



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

OPERA ID:125

Decide Ortosia - Autenticación

Autores:

Álvaro González
Luca Bonacini
Antonio Millan
Rodrigo Rojas
Antonio Romero

Tutores:

David Benavides Cuevas
Opera Id: 125
Grupo: G1

Índice

1. Resumen	2
2. Introducción y contexto	3
3. Descripción del sistema	3
4. Planificación del proyecto	3
5. Entorno de desarrollo	4
6. Gestión del cambio, incidencias y depuración	4
6.1. Gestión del código fuente	6
6.2. Gestión de la construcción e integración continua	7
6.3. Gestión de liberaciones, despliegue y entregas	8
7. Mapa de herramientas	10
8. Ejercicio de propuesta de cambio	10
9. Conclusiones y trabajo futuro	12

Índice de figuras

1. Ejemplo de incidencia	5
2. Ejemplo de estructuras de commits	6
3. Ejemplo de Pull Request a ortosia-auth	7
4. Ejemplo de Pull Request revisado y aprobado	7
5. Ejemplo de construcción con Travis CI	8
6. Resultado de travis.yml tras añadir la sección de despliegue	9
7. Mapa de herramientas	10
8. Ejemplo de incidencia	12

Índice de cuadros

1. Sistemas desplegados en Heroku	9
---	---

1. Resumen

Decide es una herramienta Open Source que se desarrolla con el fin de facilitar el voto online de manera segura, anonima y eficaz. Decide es un software heredado realizado por el equipo de Wadobo <https://github.com/wadobo/decide>. En un primer momento se realizó un fork de este sistema para que podamos trabajar de forma aislada sin tener que modificar el sistema original. El sistema se divide en varios subsistemas para facilitar el escalado y el mantenimiento del software. Dichos subsistemas se comunican entre ellos a través de llamadas REST.

Nuestro grupo se encarga del subsistema de autenticación. Es el primer paso para que un usuario pueda votar, garantizar que la persona que va a realizar el voto es quien dice ser. Como se ha comentado previamente el código fuente es heredado por lo que teníamos que seguir el lenguaje, estrategia y estructura que se había definido en un principio. El equipo decidió no desarrollar el subsistema con otra tecnología ya que podría afectar al buen funcionamiento del sistema y podríamos encontrarnos con conflictos con los resto de subsistemas.

Después el equipo llegó a un acuerdo sobre las tareas que se iban a desarrollar a lo largo del proyecto entre los que se encuentran distintos métodos de autenticación, documentación de la API, interfaz de usuario, etc. Cada miembro del equipo tenía asignadas unas tareas (*issues*) y para gestionar el código se ha usado un repositorio GIT con el fin de que tuviéramos siempre la última versión del sistema (tanto por nuestra parte como con el resto de los equipos) y poder resolver conflictos de una forma más eficiente.

2. Introducción y contexto

Es un proyecto para que el equipo de trabajo ponga en práctica y observe cómo se ponen en funcionamiento un proyecto real, en este caso se trabaja con Decide que es una plataforma didáctica de votación electrónica.

En el proyecto definimos diferentes subsistemas que serán más o menos independientes entre sí y que se interconectan implementando una API concreta. Cada subsistema se encarga de una tarea concreta en el sistema de voto. Los subsistemas a realizar son los siguientes [1]

1. Autenticación
2. Censo
3. Votaciones
4. Cabina de votación
5. Almacenamiento de votos (cifrados)
6. Recuento/ MixNet
7. Post-procesado
8. Visualización de resultados

Nuestro equipo se encargará del subsistema de autenticación que consistirá en autenticar al votante (con esto nos referimos a que se verifique que realmente la persona que vota dice ser quién es sin que se de el caso de suplantación de identidad), y que el votante solo pueda votar una vez en esa votación.

3. Descripción del sistema

4. Planificación del proyecto

Para la planificación del proyecto el equipo tuvo una reunión al principio del milestone para discutir y llegar a un acuerdo en las tareas que se iba a realizar durante el desarrollo del mismo. Las tareas estan definidas en el apartado de issues del repositorio <https://github.com/pablotabares/decide/issues> cada tarea tiene asignada una/s persona/s que seran los repsonsables de que dicha tarea se lleve a cabo en tiempo y en forma.

Para gestionar el tiempo que ha dedicado cada miembro del equipo hemos utilizado la herramienta Toggle [3] y se han incluido los reportes en el diario del equipo.

Debido a que uno de los miembros se encuentra fuera de Sevilla y debe trabajar de forma remota el equipo llego a un acuerdo de reunirse los miercoles para conocer los avances y problemas que ha tenido el proyecto. Además se usaría Telegram [4] como herramienta de comunicación de forma diaria.

5. Entorno de desarrollo

Nuestra primera aproximación fue usar como entorno de desarrollo Docker ya que este sería tambien el entorno de producción. Sin embargo, encontramos una serie de problemas para cumplir con este objetivo. El primero fue que los cambios que se hicieran en local tenían que verse reflejados en la imagen de Docker y por tanto actualizarse sin problemas. Para ello había que cambiar y configurar el Dockerfile y usar Volumes [2]. Logramos que los cambios se producesen pero no nos era posible debugear el código.

Finalmente optamos por la otra alternativa que aparece en el repositorio principal. Que consta de las siguientes tecnologías:

- Python 3.6.5
- Django 3.7.7
- PostgreSQL 10.6
- Pycharm Studio Professional 2018

Decidimos usar Pycharm Studio Professional ya que la Universidad de Sevilla cuenta con una licencia para estudiantes [5], por lo que tiene mas soporte y más características que Pycharm Studio Community. Además se puede configurar un entorno de Django desde el mismo IDE para que el desarrollo sea más efectivo.

6. Gestión del cambio, incidencias y depuración

Las incidencias y cambios en el sistema se deben publicar en el apartado de *issues* en el repositorio del proyecto. Ya que estamos trabajando con otros subsistemas y otros equipos se propone que el titulo de la incidencia debería comenzar con Auth seguido de la incidencia o cambio propuesto. Por ejemplo: Auth - Email authentication [6]

Además se debe indicar la persona responsable, una etiqueta, proyecto asociado y si es posible indicar el milestone como se muestra en la siguiente imagen:

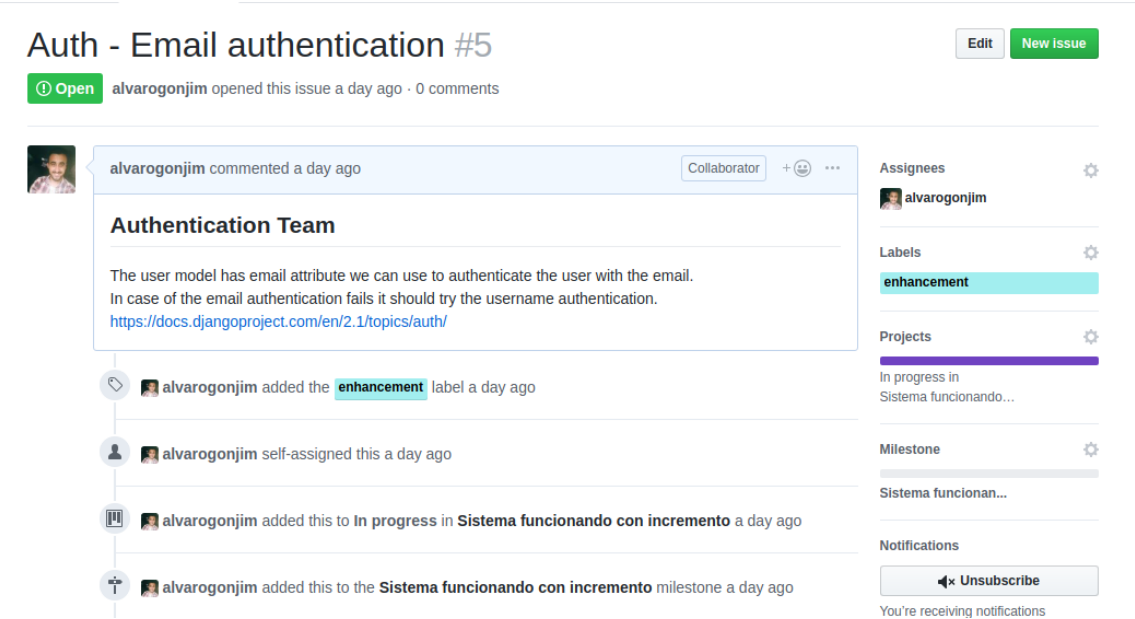


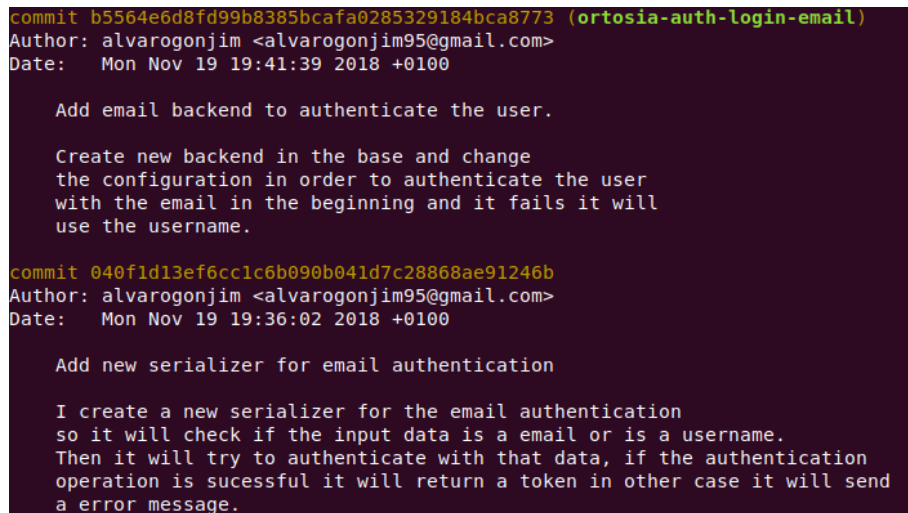
Figura 1: Ejemplo de incidencia

6.1. Gestión del código fuente

La gestión del código fuente de nuestro proyecto se hará a través de Git ya sea por comandos (en la terminal) o a través de una interfaz de usuario. Los commits deben seguir la siguiente estructura [8]:

Titulo: Extracto de los cambios en 50 caracteres o menos.

Texto explicativo más detallado, solo si es necesario. La línea en blanco que separa el titulo del resto del texto es crucial. En el pie, se pueden poner referencias (ids) de las incidencias.



```
commit b5564e6d8fd99b8385bcafa0285329184bca8773 (ortosia-auth-login-email)
Author: alvarogonjim <alvarogonjim95@gmail.com>
Date: Mon Nov 19 19:41:39 2018 +0100

    Add email backend to authenticate the user.

    Create new backend in the base and change
    the configuration in order to authenticate the user
    with the email in the beginning and it fails it will
    use the username.

commit 040f1d13ef6cc1c6b090b041d7c28868ae91246b
Author: alvarogonjim <alvarogonjim95@gmail.com>
Date: Mon Nov 19 19:36:02 2018 +0100

    Add new serializer for email authentication

    I create a new serializer for the email authentication
    so it will check if the input data is a email or is a username.
    Then it will try to authenticate with that data, if the authentication
    operation is sucessful it will return a token in other case it will send
    a error message.
```

Figura 2: Ejemplo de estructuras de commits

Para la gestión de ramas se empleará una rama por incidencia (aunque la incidencia sea muy leve o facil de realizar). El estructura de la rama debe ser *ortosia-auth-titulo de la incidencia* como por ejemplo se puede ver en la imagen previa *ortosia-auth-login-email*

Una vez que se haya finalizado el desarrollo de la incidencia se debe realizar un *Pull Request (PR)* a la rama *ortosia-auth*. Este PR debe ser revisado por una/s persona/s que no sean los responsables y/o desarrolladores de la incidencia. En caso de que apruebe el PR se mergeara, en caso contrario, el responsable deberá realizar los cambios propuestos.

Cuando la rama base de nuestro equipo (*ortosia-auth*) sea estable y no tenga fallos ni conflictos se realizará un PR a la rama *master* y posteriormente el proceso de despliegue.

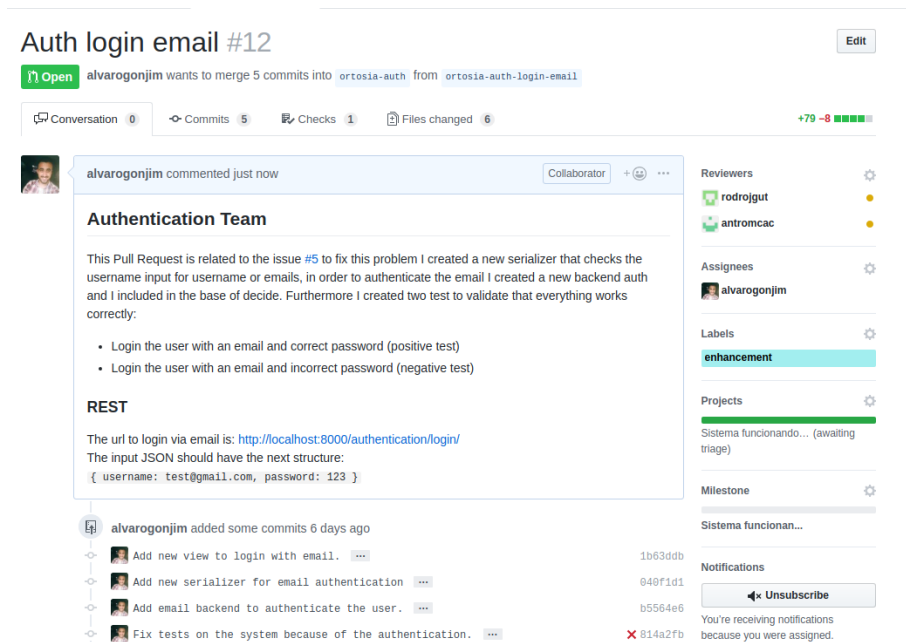


Figura 3: Ejemplo de Pull Request a ortosia-auth

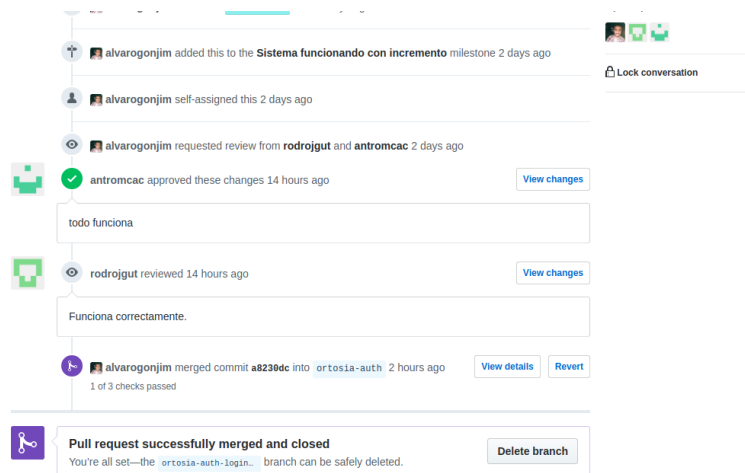


Figura 4: Ejemplo de Pull Request revisado y aprobado

6.2. Gestión de la construcción e integración continua

El proyecto que estamos realizando va ser desplegado en un servidor remoto. Para la integración continua usaremos Travis CI. Travis CI es un servicio de integración continua distribuido y alojado que se utiliza para crear y probar proyectos de softwa-

re alojados en GitHub. Travis CI recoge el código de nuestra aplicación, lo compila y realiza una serie de test con el fin de probar que funciona correctamente. Una vez registrados en Travis CI, procedemos a sincronizado el repositorio de GitHub que vamos a utilizar, creamos un archivo llamado `.travis.yml` en la raíz de nuestro proyecto. Éste archivo contiene una serie de comandos para hacer que los test funcionen y así poder comprobar el buen funcionamiento de nuestro proyecto. Travis CI detecta los cambios automaticamente, pasa los tests y construye el proyecto con los cambios realizados en el repositorio.

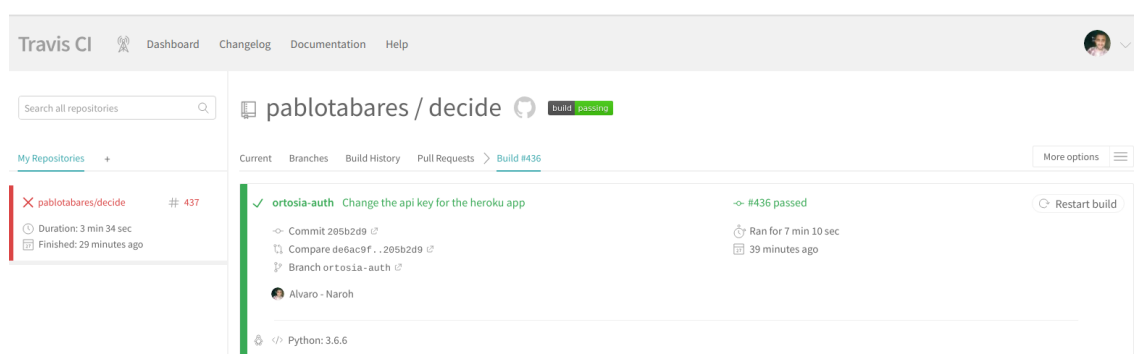


Figura 5: Ejemplo de construcción con Travis CI

6.3. Gestión de liberaciones, despliegue y entregas

Para desplegar el proyecto se va a usar Heroku. Heroku es una plataforma como servicio que permite desplegar y operar aplicaciones en la nube sin ningun problema de forma gratuita. Además se puede configurar con Travis CI para que el proceso de despliegue se realice de forma automatica cuando exista algun cambio en la rama correspondiente. De esta forma nosotros vamos a desplegar nuestro subsistema de forma aislada desde nuestra rama base `ortosia-auth` Mientras que para el sistema completo con todos los subsistemas se va a usar la rama `master`.

Para crear una aplicación de Heroku tenemos que crearnos una cuenta en su pagina principal <https://www.heroku.com>. Luego hacemos click en "Crear una nueva App" definimos un nombre para la nueva app y en la siguiente ventana nos explicará como desplegar el proyecto en esta nueva app a través de Heroku CLI. Sin embargo nosotros usamos Travis CLI (para un despliegue automatico), debemos crear un fichero `Procfile` que determina que acción/es debe realizar Heroku cuando se suba el proyecto y tambien debemos modificar el `travis.yml` con una sección de despliegue como en el siguiente ejemplo:

```
1 language: python
2 python:
3   - 3.6.6
4 install:
5   - pip install -r requirements.txt
6 sudo: required
7 services:
8   - docker
9 script:
10  - sonar-scanner
11  - cd docker
12  - docker exec -ti decide_web ./manage.py test
13  - cd ..
14 addons:
15   sonarcloud:
16     organization: pablotabares-github
17 notifications:
18   email: false
19 before_install:
20   - cd docker
21   - docker-compose up -d
22   - cd ..
23 deploy:
24   provider: heroku
25   api_key:
26     secure: Ex+oACib5efs+6Sq04Fk1muCKwE6xIf64wW/RXyGwS1S9Gu1y16QcSw4gpFdIYLEDY4/R7gVPi6GMwhX+CgsvYriqv01L6dxiRM0MSN+S44neGm4M4Dv
27   app: ortosia-auth
28   on: ortosia-auth
29   skip_cleanup: true
```

Figura 6: Resultado de travis.yml tras añadir la sección de despliegue

Se puede apreciar en la sección deploy que el cliente que vamos a usar es Heroku. El nombre de la aplicación debe coincidir con el que hemos creado previamente (ortosia-auth) y podemos indicar sobre que rama queremos que tome los cambios gracias al atributo 'on'. Podemos ver en la imagen que hay una clave de seguridad, esta clave se puede obtener desde nuestro perfil de Heroku en la sección API Key, pero debemos encriptar dicha API Key con el comando

```
travis encrypt $(heroku auth:token)--add deploy.api_key [9]
```

Sistema	URL	Usuario	Contraseña
Autenticación	https://ortosia-auth.herokuapp.com	administrator	Admin2018.

Cuadro 1: Sistemas desplegados en Heroku

7. Mapa de herramientas

Para el desarrollo hemos realizado el proyecto con django y postgresql. Dicho proyecto se sube a un repositorio (decide) en comun con el resto de equipos. Los subsistemas estaban aislados ya que cada uno tiene su rama propia. Cada vez que hay algun cambio en Github se pasa a la integración continua gracias a Travis CI, este lanza los test de calidad con SonarQube y se construye el proyecto con Docker. En caso de error se notifica al equipo de desarrollo para que solucionen el problema. Si no hay errores y la construcción del proyecto se ha realizado de forma satisfactoria Travis despliega el proyecto en Heroku donde se puede acceder desde cualquier dispositivo.

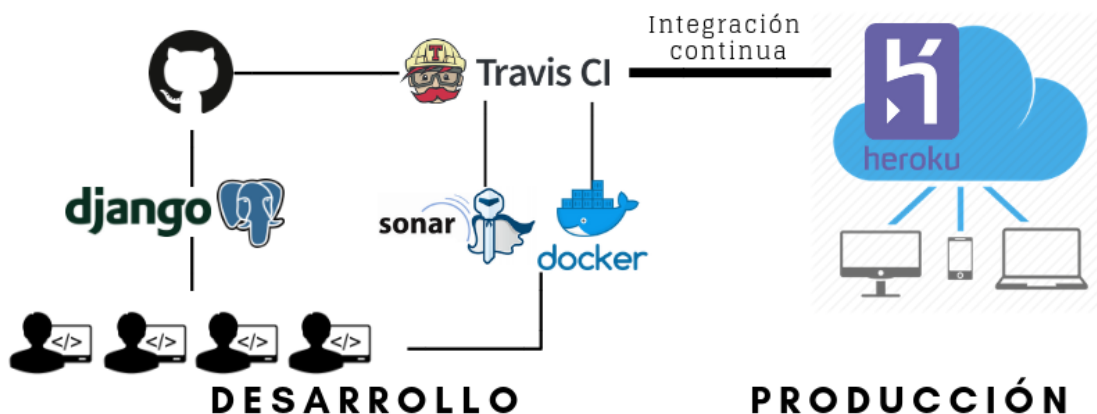


Figura 7: Mapa de herramientas

8. Ejercicio de propuesta de cambio

En esta sección vamos a explicar un cambio desde 0 incluyendo todos los pasos que deberíamos de realizar para llevarlo desde el desarrollo hasta la fase de producción.

El primer paso sería clonar el proyecto de Github:

```
$ git clone https://github.com/pablotabares/decide.git
```

Una vez hecho esto, será necesario instalar las dependencias del proyecto, las cuales están en el fichero requirements.txt:

```
$ pip install -r requirements.txt
```

Tras esto tendremos que crearnos nuestra base de datos con postgres:

```
$ sudo su - postgres
```

```
psql -c create user decide with password 'decide'"
```

```
psql -c create database decide owner decide"
```

Entramos en la carpeta del proyecto y realizamos la primera migración para preparar la base de datos que utilizaremos:

```
$ cd decide $ python manage.py migrate
```

El cambio que proponemos es el de aumentar el numero de atributos que tiene el objeto user. El primer paso que debemos hacer es crear una issue tal como se vió en el apartado de la gestion del código fuente. Esta issue fue reportada en el siguiente url:

<https://github.com/pablotabares/decide/issues/52>

Ahora lo que vamos a hacer es crear una rama siguiendo el estilo anteriormente descrito que sera donde vamos a trabajar en esta issue, para ello usamos el comando:

```
$ git checkout -b ortosia-auth-model-decideuser
```

Ahora podemos trabajar en la issue, sin problemas. Cuando hayamos terminado con los cambios en el desarrollo debemos subir el código y posteriormente hacer un PR a nuestra rama base, para ello debemos de usar el comando:

```
$ git add .
```

```
$ git commit .
```

```
$ git push origin ortosia-auth-model-decideuser
```

Cuando hayamos terminado podemos crear un pull request para ello debemos ir a Github a nuestra rama (ortosia-auth-model-decideuser) y hacer click en new pull request, nosotros ya hemos realizado el PR en la siguiente url; <https://github.com/pablotabares/decide/pull/174>

Nuestro compañero validará el pull request o nos pedirá distintos cambios y en caso de que el pull request sea aceptado podremos mergearlo a nuestra rama base (ortosia-auth).

Ahora si entramos en la web de TravisCI y entramos en la rama de ortosia-auth podemos ver en el fichero de log como se pasan los test unitarios, de calidad de código (SonarQube) y si no hay problemas se desplegará una nueva versión en Heroku. Esto se realiza de forma automatica, no tenemos que preocuparnos de nada. En nuestro caso podemos visualizar dichos resultados en el siguiente enlace:

<https://travis-ci.org/pablotabares/decide/builds/478420406>

Gracias al fichero Postman que hemos realizado podemos hacer una consulta y comprobar que el cambio que hemos realizado funciona sin problemas.

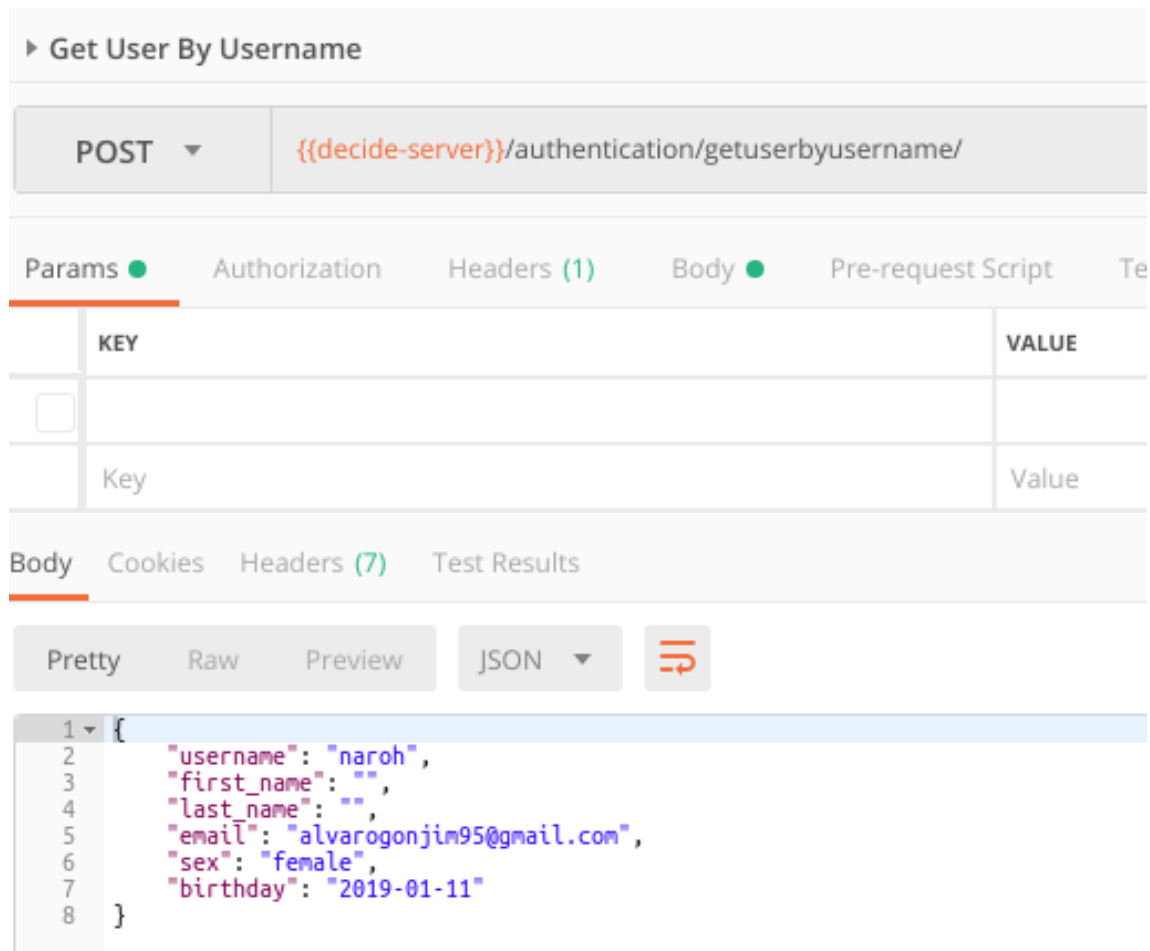


Figura 8: Ejemplo de incidencia

Como se puede ver en la figura, aparecen todos los datos del usuario ademas de los atributos de sexo y fecha de nacimiento.

9. Conclusiones y trabajo futuro

Como conclusiones finales, consideramos que el proyecto se ha realizado de forma exitosa. Se han seguido los procedimientos definidos en este documento para llevar el proyecto a cabo.

Añadir que el equipo ha aprendido nuevas tecnologías y metodologías para que el desarrollo de un producto sea lo mas eficiente posible, gracias a la integración continua, el uso de tests unitarios, test de calidad hasta llevarlo a una fase de producción.

Por último, como trabajo futuro se pueden añadir nuevas funcionalidades al modulo de autenticación como por ejemplo:

1. Autenticación con redes sociales
2. Autenticación a traves de un servidor LDAP
3. Autenticación via SMS o aplicaciones de mensajería.
4. Incrementar numero de tests para que la cobertura de código sea mayor.

Referencias

- [1] Dani GM <https://github.com/EGCETSII/decide/blob/master/doc/subsistemas.md>
- [2] Bialba, 21 Septiembre 2016. <https://stackoverflow.com/questions/39615405/updating-code-in-a-docker-container/>
- [3] <https://toggl.com/>
- [4] <https://web.telegram.org/>
- [5] <https://education.github.com/pack>
- [6] <https://github.com/pablotabares/decide/issues/5>
- [7] <https://www.heroku.com/>
- [8] Miguel Angel Bueno Ferrer, 23 Abril 2017. <https://blog.kirei.io/>
- [9] <https://docs.travis-ci.com/user/deployment/heroku/>