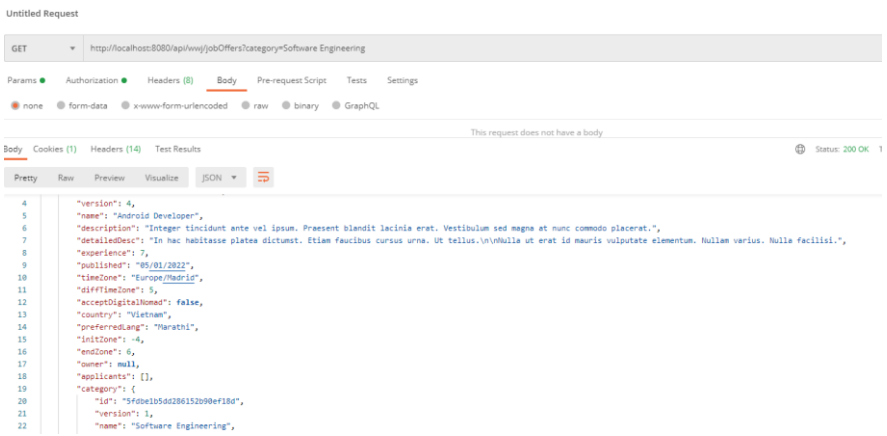
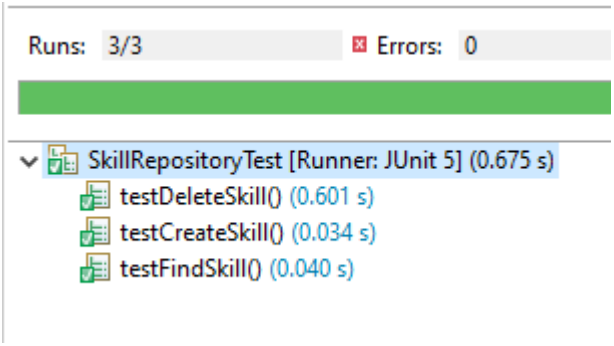
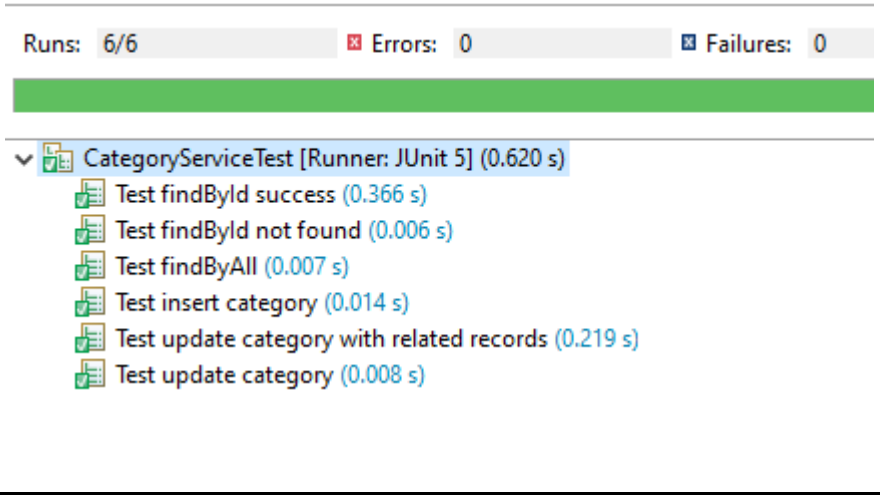
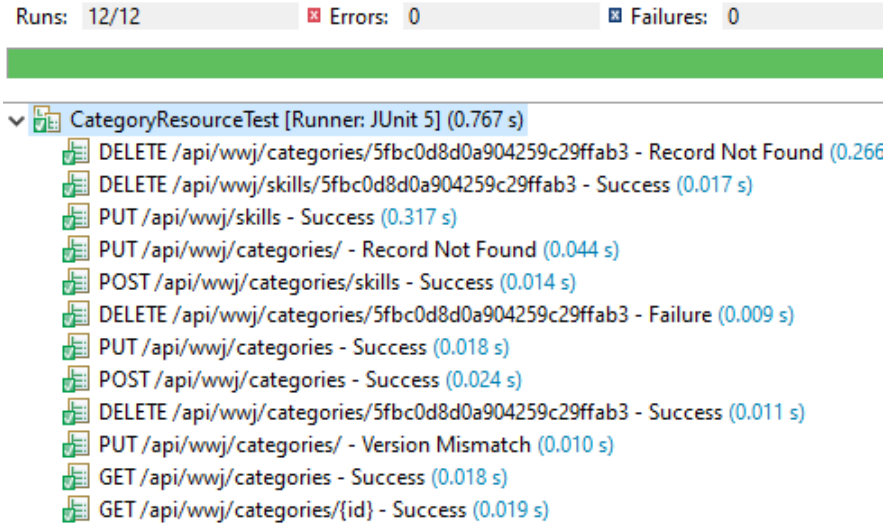
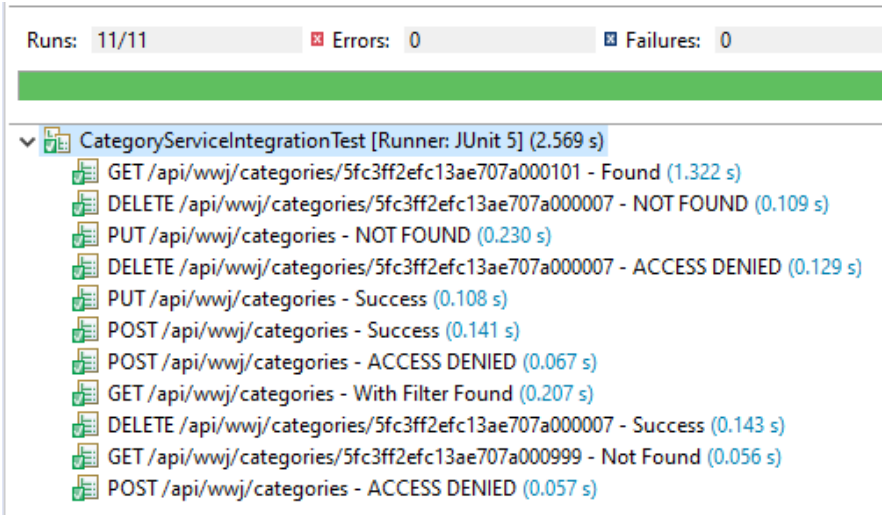
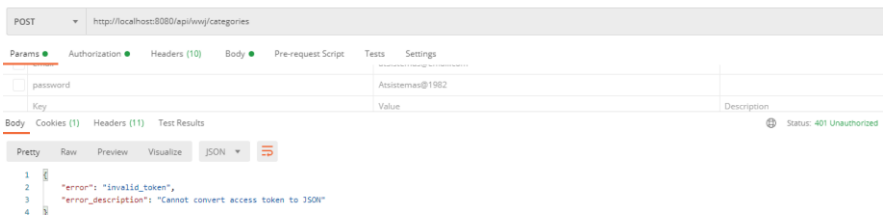
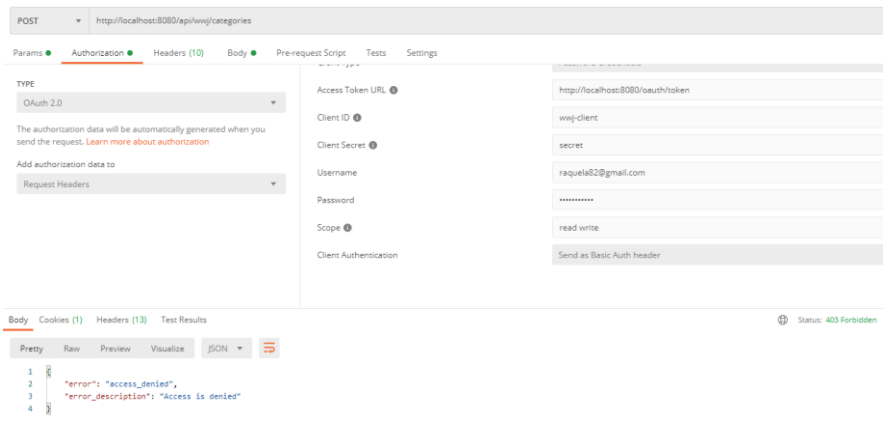
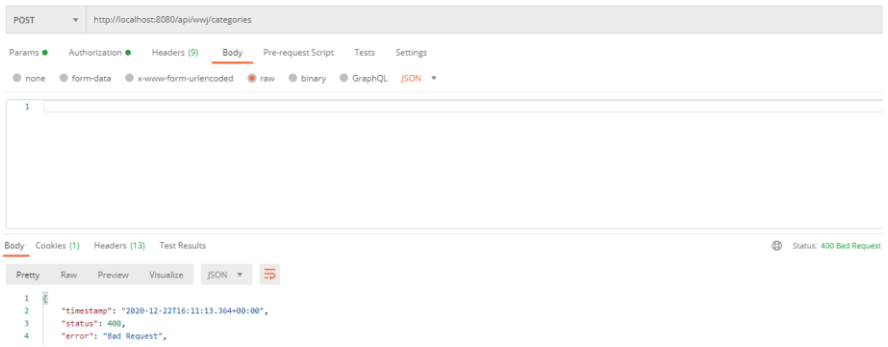
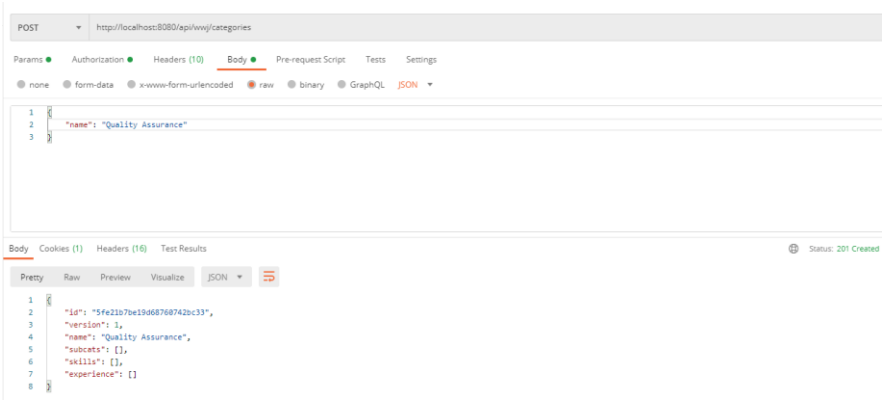
Integration	 En esta captura se presentan algunas de las combinaciones de filtros testeadas en US-007 incluyendo el filtrado por zona
Test End Point - Postman	
200 OK	 Filtrado por categoría

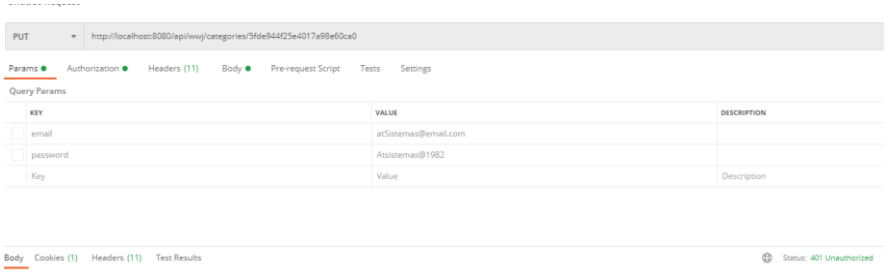
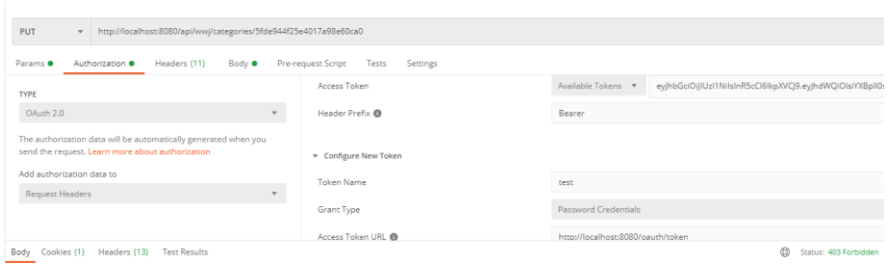
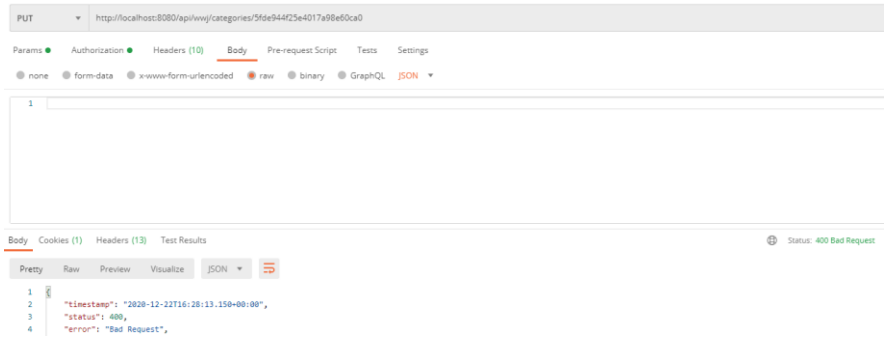
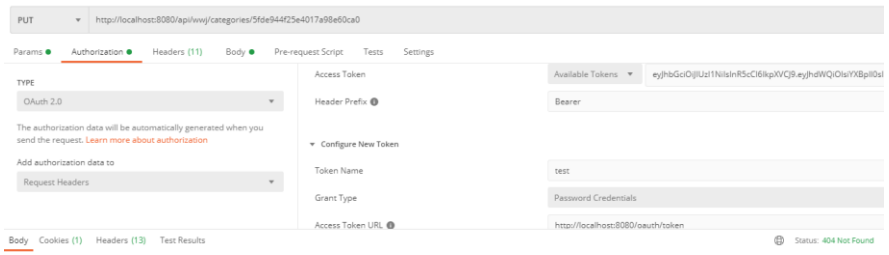
TABLA 27 – DETALLE TESTING HISTORIA DE USUARIO 011

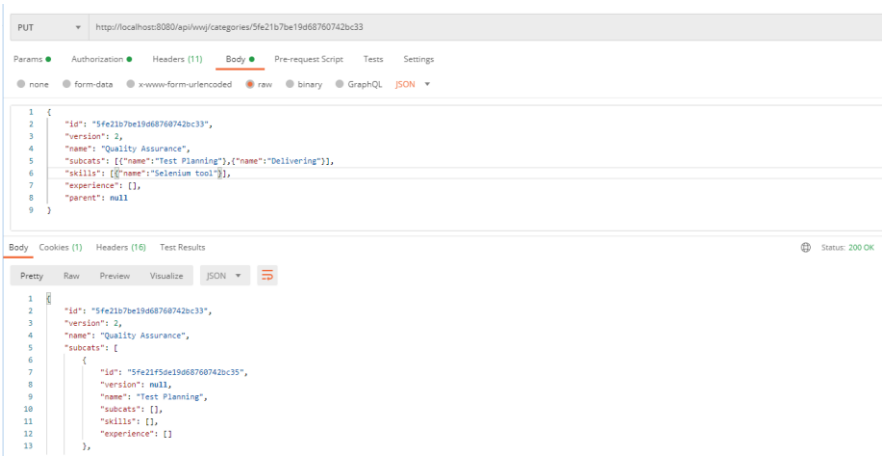
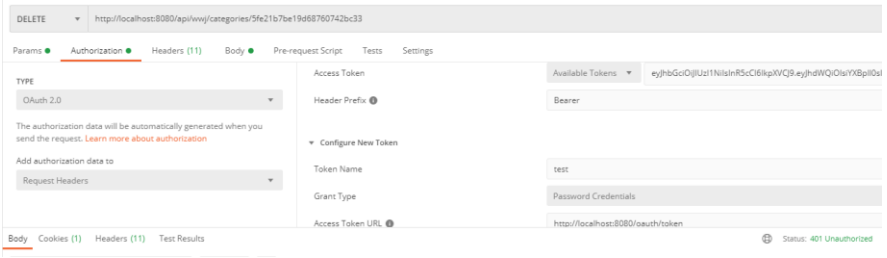
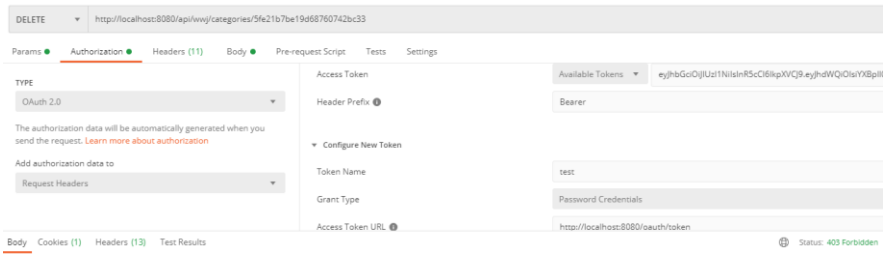
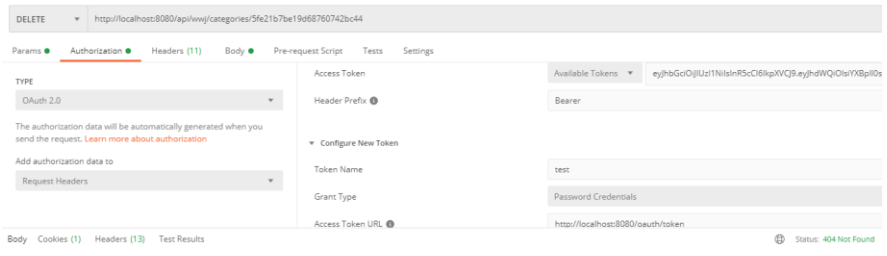
US - 011	Cómo administrador quiero gestionar categorías, subcategorías y habilidades para tener una clasificación de ofertas consistentes.
Test End Point - Junit	
POST	/categories

Repository	
Service	
Resource	

Integration	
PUT	/categories/{id}
	Se indican los casos verificados en el detalle del POST
DELETE	/categories/{id}
	Se indican los casos verificados en el detalle del POST
Test End Point - Postman	
POST	/categories/
401	 Si no me autentico

403	 Si me autentico como empresa o profesional
400 Bad Request	
201 Created	 Si me autentico como admin y creo una categoría con el formato de request correcto
PUT	/categories/{id}

401	 Sin autenticar
403	 Autenticada como profesional o empresa
400	 Se verifican los casos más habituales, el correcto, el error de versionado y el recurso no encontrado
404	 Si paso un identificador inexistente

200 ok	 Actualizamos la categoría creada en POST modificando su nombre y añadiéndole registros relacionados
DELETE	/categories/{id}
401	 Si no me autentico
403	 Si me autentico como empresa o profesional
404	 Si pasamos un id de categoría inexistente

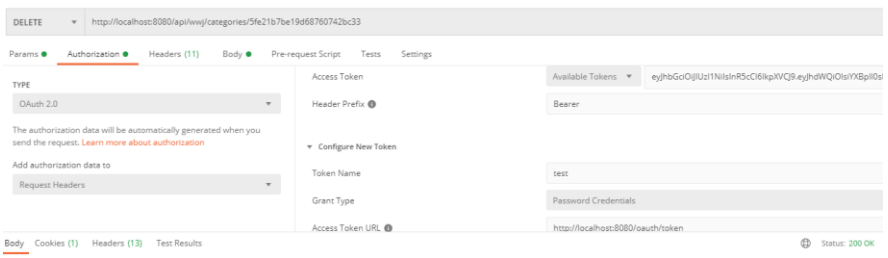
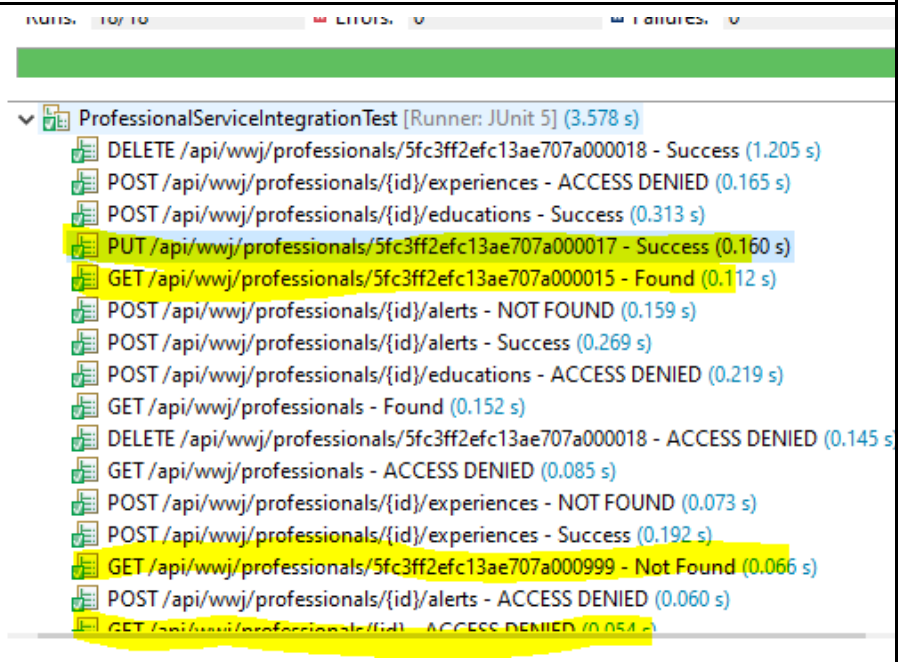
200 OK	 Comprobamos la correcta ejecución y borrado de la categoría en base de datos

TABLA 28 – DETALLE TESTING HISTORIA DE USUARIO 012

US - 012	Cómo administrador quiero poder desactivar perfiles profesionales y de compañías si considero que son inadecuados (descripciones fuera de lugar, etc…) para que no aparezcan en las búsquedas ni listados
Test End Point - Junit	
PUT	/companies/{id}
PUT	/professionals/{id}
	Los casos de prueba para este requisito se detallan en los PUT de US-001
Profesionales	

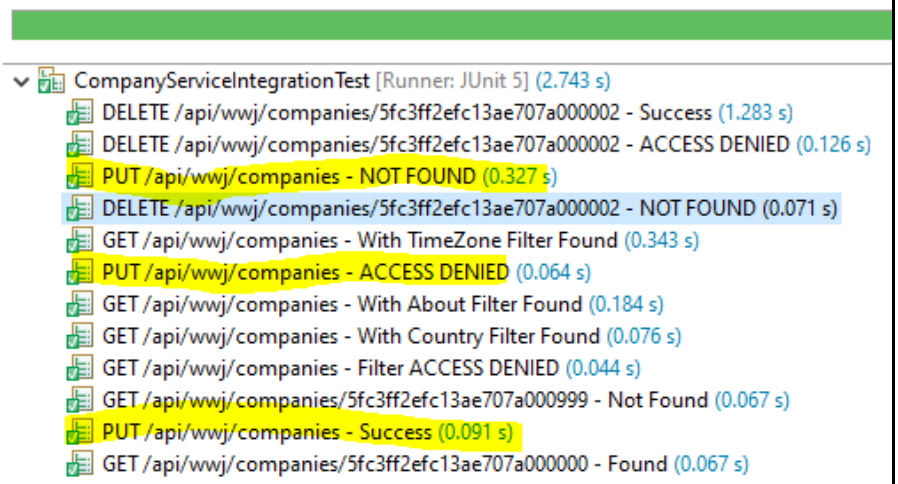
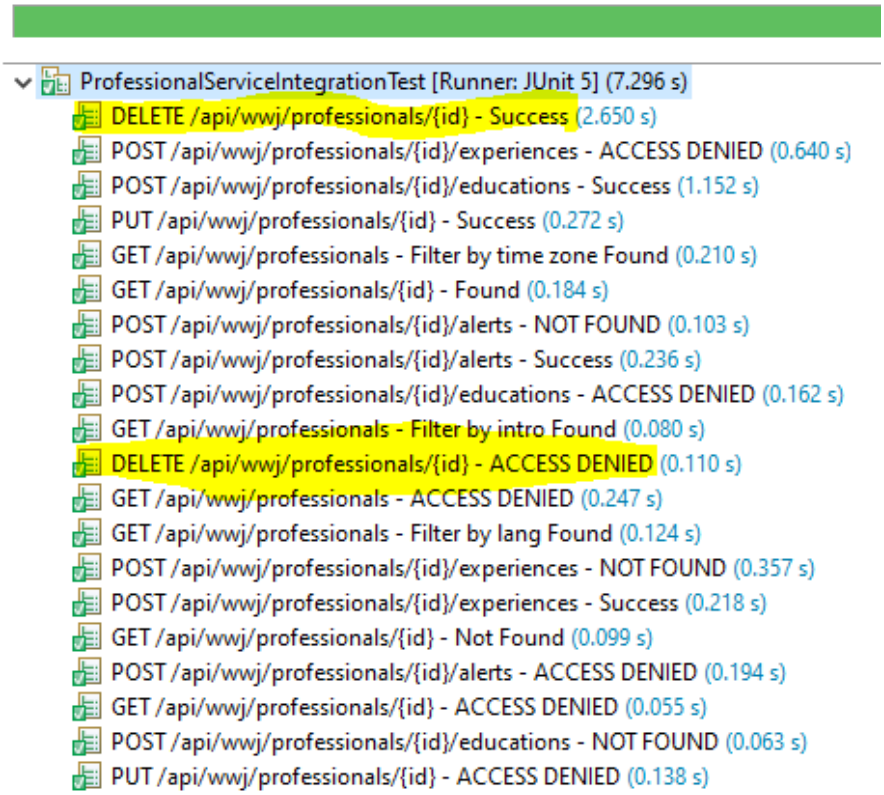
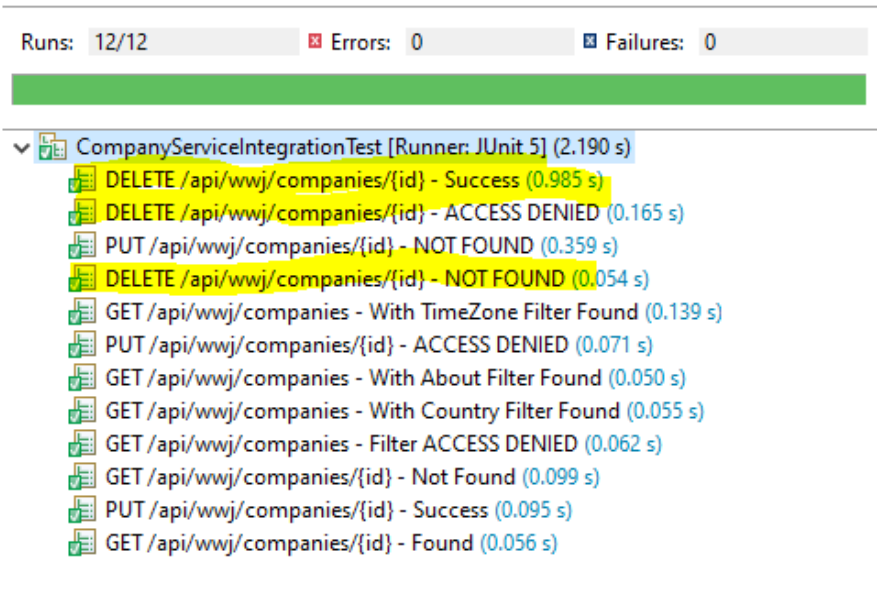
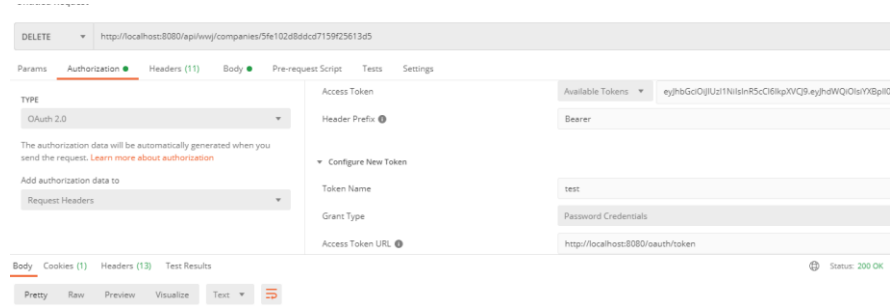
Empresas	

TABLA 29 – DETALLE TESTING HISTORIA DE USUARIO 013

US - 013	Cómo administrador quiero poder eliminar perfiles profesionales o de compañías por reincidir en malas prácticas para mantener la reputación de la plataforma
Test End Point - Junit	
DELETE	/companies/{id}
DELETE	/professionals/{id}

Profesionales	
Companies	
Test End Point - Postman	

Empresas	 Eliminada correctamente
Profesionales