



**UNIVERSIDAD DE CASTILLA-LA MANCHA**  
**ESCUELA SUPERIOR DE INFORMÁTICA**

**“IntraVita – Práctica 1”**

*de Cózar Ruano, Fernando*

*Diezma Rodríguez, José María*

*Rivera Balseras, Christian*

*Gómez Herencia, Daniel*

*Piqueras López, Juan Ángel*

*Álvaro Díaz-Crespo, Miguel*

Grupo:	4
Asignatura:	Procesos de Ingeniería del Software
Titulación:	Grado en Ingeniería Informática
Fecha:	12/12/2017

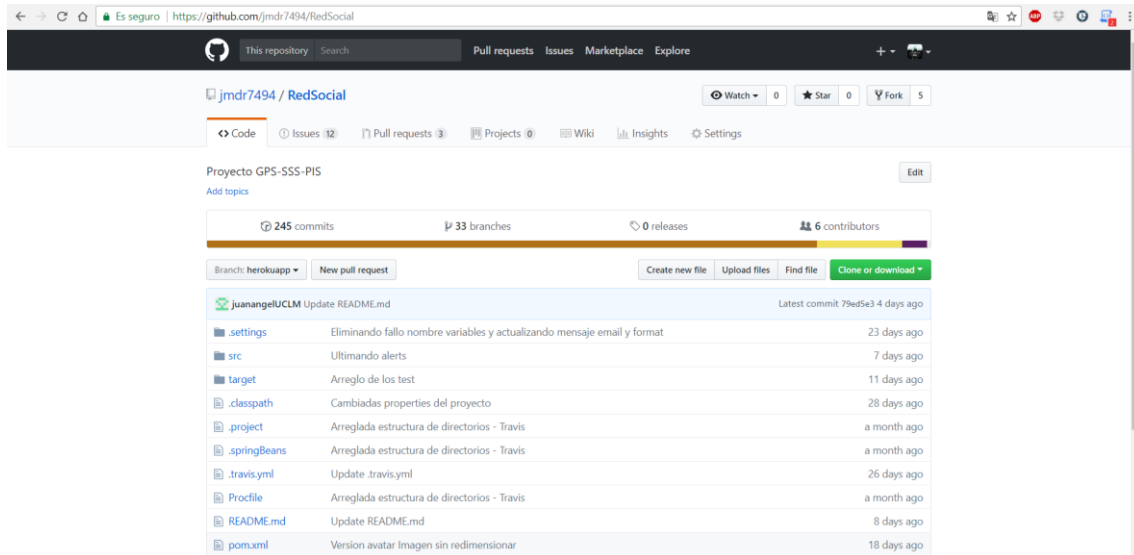
## Índice:

1- Entorno	2
a. GitHub	2
b. ZenHub	2
c. Heroku	3
d. Travis CI	5
e. Spring	9
f. mLab	13
g. Slack	13
2- Gestión de Ramas	14
3- Vistas	14
4- BDD	14
5- Test	15
a. Dominio y persistencia	15
b. Presentación	15

# 1- Entorno

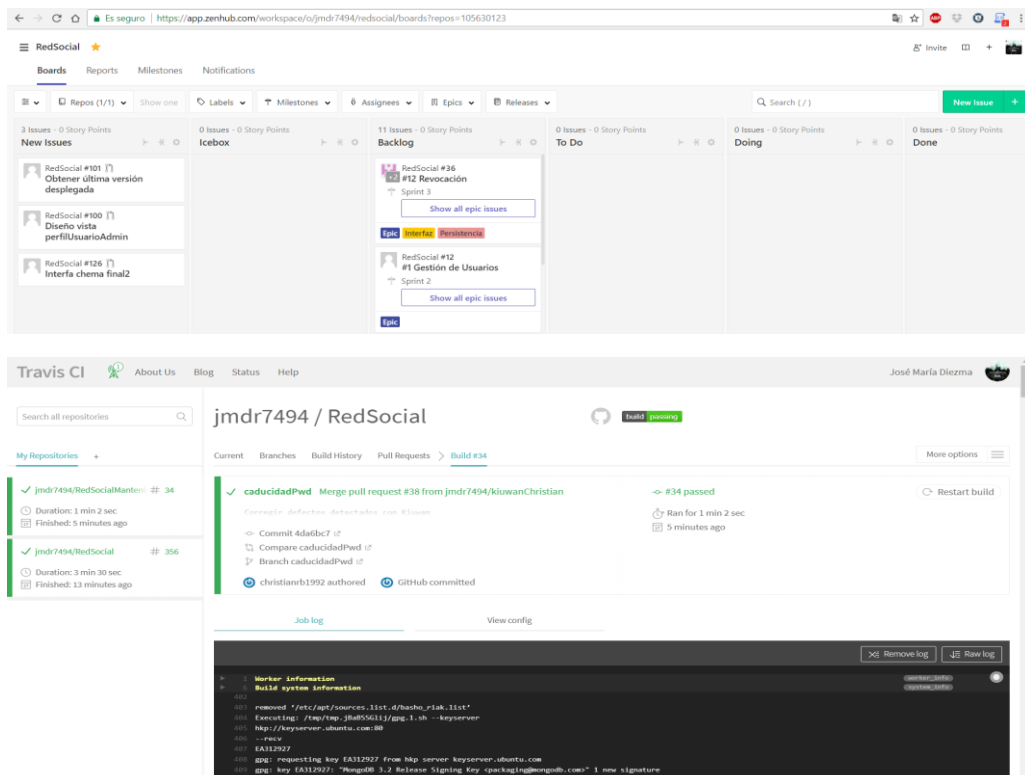
## a. GitHub

Contamos con un repositorio público sobre el que el equipo, previamente invitado, podrá trabajar.



## b. ZenHub

Una vez creado el repositorio, podremos asociar ZenHub al mismo, para crear y gestionar las tareas del mismo.

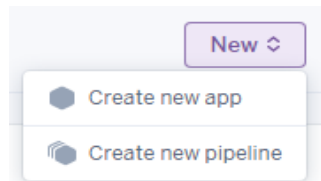


## c. Heroku

Esta herramienta nos va a permitir desplegar la aplicación a través de GitHub, así como a través de la herramienta de integración continua Travis CI.

Lo primero que tenemos que hacer es crearnos una cuenta en <https://www.heroku.com/> y seguimos los siguientes pasos:

- Vamos a la pestaña de New -> Create new app



- Una vez en dentro, rellenamos el siguiente formulario para crear la nueva app

Create New App

App name

Choose a region

Europe

Add to pipeline...

Create app

- El siguiente paso será crear el archivo Procfile (dyno en Heroku) el cual estará localizado en el directorio raíz de nuestro proyecto:

Proyecto GPS-SSS-PIS

245 commits

33 branches

0 releases

6 contributors

Branch: herokuapp

New pull request

Create new file

Upload files

Find file

Clone or download

juanangelUCLM Update README.md

Latest commit 79ed5e3 11 days ago

.settings

Eliminando fallo nombre variables y actualizando mensaje email y format

a month ago

src

Ultimando alerts

14 days ago

target

Arreglo de los test

18 days ago

.classpath

Cambiadas properties del proyecto

a month ago

.project

Arreglada estructura de directorios - Travis

a month ago

.springBeans

Arreglada estructura de directorios - Travis

a month ago

.travis.yml

Update .travis.yml

a month ago

Procfile

Arreglada estructura de directorios - Travis

a month ago

README.md

Update README.md

15 days ago

pom.xml

Version avatar Imagen sin redimensionar

25 days ago

Branch: herokuapp ▾ RedSocial / Procfile

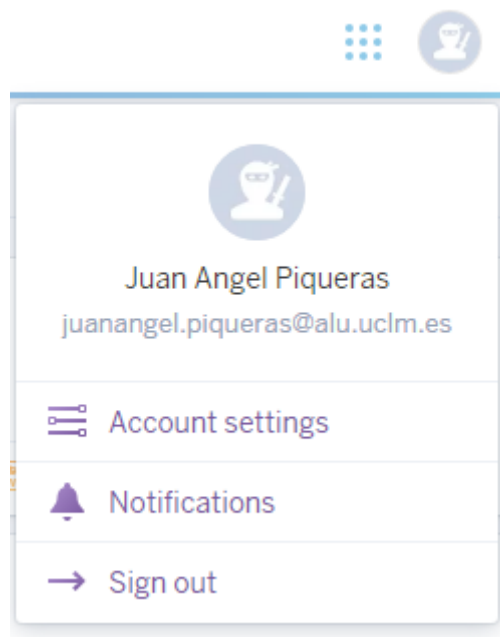
miguelalvaro Arreglada estructura de directorios - Travis

1 contributor

1 lines (1 sloc) | 87 Bytes

1 web: java \$JAVA\_OPTS -jar target/dependency/webapp-runner.jar --port \$PORT target/\*.war

- Ahora volvemos a Heroku y vamos a la pestaña *Account Settings* para obtener la API Key, esta clave le va a permitir a Travis CI conectar con la plataforma de despliegue Heroku automáticamente después de cada *Commit* realizado en Github.



API Key

e109a861-ecbd-4a0c-9653-c650019d489a

Regenerate API Key...

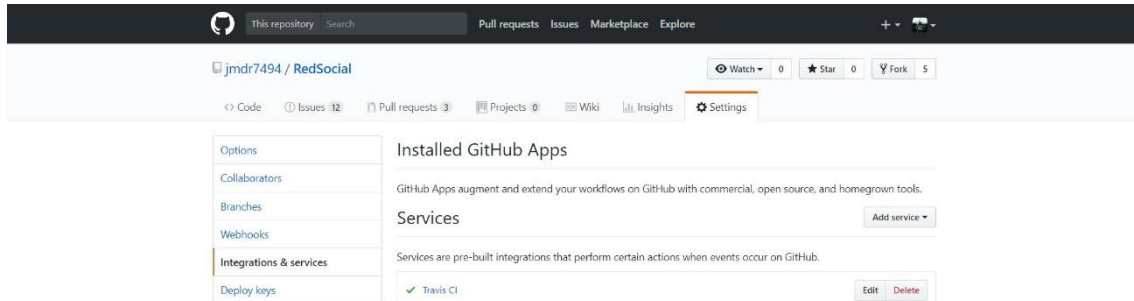
Una vez desplegada, tendremos nuestra aplicación en la web, en nuestro caso, en la siguiente dirección: <https://intravitawebapp.herokuapp.com/>



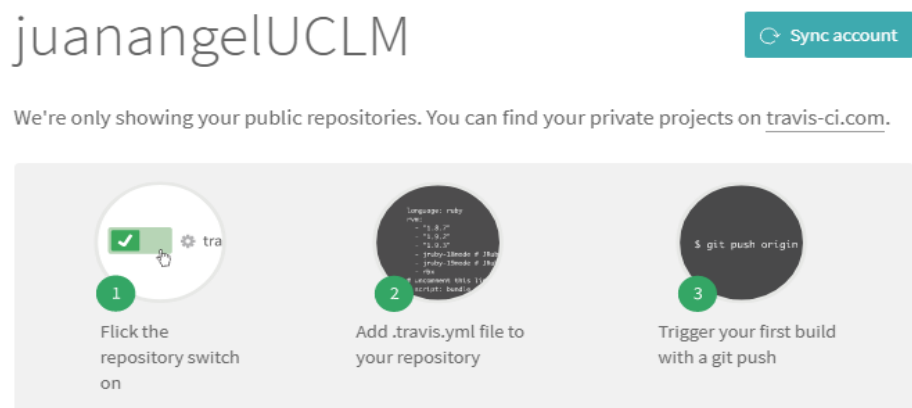
## d. Travis CI

Travis es un servicio de integración continua para construir y probar (testing/deploy) proyectos alojados en GitHub.

- El repositorio contará con el servicio de *Travis CI* activado.

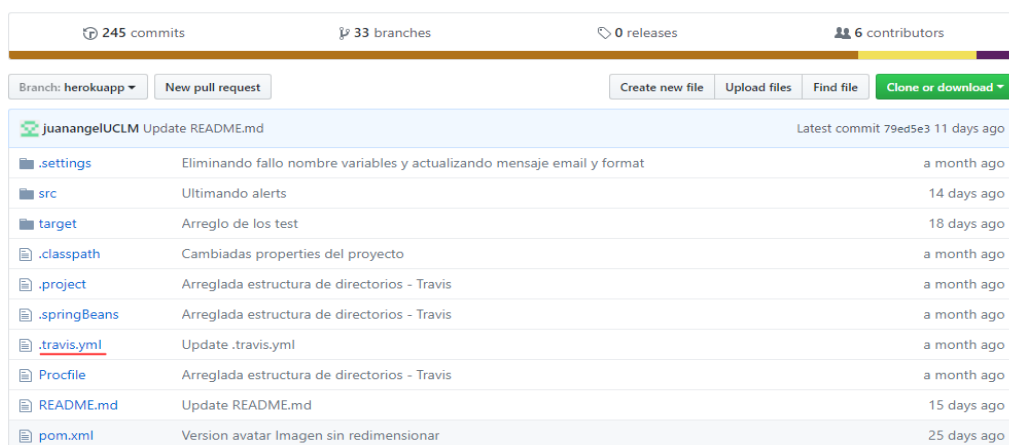


- Accedemos a Travis CI con nuestra cuenta de Github y sincronizamos para que Travis detecte el proyecto.






- Una vez detectado nuestro proyecto, para que Travis CI realice las builds correspondientes a cada commit de Github, tendremos que añadir el archivo `.travis.yml` al directorio raíz de nuestro proyecto:

Proyecto GPS-SSS-PIS



Branch: herokuapp ▾ RedSocial / .travis.yml

 juanangelUCLM Update .travis.yml

2 contributors  

18 lines (15 sloc) | 278 Bytes

```
1  language: java
2  jdk:
3    - oraclejdk8
4
5  deploy:
6    provider: heroku
7    api_key:
8      secure: e109a861-ecbd-4a0c-9653-c650019d489a
9    app: intravitawebapp
10   on:
11     repo: jmdr7494/RedSocial
12     branch: herokuapp
13
14  notifications:
15    slack:
16      rooms:
17        - intravita-g4:DCYo1QUObHAPAGOX1DWM4Bcd#general
```

- En la imagen anterior, podemos observar el contenido del archivo .travis.yml en el que se describe el lenguaje y jdk del proyecto, el proveedor de despliegue de la aplicación en nuestro caso Heroku con su correspondiente API Key anteriormente comentada y finalmente para notificar al equipo el estado final de cada build a través de Slack.
- Para finalizar se muestra un ejemplo del log de una build, en el cual cabe destacar el comando *mvn install*, *mvn test* y finalmente el despliegue de la aplicación.

```

444 $ mvn install -DskipTests=true -Dmaven.javadoc.skip=true -B -V
445 Picked up _JAVA_OPTIONS: -Xmx2048m -Xms512m
446 Apache Maven 3.5.0 (ff8f5e7444045639af65f6095c62210b5713f426; 2017-04-03T19:39:06Z)
447 Maven home: /usr/local/maven-3.5.0
448 Java version: 1.8.0_144, vendor: Oracle Corporation
449 Java home: /usr/lib/jvm/java-8-oracle/jre
450 Default locale: en_US, platform encoding: UTF-8
451 OS name: "linux", version: "4.9.6-040906-generic", arch: "amd64", family: "unix"
452 [INFO] Scanning for projects...
453 [INFO]
454 [INFO] -----
455 [INFO] Building redSocial 1.0.0-BUILD-SNAPSHOT
456 [INFO] -----
457 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-resources-plugin/2.6/maven-resources-plugin-
2.6.pom
458 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-resources-plugin/2.6/maven-resources-plugin-
2.6.pom (8.1 kB at 27 kB/s)
459 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/23/maven-plugins-23.pom
460 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/23/maven-plugins-23.pom (9.2 kB at 484
kB/s)
461 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/22/maven-parent-22.pom
462 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/22/maven-parent-22.pom (30 kB at 1.6 MB/s)
463 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/apache/11/apache-11.pom
464 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/apache/11/apache-11.pom (15 kB at 741 kB/s)
465 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-resources-plugin/2.6/maven-resources-plugin-
2.6.jar
466 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-resources-plugin/2.6/maven-resources-plugin-
2.6.jar (30 kB at 1.8 MB/s)
467 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-compiler-plugin/2.5.1/maven-compiler-plugin-
2.5.1.pom
468 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-compiler-plugin/2.5.1/maven-compiler-plugin-
2.5.1.pom (7.9 kB at 721 kB/s)

1718 $ mvn test -B
1719 Picked up _JAVA_OPTIONS: -Xmx2048m -Xms512m
1720 [INFO] Scanning for projects...
1721 [INFO]
1722 [INFO] -----
1723 [INFO] Building redSocial 1.0.0-BUILD-SNAPSHOT
1724 [INFO] -----
1725 [INFO]
1726 [INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ redsocial ---
1727 [WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
1728 [INFO] Copying 1 resource
1729 [INFO]
1730 [INFO] --- maven-compiler-plugin:2.5.1:compile (default-compile) @ redsocial ---
1731 [INFO] Nothing to compile - all classes are up to date
1732 [INFO]
1733 [INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ redsocial ---
1734 [WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
1735 [INFO] Copying 1 resource
1736 [INFO]
1737 [INFO] --- maven-compiler-plugin:2.5.1:testCompile (default-testCompile) @ redsocial ---
1738 [INFO] Nothing to compile - all classes are up to date
1739 [INFO]
1740 [INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ redsocial ---
1741 [INFO] Surefire report directory: /home/travis/build/jmdr7494/RedSocialMantenimiento/target/surefire-reports
1742 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-junit4/2.12.4/surefire-junit4-2.12.4.pom
1743 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-junit4/2.12.4/surefire-junit4-2.12.4.pom (2.4
kB at 10 kB/s)
1744 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-providers/2.12.4/surefire-providers-
2.12.4.pom
1745 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-providers/2.12.4/surefire-providers-2.12.4.pom
(2.3 kB at 294 kB/s)
1746 [INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-junit4/2.12.4/surefire-junit4-2.12.4.jar
1747 [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-junit4/2.12.4/surefire-junit4-2.12.4.jar (37
kB at 1.9 MB/s)

```



```

1749 -----
1750 T E S T S
1751 -----
1752 Picked up _JAVA_OPTIONS: -Xmx2048m -Xms512m
1753 Running com.webapp.redsocial.RunTest
1754 Feature: Eliminar amistades
1755 INFO : org.mongodb.driver.cluster - Cluster created with settings {hosts=[ds135790.mlab.com:35790], mode=SINGLE,
requiredClusterType=UNKNOWN, serverSelectionTimeout='30000 ms', maxWaitQueueSize=500}
1756 INFO : org.mongodb.driver.cluster - No server chosen by WritableServerSelector from cluster description ClusterDescription{type=UNKNOWN,
connectionMode=SINGLE, serverDescriptions=[ServerDescription{address=ds135790.mlab.com:35790, type=UNKNOWN, state=CONNECTING}]}: Waiting
for 30000 ms before timing out
1757 INFO : org.mongodb.driver.connection - Opened connection [connectionId{localValue:1, serverValue:253362}] to ds135790.mlab.com:35790
1758 INFO : org.mongodb.driver.connection - Monitor thread successfully connected to server with description
ServerDescription{address=ds135790.mlab.com:35790, type=REPLICA_SET_PRIMARY, state=CONNECTED, ok=true, version=ServerVersion{versionList=
[3, 4, 7]}, minWireVersion=0, maxWireVersion=5, maxDocumentSize=16777216, roundTripTimeNanos=30880537, setName='rs-ds135790',
canonicalAddress=ds135790-a.mlab.com:35790, hosts=[ds135790-a.mlab.com:35790], passives=[], arbiters=[], primary='ds135790-
a.mlab.com:35790', tagSet=TagSet{[]}, electionId=7fffffff0000000000000003, setVersion=1, lastWriteDate=Mon Dec 04 17:04:57 UTC 2017,
lastUpdateTimeNanos=61996419650579}
1759 INFO : org.mongodb.driver.connection - Opened connection [connectionId{localValue:2, serverValue:253363}] to ds135790.mlab.com:35790
1760
1761 @Scenario1
1762 Scenario Outline: Eliminar amistad valido                                # com/webapp/redsocial/BorrarAmigo.feature:12
1763   Given Usuario conectado para eliminar un amigo                        # BorrarAmigoTest.Usuario_conectado_para_eliminar_un_amigo()
1764   Given Usuario conectado para eliminar un amigo                        # BorrarAmigoTest.Usuario_conectado_para_eliminar_un_amigo()
1765     When "emisor@hotmail.com" borra a "receptor@hotmail.com" # BorrarAmigoTest.borra_a(String,String)
1766   When "emisor@hotmail.com" borra a "receptor@hotmail.com" # BorrarAmigoTest.borra_a(String,String)
1767     Then Borrar de amigos                                              # BorrarAmigoTest.Borrar_de_amigos()
1768   Then Borrar de amigos                                              # BorrarAmigoTest.Borrar_de_amigos()
1769
1770 @Scenario1
1771 Scenario Outline: Eliminar amistad valido                                # com/webapp/redsocial/BorrarAmigo.feature:13
1772   Given Usuario conectado para eliminar un amigo                        # BorrarAmigoTest.Usuario_conectado_para_eliminar_un_amigo()
1773   Given Usuario conectado para eliminar un amigo                        # BorrarAmigoTest.Usuario_conectado_para_eliminar_un_amigo()

```

```

1988 Installing deploy dependencies
1989 Fetching: rendezvous-0.1.2.gem (100%)
1990 Successfully installed rendezvous-0.1.2
1991 1 gem installed
1992 Fetching: multipart-post-2.0.0.gem (100%)
1993 Successfully installed multipart-post-2.0.0
1994 Fetching: faraday-0.13.1.gem (100%)
1995 Successfully installed Faraday-0.13.1
1996 2 gems installed
1997
1998 Preparing deploy
1999 authentication succeeded
2000 checking for app intravita-g1-maintenance-g4
2001 found app intravita-g1-maintenance-g4
2002 Cleaning up git repository with `git stash --all`. If you need build artifacts for deployment, set `deploy.skip_cleanup: true`. See
https://docs.travis-ci.com/user/deployment/#Uploading-Files.
2003 Saved working directory and index state WIP on (no branch): 1def536 Merge pull request #79 from jmdr7494/EditorPublicacion2
2004
2005 Deploying application
2006 creating application archive
2007 uploading application archive
2008 triggering new deployment
2009
2010 -----> Java app detected
2011 -----> Installing OpenJDK 1.8... done
2012 -----> Installing Maven 3.3.9... done
2013 -----> Executing: mvn -DskipTests clean dependency:list install
2014 [INFO] Scanning for projects...
2015 [INFO]
2016 [INFO] -----
2017 [INFO] Building redSocial 1.0.0-BUILD-SNAPSHOT
2018 [INFO] -----
2019 [INFO]

```

```

2045 [INFO]
2046 [INFO] --- maven-war-plugin:2.2:war (default-war) @ redsocial ---
2047 [INFO] Packaging webapp
2048 [INFO] Assembling webapp [redsocial] in [/tmp/build_ca441418224d66471c7f744228a4ec4a/target/redsocial-1.0.0-BUILD-SNAPSHOT]
2049 [INFO] Processing war project
2050 [INFO] Copying webapp resources [/tmp/build_ca441418224d66471c7f744228a4ec4a/src/main/webapp]
2051 [INFO] Webapp assembled in [223 msecs]
2052 [INFO] Building war: /tmp/build_ca441418224d66471c7f744228a4ec4a/target/redsocial-1.0.0-BUILD-SNAPSHOT.war
2053 [INFO] WEB-INF/web.xml already added, skipping
2054 [INFO]
2055 [INFO] --- maven-dependency-plugin:2.3:copy (default) @ redsocial ---
2056 [INFO] Configured Artifact: com.github.jsimone:webapp-runner:8.5.23.0:jar
2057 [INFO] Copying webapp-runner-8.5.23.0.jar to /tmp/build_ca441418224d66471c7f744228a4ec4a/target/dependency/webapp-runner.jar
2058 [INFO]
2059 [INFO] --- maven-install-plugin:2.4:install (default-install) @ redsocial ---
2060 [INFO] Installing /tmp/build_ca441418224d66471c7f744228a4ec4a/target/redsocial-1.0.0-BUILD-SNAPSHOT.war to
/app/tmp/cache/.m2/repository/com/webapp/redsocial/1.0.0-BUILD-SNAPSHOT/redsocial-1.0.0-BUILD-SNAPSHOT.war
2061 [INFO] Installing /tmp/build_ca441418224d66471c7f744228a4ec4a/pom.xml to /app/tmp/cache/.m2/repository/com/webapp/redsocial/1.0.0-
BUILD-SNAPSHOT/redsocial-1.0.0-BUILD-SNAPSHOT.pom
2062 [INFO] -----
2063 [INFO] BUILD SUCCESS
2064 [INFO] -----
2065 [INFO] Total time: 7.038 s
2066 [INFO] Finished at: 2017-12-04T17:05:46+00:00
2067 [INFO] Final Memory: 25M/234M
2068 [INFO] -----
2069 -----> Discovering process types
2070 Procfile declares types -> web
2071
2072 -----> Compressing...
2073 Done: 111.4M
2074 -----> Launching...
2075 Released v69
2076 https://intravita-g1-maintenance-g4.herokuapp.com/ deployed to Heroku

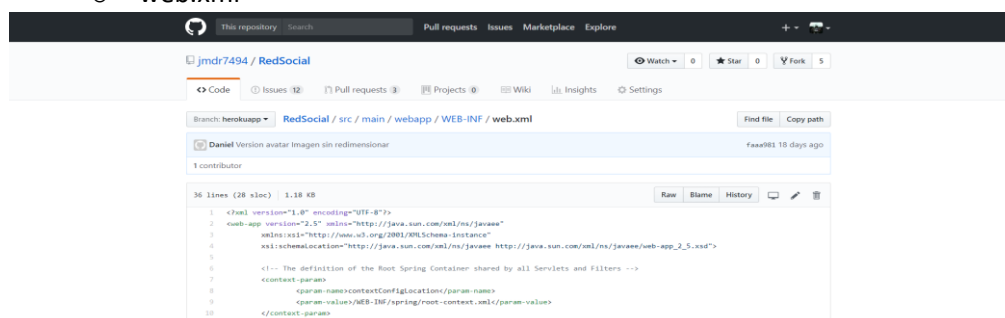
```

## e. Spring

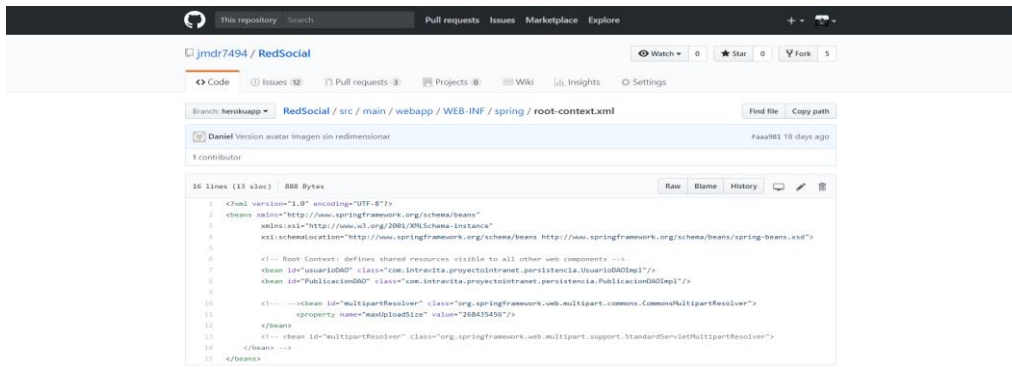
En este apartado, vamos a identificar los ficheros de configuración, sus dependencias en el pom.xml y los controladores.

- Dependencias: además de las típicas de spring, tenemos log4j, para poder ver los mensajes a través de la consola. (No vamos a incluir el pom.xml debido a su tamaño, y a que está incluido en el repositorio y se puede consultar sin problema).
- Ficheros de configuración: Los vamos a ver (parcialmente) en las imágenes.

### o web.xml



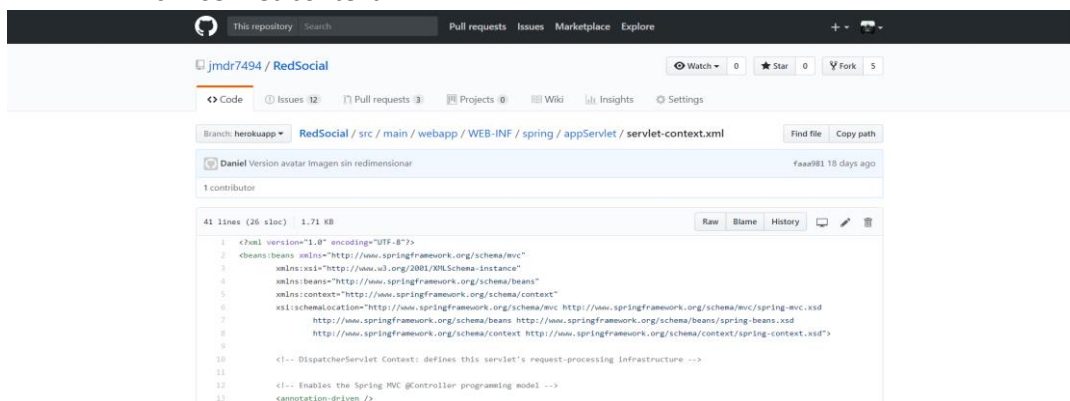
## ○ root-context.xml



The screenshot shows the GitHub repository for jmdr7494/RedSocial. The file root-context.xml is selected, showing its content. The file is 16 lines long, 13 kilobytes, and 888 bytes. It contains XML code for Spring configuration, including beans for user and publication management, and a multipart resolver.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
5
6     <!-- Root Context: defines shared resources visible to all other web components -->
7     <bean id="usuarioDAO" class="com.intravita.proyectointrant.persistencia.UsuarioDAOImpl"/>
8     <bean id="publicacionDAO" class="com.intravita.proyectointrant.persistencia.PublicacionDAOImpl"/>
9
10    <!-- -->
11    <bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
12      <property name="maxUploadSize" value="268435456"/>
13    </bean>
14    <!-- -->
15    <bean id="multipartResolver" class="org.springframework.web.multipart.support.StandardServletMultipartResolver"/>
16  </beans>
```

## ○ servlet-context.xml



The screenshot shows the GitHub repository for jmdr7494/RedSocial. The file servlet-context.xml is selected, showing its content. The file is 41 lines long, 26 kilobytes, and 1.71 KB. It contains XML code for Spring configuration, including beans for MVC and context, and a dispatcher servlet.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/mvc"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:beans="http://www.springframework.org/schema/beans"
5       xmlns:context="http://www.springframework.org/schema/context"
6       xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd
7       http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
8       http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">
9
10    <!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->
11
12    <!-- Enables the Spring MVC @Controller programming model -->
13    <annotation-driven />
```

Aparte tendríamos el controlador, y las dependencias necesarias para activarlo.

## Tratamiento de imágenes a través de Spring:

Para permitir el almacenamiento de una imagen, por ejemplo para incluir el avatar que tendrá el usuario, es necesaria la inclusión de múltiples dependencias en pom.xml. También es necesario añadir en los archivos de configuración, en las vistas de presentación y en el código fuente del controlador la activación de la característica MultiPart.

Multipart nos permite tomar imágenes desde capa de presentación, tomarlas a través de un servicio y enviarlas a base de datos. Pero Multipart no solo sirve para imágenes, en general es útil para guardar archivos desde capa de presentación para tratarlos en el interior de un servicio y realizar las acciones que correspondan según la naturaleza del problema.

En nuestro caso, lo utilizaremos para tratar el avatar de usuario, desde la selección del archivo en capa de presentación hasta su almacenamiento en base de datos. Donde posteriormente para mostrar la imagen se realizará el camino inverso teniendo en cuenta que antes de mostrar la imagen en presentación es necesaria su codificación en base64 para su interpretación como imagen en interfaz de usuario.

Los pasos son los siguientes:

1. Activar la característica de subida de archivos en apache, para ello usaremos las siguientes dependencias en pom.xml

```

<!-- Apache Commons FileUpload -->
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3.1</version>
</dependency>

<!-- Apache Commons IO -->
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.4</version>
</dependency>

```

## 2. Activar las características de configuración en el archivo root-context.xml

```

<!-- --><bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <property name="maxUploadSize" value="268435456"/>
</bean>

```

## 3. Activar las características de configuración en el archivo servlet-context.xml

```

<beans:bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">

    <!-- setting maximum upload size -->
    <beans:property name="maxUploadSize" value="400000" />

```

## 4. Preparar el servicio para recibir la imagen(UsuarioServlet método registrar)

```

MultipartHttpServletRequest multipartRequest = (MultipartHttpServletRequest) request;

CommonsMultipartFile multipartFile = (CommonsMultipartFile) multipartRequest.getFile("fichero");

byte[] imagen = multipartFile.getBytes();

usuario.setImagen(imagen);

```

Guardando la imagen como tipo de dato byte[] aseguramos que no existirá una superpoblación de caracteres en las entradas correspondientes en base de datos que contengan archivos

5. Preparar la interfaz para enviar la imagen(registrar.jsp)  
Activar en la acción la característica Multipart

```
<form action="registrar" method="POST" enctype="multipart/form-data">
```

Usar un selector de archivos

```
<input id="input-b5" name="fichero" type="file"
```

6. Preparar el servicio para enviar la imagen(UsuarioServlet método login)

```
usuario = usuarioDao.selectNombreImagen(nombre);  
request.getSession().setAttribute(usuario_conect, usuario);  
String base64Encoded = DatatypeConverter.printBase64Binary(usuario.getImagen());  
model.addAttribute("imagen", base64Encoded);  
listarPublicacion(request, model);  
return cadenaUrl += welcome;
```

Tomamos la imagen almacenada en el usuario conectado y la añadimos en presentación codificada en base64.

Para el resto de usuarios utilizamos el método listarPublicacion, donde realizaremos el mismo paso para todos los usuarios a los que tenga acceso el usuario conectado.

7. Preparar la interfaz para recibir la imagen(bienvenido.jsp)  
De la siguiente manera mostramos el avatar del usuario conectado:

```

```

Para el resto de usuarios a los que tiene acceso el usuario conectado, la manera de mostrar la imagen tendrá el mismo procedimiento.

Además, y para finalizar con Spring, se han utilizado componentes para el modelo usuario llamados "bean". Estos componentes son características que contiene Java por defecto y permiten, entre sus otras funcionalidades, añadir el contenido de un modelo en presentación tratándolo como un modelo tanto en el servicio como en capa de presentación.

De la siguiente manera añadimos el bean en bienvenido.jsp

```
<jsp:useBean id="usuarioConectado" scope="session" class="com.intravita.proyectointranet.modelo.Usuario"></jsp:useBean>
```

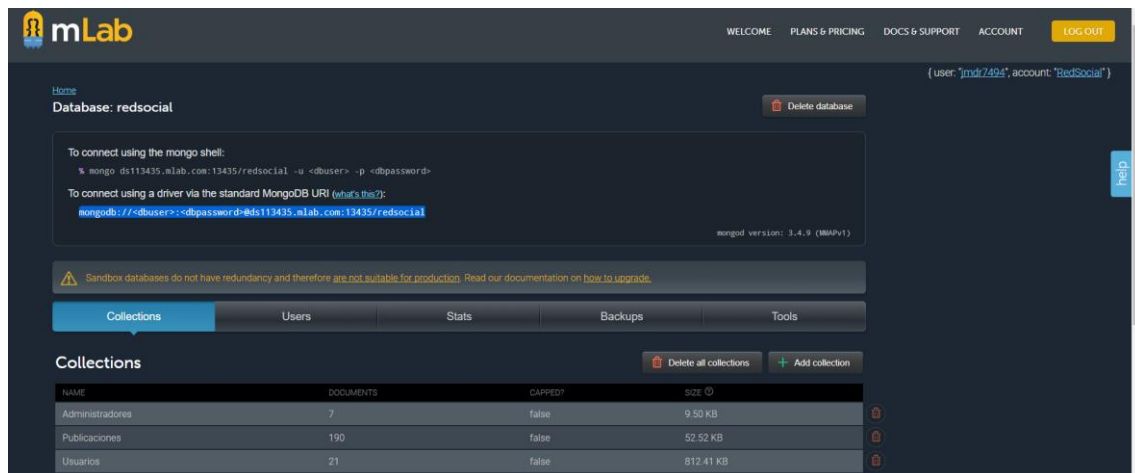
De la siguiente manera podemos usar el bean en un campo de texto o imagen

```
<jsp:getProperty name="usuarioConectado" property="nombre"/>
```

## f. mLab

La base de datos por la que nos hemos decantado ha sido mlab, ya que está en la nube y nos permite trabajar de una forma distribuida y sin preocuparnos de la misma.

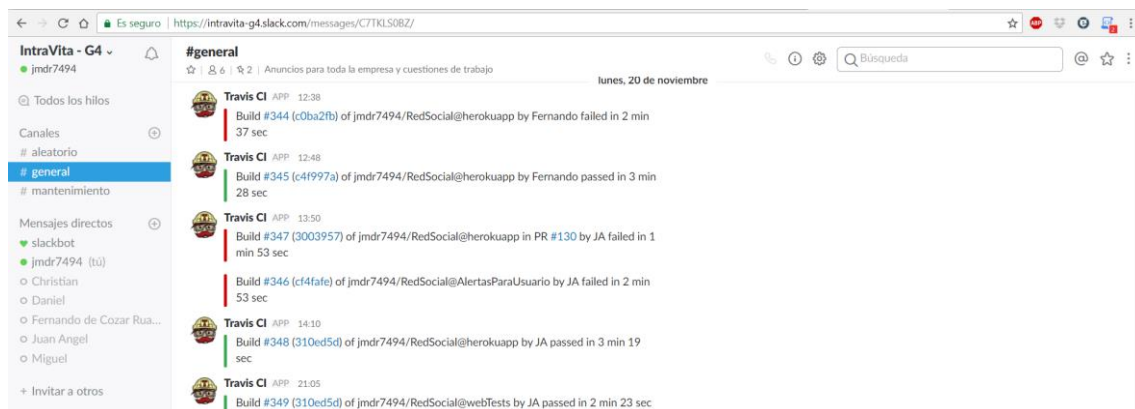
Cuenta con tres colecciones de datos, las cuales aparecen en la siguiente imagen, y cada una de ellas contiene los campos que han sido precisos para almacenar a los usuarios y administradores (sus contraseñas se han almacenado encriptadas) y las publicaciones (con los “me gusta” y las personas que las han compartido).



## g. Slack

Esta herramienta que nos ha permitido mantener una comunicación muy buena a todo el equipo. Gracias a ella hemos podido avisar de actualizaciones, problemas e imprevisto a todo el equipo de una manera cómoda y muy familiar (parecido a telegram).

Además, nos ha permitido enlazar a ella otras herramientas como Travis CI, la cual nos comunica si hace bien o no los *build* cada vez que se hacía cualquier *commit*.



## 2- Gestión de Ramas

Cada componente del equipo de trabajo se auto-asigna tareas definidas dentro de ZenHub conforme a sus conocimientos, intereses y carga de trabajo, cada vez que elige hacer un trabajo, creará una rama a partir de la “*máster*” que siempre tiene los cambios que están probados y completos.

Una vez se tiene una rama, la cual suele tener el nombre de la tarea a realizar, ésta se importa en Eclipse, y se harán y probarán los cambios para la nueva funcionalidad. Después, se hará *commit* en la rama de trabajo, y se hará “*Pull Request*” sobre la *máster* para que cuente con las nuevas funcionalidades.

En caso de haber problemas para hacer la mezcla, escogeremos los cambios que queremos sobrescribir, o mediante una forma más manual, hemos tenido casos de incluir cambios a mano mediante comparadores de texto que nos han facilitado la detección de cambios para no “machacar” el trabajo de los compañeros, es una forma de solucionar estas situaciones que a priori puede parecer una locura, pero ha resultado ser bastante efectiva.

Por último en este apartado, comentar que la comunicación del grupo es fundamental para evitar problemas, y por ello se ha utilizado *telegram* en primer lugar, para después pasar a *Slack*, herramientas que nos han permitido estar en una continua comunicación.

## 3- Vistas

Para la realización del diseño, nos hemos apoyado en Bootstrap para hacer que el sitio web sea perfectamente visible desde cualquier dispositivo con el que se quiera acceder.

Gracias a Bootstrap tendremos una colocación de elementos en una especie de tabla con doce columnas, que podremos manejar conforme a nuestras necesidades y se adaptarán a las pantallas de los dispositivos.

Para dar estilo al sitio web, es decir, usar colores de fondo, botones personalizados, nos hemos apoyado en los estilos predefinidos de Bootstrap y en CSS para adaptarlos a nuestra interfaz.

Todo el código referente a las hojas de estilo en cascada se encuentra en la cabecera (<head>) de las vistas, ya que no pudimos crear una única hoja que heredaran por igual todas las páginas del sitio web, lo que sería la opción más adecuada.

La interfaz en general ha sido muy cuidada, intentando hacerla lo más cómoda e intuitiva posible al usuario para que el aprendizaje sea rápido, mediante el uso de modales de ayuda, títulos en los botones, uso de iconos y colores, además de alertas que informen de los errores.

## 4- BDD

Para desarrollar el proyecto se ha optado por aplicar el desarrollo basado en pruebas, donde se escriben las pruebas antes del código fuente.

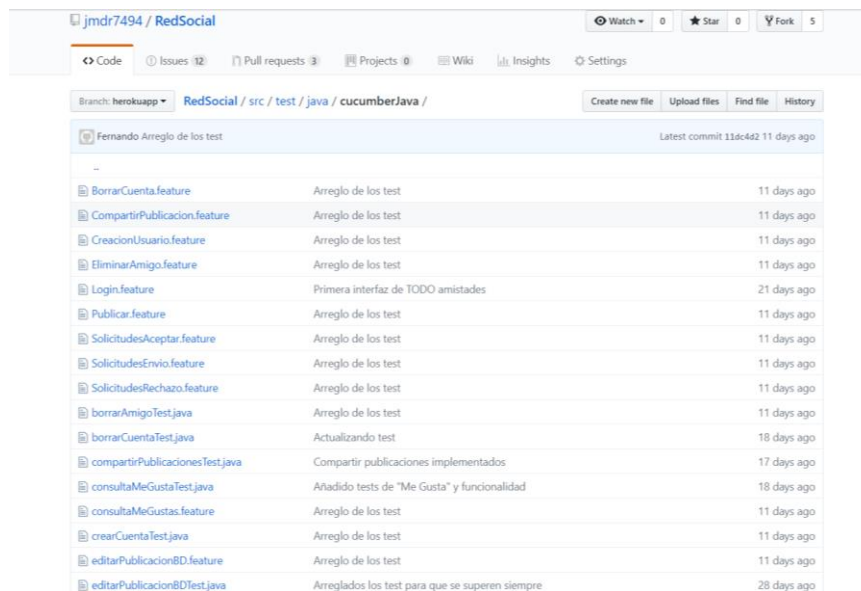
Para realizar esto, necesitamos a *cucumber*, herramienta que automatizará nuestras pruebas BDD, permitiendo ejecutar descripciones funcionales en texto plano como pruebas de software automatizadas (mediante *JUnit*).

\*En el *pom.xml* contamos con las dependencias *Maven* para el uso de *cucumber*.

En la imagen podemos ver el directorio donde se encuentran los test.

Durante el desarrollo de los diferentes sprint, las pruebas se han ido refinando.

Inicialmente teníamos diferentes archivos .feature para la definición de los escenarios y un único archivo .java para la ejecución de los mismos. Por cuestiones de organización, a partir del segundo sprint se decidió tener un único .java para cada .feature. Durante la realización del primer sprint, los test el fallo que tenían es que funcionaban únicamente dadas unas condiciones concretas en la base de datos, eso se solucionó en el sprint 2. Durante el sprint 2 (a parte de, como resulta evidente, añadir nuevos test a las nuevas funcionalidades), se encontró el problema que, cada vez que se ejecuta X test, la base de datos se modifica (se insertaban nuevos registros), eso en sí no es problema, pero para hacer el despliegue, subirlo a github... los test se ejecutaban múltiples veces y eso lo que conseguía era llenar la base de datos de registros inútiles, en el tercer sprint esto se solucionó y los test ahora funcionaban sobre cualquier estado de la base de datos y una vez ejecutados el estado inicial coincidía con el estado final. En el desarrollo de estos test no se consideró el uso de tablas Gherkin para la inclusión de múltiples ejemplos en la ejecución de los mismos.



jmdr7494 / RedSocial		
Watch 0 Star 0 Fork 5		
Code Issues 12 Pull requests 3 Projects 0 Wiki Insights Settings		
Branch: herokuapp RedSocial / src / test / java / cucumberJava /		
Create new file Upload files Find file History		
Fernando Arreglo de los test Latest commit 11d442 11 days ago		
-		
BorrarCuenta.feature	Arreglo de los test	11 days ago
CompartirPublicacion.feature	Arreglo de los test	11 days ago
CreacionUsuario.feature	Arreglo de los test	11 days ago
EliminarAmigo.feature	Arreglo de los test	11 days ago
Login.feature	Primera interfaz de TODO amistades	21 days ago
Publicar.feature	Arreglo de los test	11 days ago
SolicitudesAceptar.feature	Arreglo de los test	11 days ago
SolicitudesEnvio.feature	Arreglo de los test	11 days ago
SolicitudesRechazo.feature	Arreglo de los test	11 days ago
borrarAmigoTest.java	Arreglo de los test	11 days ago
borrarCuentaTest.java	Actualizando test	18 days ago
compartirPublicacionesTest.java	Compartir publicaciones implementados	17 days ago
consultaMeGustaTest.java	Añadido tests de "Me Gusta" y funcionalidad	18 days ago
consultaMeGustas.feature	Arreglo de los test	11 days ago
crearCuentaTest.java	Arreglo de los test	11 days ago
editarPublicacionBD.feature	Arreglo de los test	11 days ago
editarPublicacionBDTest.java	Arreglados los test para que se superen siempre	28 days ago

## 5- Test

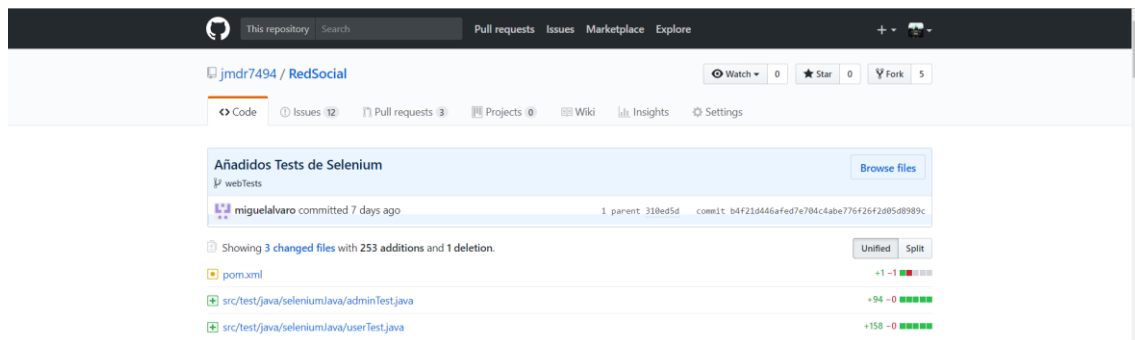
### a. Dominio y persistencia

Apartado anteriormente visto en BDD.

### b. Presentación

Las pruebas de la interfaz se han realizado con POSTMAN, para comprobar la correcta visualización de todos los elementos que componen la interfaz (estilos, posiciones...) y con Selenium para corroborar que todas las alertas se muestran adecuadamente, mediante la introducción de datos en los formularios (registros, publicaciones, credenciales...).





En la imagen, vemos la rama en la que se han incluido los test de Selenium, junto con su correspondiente modificación del POM.