LandSense API Webapp

This webapp is packaged as a .war file which should be deployed onto a Tomcat 7 container. It uses the RestEasy JAX-RS implementation and the Servlet 3.0 spec, and is documented with openAPI (swagger).

# Structure

The application is structured as a multi-module maven project, with a parent project (wps-parent), the main application module consisting of the algorithms (wps-app), and a module containing the RESTful API and OAuth2 implementations (wps-rest) which depends on wps-app. wps-rest.war will be the web archive deployed onto the Tomcat container.

The application structure is fairly straightforward. The top level swagger definition can be found in ISwaggerDefinition.java, and there will be little need to change this. Each logical grouping of REST endpoints (currently thematic accuracy, positional accuracy, logical consistency) consists of the following:

* A **resource** class which exposes one or more REST endpoints with JAX-RS annotations, and documents them with swagger annotations. This should parse/transform input data as required and delegate to the service class, and return a Response object to the user. e.g. PositionalAccuracyResource.java
* A **service** class (an interface and an implementation). This contains all of the logic for using the algorithms from wps-app, error checking, and marshalling data into a model class (if required) to present to the user as part of the HTTP response. E.g. PositionalAccuracyService.java and PositionalAccuracyServiceImpl.java
* Optional **model** classes (POJOs) which may have swagger annotations within. E.g. TargetReferenceDoublet.java. Note in particular the use of the @JsonRawValue annotation to return Strings which are actually JSON (used for FeatureCollections etc.)

The wps-config.xml file is registered in WPSServletContextListener.java when the servlet is first created, and any OAuth2 token checking is performed by the TokenValidationFilter.java servlet filter, which will check any REST endpoint marked with the @Secured annotation.

# Extending

To extend the REST API with a new logical grouping:

1. Create a new package for the new grouping of rest endpoints
2. Create any model classes (POJOs) that you will need. Add swagger annotations if you want/need to.
3. Create a service interface and implementation class. This will take the inputs from the REST request and supply them to the algorithms from wps-app, and create a response object using the models created previously (or an inbuilt class).
4. Create a new resource class and add a method for each endpoint. See other resource classes for JAX-RS and swagger annotations used to expose and document the endpoints. Each method here will correspond to a method in your service class, and a particular rest endpoint. Use this class to instantiate a service class, and pass the input values from the REST request to it.
5. Mark any endpoints in your resource class which you want to protect with OAuth2 with the @Secured annotation and add swagger annotations if you wish to (currently the swagger @Authorization annotation is used at the top of each resource class and so will be applied to each endpoint. If you intend to have unprotected endpoints then this annotation needs to be at the method level instead.
6. Register your resource class in RestApplication.java

(or obviously you can append to the groupings which already exist in a similar fashion).

# Usage

Once the webapp is built and deployed, the REST endpoints can be used either using CURL or a similar browser http requesting tool (e.g. postman, rest console) or by using the swagger UI page found at <http://localhost:8080/wps-rest/swagger/index.html>

You’ll find the oauth.properties file in WEB-INF/classes (it’s currently hardcoded to be there). This file specifies the following:

* **client.id**: The client ID used for authorizing the service to the auth server
* **client.secret**: The client secret, also for authorization
* **admin.key**: A token value which can be used to bypass the token introspection (for testing).
* **introspect.url**: The introspection endpoint provided by the authorization server

You’ll also find the wps-config.xml file in the WEB-INF/classes folder. Again, this location is currently hardcoded.

The swagger UI page can be used to pass an OAuth2 token by using the “authorize” button at the top of the page and providing your token as an API key. This key will be passed with each query as an “access\_token” query parameter. This should be done over HTTPS, however currently it’s just using HTTP.

There is currently an admin token which can be used to bypass oauth filtering (for testing) which can be specified in the oauth.properties file. You can omit this entry in the properties file to disable the feature, but will probably want to remove the feature entirely when ready for production (see TokenValidationFilter.java).