Fragalysis - Job Launcher

Author:	Duncan Peacock
Version:	Approved
Date:	08/03/2022

Release History

Version	Contents	
0.1	Initial discussion document - will change wildly..	
0.2	Main points meeting 14/02/2022: - When a snapshot is created for purposes of running jobs, all the relevant files relating to the jobs will be copied into the project. - Workflows will be used for combinations rather than groups. - Definition of success window and automatic upload of SDF results files in scope - with added Metadata from job (e.g. PDF/fragments) - Deletion of computed sets is in scope (see Appendix for original discussion).	
0.3	17/02/2022: Updated following discussions with Alan on how Squonk and Fragalysis will interact: - Added Roadmap section with possible individual tickets and initial estimates. - Ready for initial review on overall structure and process.	
1.0draft	03/03/2022: Major update following design meeting on job launcher window. Main points: - Job configuration table will be dropped. Job parameters will be provided directly to the frontend as a yaml file. - Fragalysis will be responsible for the job_launcher window using the Forms library in a similar way to Squonk. - Complete document reworked.	
1.0draft - v2	07/03/2022: Update following review comments: - POSTing the job instance to Squonk2 will now be done by the Fragalysis backend. This has allowed us to simplify the Frontend (and Squonk2) processing. - The squonk-job-config.yaml definition has been updated. - The apo-desolv.pdb must be stored in the database/exported to Squonk2.	
1.0	Added session project to the *job_request* table for ISPYB check on results for private projects.	

High level GUI Task:

https://github.com/m2ms/fragalysis-frontend/issues/791

Design Task:
https://github.com/m2ms/fragalysis-frontend/issues/793

Introduction

https://github.com/m2ms/fragalysis-frontend/issues/791 outlines high level requirements for running jobs in the Squonk 2 ecosystem from the Fragalysis frontend.

This document outlines the work required to the Fragalysis frontend (React Viewer), Fragalysis backend (Database and API) and Squonk2 to be able to launch and track Squonk2 jobs from Fragalysis.

Terminology

Squonk2 also has a UI frontend (React) but with a NextJS server side component and a Flask-based backend known as the Data Manager. For clarity I've stuck to Squonk2 throughout this document.

High Level Changes

At a high level (for technical detail see the Solution Outline and following sections) the process will be as follows:

1. A default project for each target will be defined in Squonk2 in which the jobs will be run. This will be configured in Fragalysis.
2. The user logs on (only logged on users should be able to start jobs. The Fragalysis user will also be the user for Squonk2 using keycloak/single sign-on).
3. The frontend creates/selects a snapshot, based on a subset of the molecules (LHS and in future RHS).
4. The frontend calls the backend to initiate an asynchronous copy of the files related to the snapshot from Fragalysis to the Squonk2 project controlled by a new *job_file_transfer* table.
5. To launch a job, the job is selected on the Job launcher pane in Fragalysis. This will open/extend the Job launcher pane by using a *squonk_job_config.yaml* file (provided to the frontend repo by IM with the parameters Squonk2 needs to launch jobs) to flexibly construct an options dialog in Fragalysis using a React Forms library.
6. When the launch button is pressed, Fragalysis will check that the file copy has completed successfully (and in future that this is not a repeat request) and then Squonk2 will be called from Fragalysis (backend) to start the job - the request containing a generated callback URL - and than a job record will be created in Fragalysis to track the job.
7. Squonk2 will use the callback URL to update the status of the job. The job record will also provide the Fragalysis frontend with a Squonk Task-id that can be used to create a URL that can be used to navigate to the Squonk2 frontend to view progress/results.

Although **not** part of the first phase of coding, functionality to automatically upload results back into Fragalysis have been considered. These are shown below:

8. Once a job has finished then, if the job has been configured for the automatic upload of results (of a compatible SDF file), the Fragalysis backend will call Squonk2 to retrieve the results file(s), add any metadata from the job (e.g. PDF or fragments) and upload them into Fragalysis using the Computed sets upload process.
9. If the *job_request* record is subsequently queried from the frontend and a compatible (SDF) results file has been uploaded then additional information on these new compounds can be provided to the frontend.
10. The frontend can then regenerate the snapshot with the results compounds from the SDF selected to show the differences (e.g. in the 3D viewer)
11. The existing computed set functionality should also be modified to allow deletion of existing computed sets for logged-in users. This will allow the computed sets to be managed following the upload.

Assumptions:

1. It will only be possible for the users *who are logged on* to run jobs. Otherwise any APIs developed could potentially be used maliciously.
2. We will assume, for the sake of simplicity, that a Project has already been created in Squonk2 for the relevant target in Fragalysis.
3. Creation/management of groups will primarily be handled by the existing snapshot functionality on the frontend. This allows molecules from both left and right to be included.
4. Jobs involving different combinations of the same molecules will be done using Squonk2 workflows. For example: A job involving processing the same 4 molecules in different combinations would involve a single request with parameters to indicate the processing required (e.g. combination of 4 molecules or pairs of molecules among the 4). Squonk2 will have workflows defined for doing this.
5. In general, the fragalysis front end will have to call the backend to refresh with status updates. We discussed that it would be beneficial to do this automatically every minute or so. This is not covered further in the document as it is a frontend only issue.

Limitations:

1. We initially propose only setting up a *default_squonk_project_id* for each target so all jobs for this target will run under the same Squonk project. This can be expanded at a future date to have multiple projects within a target.
2. Transfer of files to Squonk2. Initially we are planning to use the current Squonk2 API which takes a single file - on the assumption that the network will be the primary bottleneck rather than number of API calls. If this proves too slow in future to transfer large numbers of files in a snapshot then improvements can be made to speed up the process. For example: transfer a zip file of files so we can “chunk” the file transfer or to place the files in a shared location and then queue Squonk2 to import them into the correct location in Squonk.
3. In phase 1 for simplicity only files from the LHS of the Fragalysis viewer will be chosen for transfer to Squonk2.
4. In phase 1 for simplicity there will (probably) be no check on duplicate job requests, but we will design the process so that this can be easily added later. The idea is that if the same request is queued twice then a suitable message will be shown, or, if the job has already finished, the user will be directed to the existing results.
5. In phase 1 for simplicity the *squonk_job_config.yaml* will contain the complete configuration. However ideally the definition for a job could be retrieved from Squonk2 directly and then updated with Fragalysis-specific options.

Use Cases:

UC	Description
1	Configure environment parameters for communication between Fragalysis and Squonk2. Configure project in Squonk2 for the Target and update in Fragalysis. Configure *squonk_job_config.yaml*
2	Job Submission: - Frontend: Logon - Frontend: Select molecules from LHS/RHS - Backend: Transfer files (async) - Frontend: Select job - Frontend: Create job launcher pane from squonk_job_config.yaml - Frontend: Select files + options - Frontend: Call Backend to create/launch job - Backend: Check file transfer - Backend: Post job in Squonk2 with generated callback URL - Squonk2: Call Backend to update job record job status STARTED. - Squonk2: Call Backend to update job record job status SUCCESS/FAILURE. - Backend: Backend: (not phase 1) If applicable, initiate automatic upload of results SDF file from Squonk2 into Fragalysis. - Frontend: check job status - receives status of success.
3	Job status update: - Frontend: Request status update for job (user/target/snapshot/id) - Note that this will be initiated from the Frontend. - Backend: Returns details for job with URL. - Frontend: User has option to navigate to Squonk2 to view details
4	Job completes successfully in Squonk2: - Frontend: Request status update for job (user/target). - Backend: Returns details for job including ability to jump to Squonk2 to view job execution details. - Frontend: (not phase 1) show Success dialog. User selects to “see results”. Results snapshot created and user redirected to uploaded molecules in 3D viewer.
5	Job completed unsuccessfully in Squonk2: - Squonk2: Calls Fragalysis backend Update Link with status update (*FAILURE, RETRY, REVOKED*) - Frontend: Request status update for job (user/target). - Backend: Returns details for job. - Frontend: User has option to navigate to Squonk2 to view details
6	User attempts to queue the same job more than once (Phase > 1) - If the job is processing or finished, job details will be returned and a URL to the processing/finished job details will be provided. - If the job has not yet started (for example, waiting for files to transfer), then an error message will be returned.

7	Upload New Target data replacing some files in the snapshot in Fragalysis that have already been transferred to Squonk2. - Files connected to a snapshot are transferred to Squonk2 - A new version of the Target is uploaded which updates these files. - *The next time a job is run for the snapshot, the file transfer will be rerun (See note 1).*
8	Data is cleaned from Squonk2. User in Fragalysis navigates to Squonk2 to view old data. - *Processing data (e.g. nextflow work files) will be cleaned from Squonk2 after a period of time.* - *Results data will be saved indefinitely.*
9	Delete computed set (not phase 1): - User logs on and navigates to the upload_cset page (details tbc) - User can view sets that they own and anonymous sets. - User can update/delete computed sets that they own.

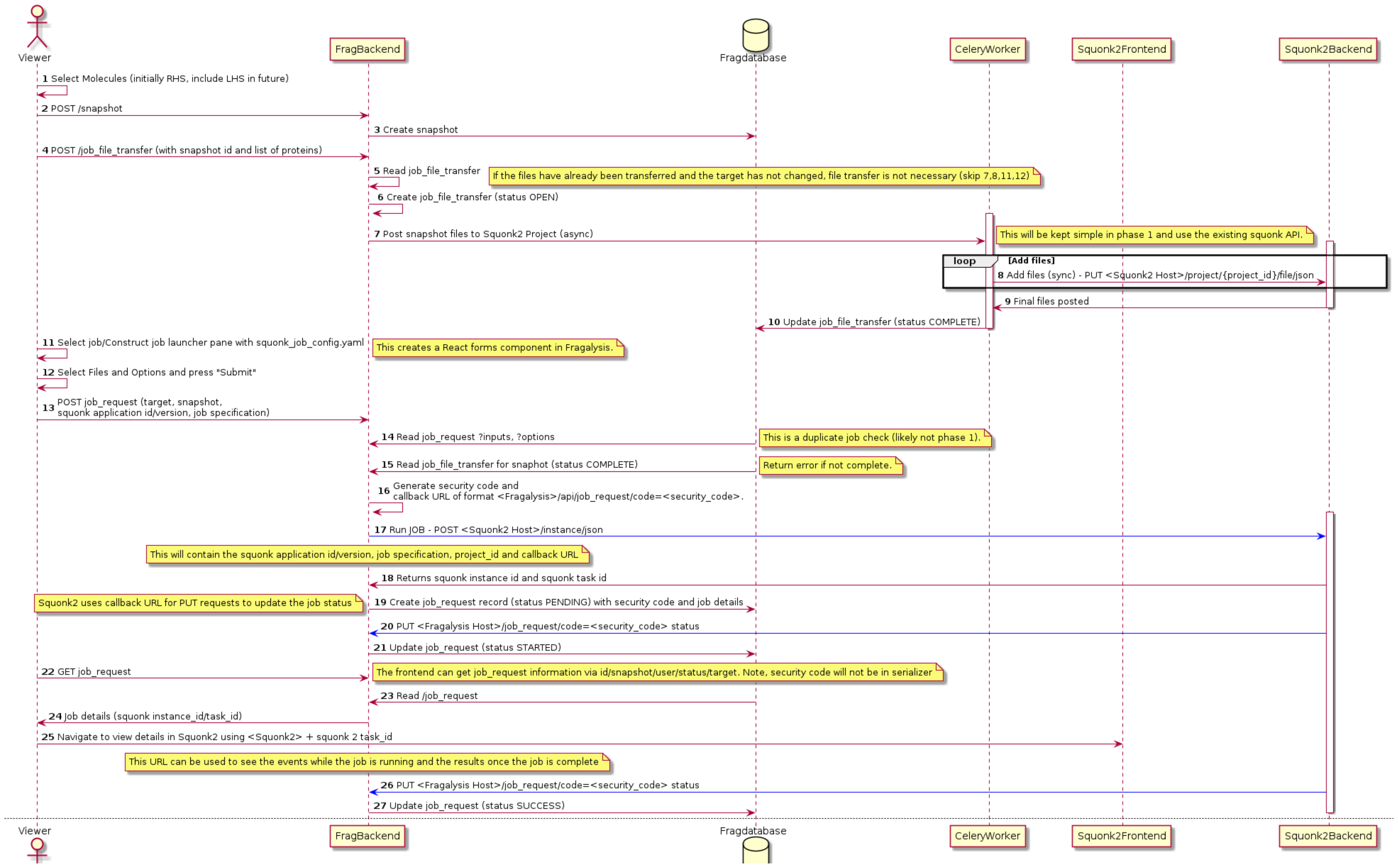
Notes:

1. This can be improved in future to, for example, add versioning of files that have been transferred to Squonk. However this will require changes to the Fragalysis target data loader (see the Data Upload Issue)

Solution Outline

Job Submission Process Flow

The figure below shows the in-scope end-to-end process for job-submission up to the future automatic processing.



1 Select Molecules (initially RHS, include LHS in future)
2 POST /snapshot
3 Create snapshot
4 POST /job_file_transfer (with snapshot id and list of proteins)
5 Read job_file_transfer
If the files have already been transferred and the target has not changed, file transfer is not necessary (skip 7,8,11,12)
6 Create job_file_transfer (status OPEN)
7 Post snapshot files to Squonk2 Project (async)
This will be kept simple in phase 1 and use the existing squonk API.
8 Add files (sync) - PUT <Squonk2 Host>/project/{project_id}/file/json
9 Final files posted
10 Update job_file_transfer (status COMPLETE)
11 Select job/Construct job launcher pane with squonk_job_config.yaml
This creates a React forms component in Fragalysis.
12 Select Files and Options and press "Submit"
13 POST job_request (target, snapshot,
squonk application id/version, job specification)
14 Read job_request ?inputs, ?options
This is a duplicate job check (likely not phase 1).
15 Read job_file_transfer for snaphot (status COMPLETE)
Return error if not complete.
16 Generate security code and
callback URL of format <Fragalysis>/api/job_request/code=<security_code>.
17 Run JOB - POST <Squonk2 Host>/instance/json
This will contain the squonk application id/version, job specification, project_id and callback URL
18 Returns squonk instance id and squonk task id
Squonk2 uses callback URL for PUT requests to update the job status
19 Create job_request record (status PENDING) with security code and job details
20 PUT <Fragalysis Host>/job_request/code=<security_code> status
21 Update job_request (status STARTED)
22 GET job_request
The frontend can get job_request information via id/snapshot/user/status/target. Note, security code will not be in serializer
23 Read /job_request
24 Job details (squonk instance_id/task_id)
25 Navigate to view details in Squonk2 using <Squonk2> + squonk 2 task_id
This URL can be used to see the events while the job is running and the results once the job is complete
26 PUT <Fragalysis Host>/job_request/code=<security_code> status
27 Update job_request (status SUCCESS)

Steps:

Create Snapshot/List of Proteins

1 - 3: The frontend creates/selects a snapshot, based on a subset of the molecules. Initially this will generate a list of Protein codes from the LHS - but can be extended to include compounds from the RHS. As this is similar to current functionality (e.g. selecting proteins for the existing *download_structures* API), no more detail will be given in this document.

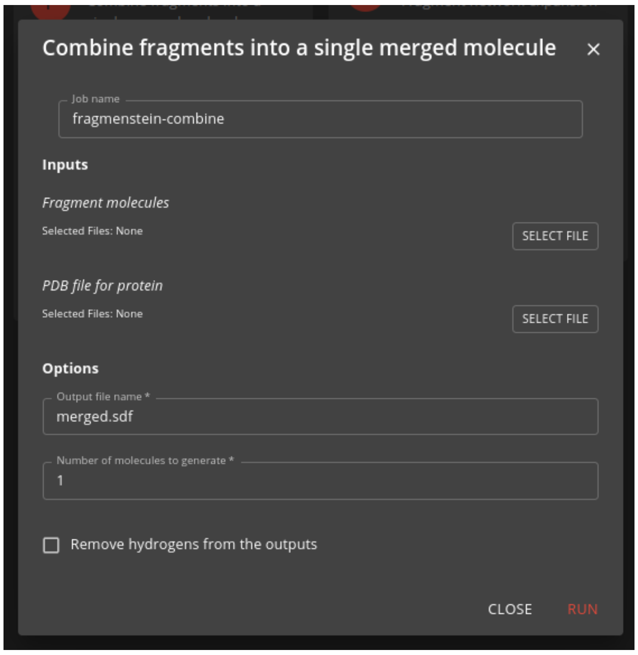
File Transfer

4 - 8: The frontend POST to the new job_file_transfer API with a list of protein_codes (and in future compounds) to initiate a celery task to copy the relevant files related to the snapshot from Fragalysis to the Squonk2 project (which will be provided to the frontend in the target information). These will be used for **all** jobs initiated from that snapshot. Only applicable file types will be copied and only files that don't already reside in the project in squonk.

9 - 10: Once the file transfer has finished, the status will be updated to COMPLETE. A status of COMPLETE is a precondition for a job to be submitted.

Launch Job

11 - 13: To launch a job, the job is selected on the Job launcher pane. This will open/extend the Job launcher pane with options based on the Squonk_job_config.yaml file that will be provided to the frontend developers by IM. The file will be used to build a job launcher dialog using a React forms library in Fragalysis in a similar way to how the Squonk2 UI currently works. The configuration will define: how many and what types of file are required for the job and any other necessary. For example: The Squonk2 fragmenstein-combine job requires 2 or more mol files and a pdb file. An example of this window is below:


Combine fragments into a single merged molecule

Job name: fragmenstein-combine

Inputs

Fragment molecules
Selected Files: None

PDB file for protein
Selected Files: None

Options

Output file name *: merged.sdf

Number of molecules to generate *: 1

☐ Remove hydrogens from the outputs

When the launch button is pressed the frontend will POST to the new job_request api to create the job. This starts the handshake process with Squonk2 from the Fragalysis Backend.

Note: the *squonk2_job_config.yaml* file process as listed here is intended to be a short term solution for the proof of concept. A better longer term solution would be to call Squonk2 directly to get the job parameters and then the squonk2_job_config.yaml file would contain only additional Fragalysis specific options (such as the automatic output definition).

14 - 19: To launch a job, the Fragalysis backend will check/do the following:

- That the same job has not been queued before (by checking the job_request table using the job, inputs and options from the job submission window (not for phase 1)
- That the files for the job have been successfully transferred over to Squonk2 (by checking the job_file_transfer table).
- Create a security code and generate a callback URL that will be used by the Squonk2 to update the status of the job. The Callback URL will be of form: *<Fragalysis Host>/api/job_request/code=<security_code>.* This security code is specific to the job and allows Squonk2 to make updates in a relatively safe way via the API without having to authenticate.
- Call the POST <Squonk2 Host>/instance/json API with the details of the job to be started: This will contain the *squonk application id/version* and *job specification* (inputs and options), squonk2_project_id, Callback URL. Note that the Squonk2 Host and *Fragalysis Host* will be environment variables, allowing multiple (development) systems to call a single Squonk2 instance and squonk to be able to reply to the correct version of Fragalysis.
- A successful reply from Squonk2 will contain the job_instance and task_instance. These will be used to create a new record in the job_request table using the target, snapshot, job, inputs and options, and the instance_id and task_id from squonk2. The initial status will be "PENDING".

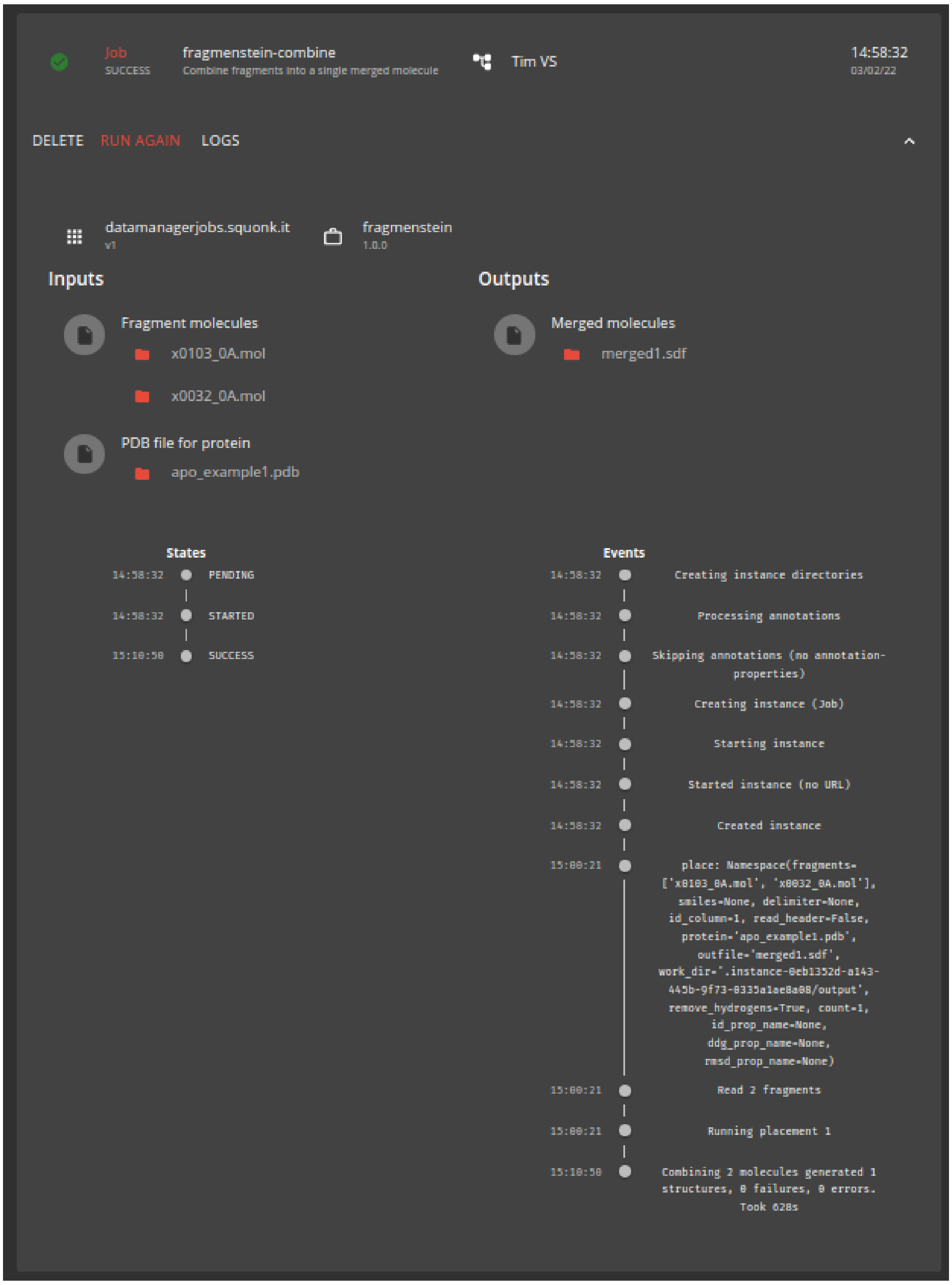
Call Back From Squonk2

20 - 21: Once a job has been started in Squonk2, it will use the Callback URL to PUT to the Fragalysis Backend using the new job_request API to update the status to *STARTED*.

26 - 27: Squonk2 will monitor the job and use the callback URL to notify the backend of the final status of the job (*PENDING, STARTED, SUCCESS, FAILURE, RETRY, REVOKED)* when it has finished.

Monitoring Squonk2 Jobs

22 - 25: Once the job request has been created, the *squonk_task_id* will be available to the Fragalysis frontend. The Fragalysis frontend can use this to create a clickable link to navigate to squonk directly to view detailed job progress on the Squonk2 task window (see below) and the results when the job has finished.



Job
SUCCESS

fragmenstein-combine
Combine fragments into a single merged molecule

Tim VS

14:58:32
03/02/22

DELETE RUN AGAIN LOGS

datamanagerjobs.squonk.it
v1

fragmenstein
1.0.0

Inputs

Fragment molecules
x0103_0A.mol
x0032_0A.mol

PDB file for protein
apo_example1.pdb

Outputs

Merged molecules
merged1.sdf

States

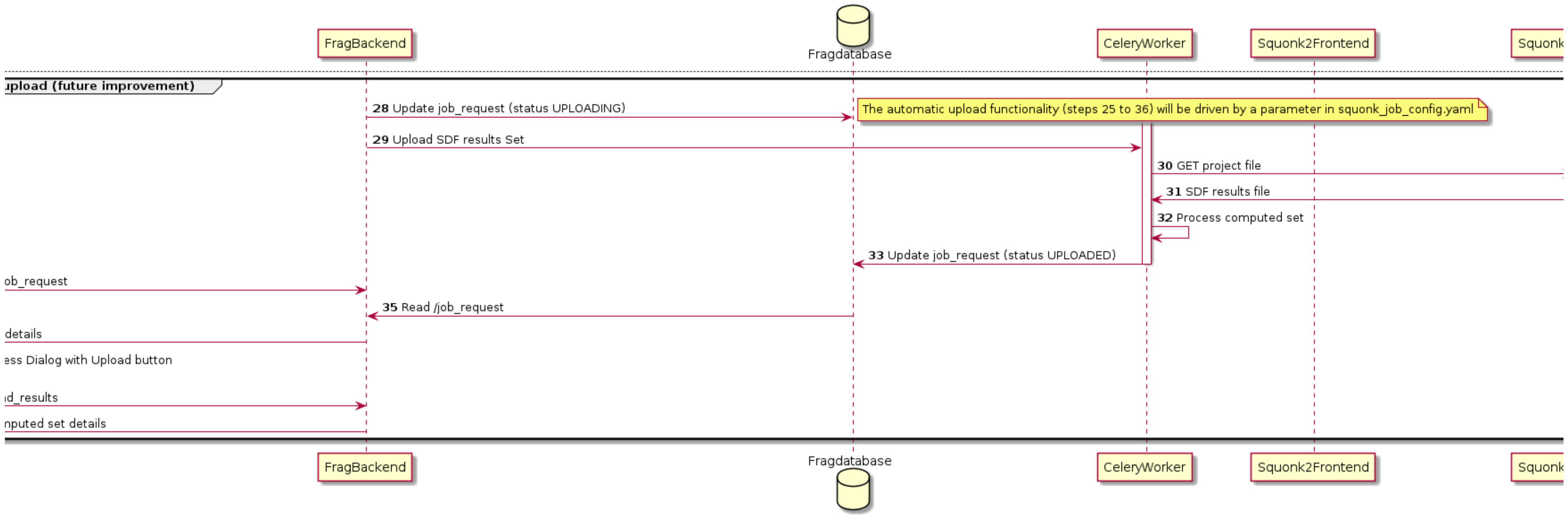
```
14:58:32  PENDING
14:58:32  STARTED
15:10:50  SUCCESS
```

Events

```
14:58:32  Creating instance directories
14:58:32  Processing annotations
14:58:32  Skipping annotations (no annotation-
          properties)
14:58:32  Creating instance (Job)
14:58:32  Starting instance
14:58:32  Started instance (no URL)
14:58:32  Created instance
15:00:21  place: Namespace(fragments=
          ['x0103_0A.mol', 'x0032_0A.mol'],
          smiles=None, delimiter=None,
          id_column=1, read_header=False,
          protein='apo_example1.pdb',
          outfile='merged1.sdf',
          work_dir='.instance-0eb1352d-a143-
          445b-9f73-0335a1ae8a08/output',
          remove_hydrogens=True, count=1,
          id_prop_name=None,
          ddg_prop_name=None,
          rmsd_prop_name=None)
15:00:21  Read 2 fragments
15:00:21  Running placement 1
15:10:50  Combining 2 molecules generated 1
          structures, 0 failures, 0 errors.
          Took 628s
```

For completeness, the figure below shows the proposed automatic upload extension (not phase 1).

Automatic Upload


FragBackend
Fragdatabase
CeleryWorker
Squonk2Frontend
Squonk
ıpload (future improvement)
28 Update job_request (status UPLOADING)
The automatic upload functionality (steps 25 to 36) will be driven by a parameter in squonk_job_config.yaml
29 Upload SDF results Set
30 GET project file
31 SDF results file
32 Process computed set
33 Update job_request (status UPLOADED)
ob_request
35 Read /job_request
details
ess Dialog with Upload button
ıd_results
nputed set details

Automatic Upload (not for phase 1)

28 - 33: Once a job has finished (the status has been updated to SUCCESS) then, if the job has been configured for the automatic upload of results (of a compatible SDF file), the Fragalysis backend will call Squonk2 to retrieve the results file(s), add any metadata from the job (e.g. PDF or fragments) and upload them into Fragalysis using the Computed sets upload process. This is another async job, when it completes the status will be finally changed to UPLOADED.

34 - 36: If the *job_request* record is subsequently queried from the frontend, then a summary of the results can be provided. Furthermore, if a compatible (SDF) results file has been uploaded then additional information on these new compounds will be provided.

37 - 38: The frontend can then regenerate the snapshot with the results compounds from the SDF selected to show the differences (e.g. in the 3D viewer)

39: The existing computed set functionality will be modified to allow deletion of existing computed sets for logged-in users. This will allow the computed sets to be managed following the upload.

Fragalysis Frontend

Changes described here will only refer to basic functionality to clarify the flow of information. Look and feel is out of scope of this document.

The Fragalysis frontend will handle (I may have missed some things - for details see the Front end issues):

- Selecting proteins and ligands to be kept in a snapshot (existing functionality)
- Selecting proteins and ligands to add to a file transfer and check whether the transfer has succeeded.
- Creation/processing of the job launch pane based on *squonk_job_config.yaml* (see below). From a technical perspective this will (at least in part) use the same library to create the options dialog (the bottom part of the dialog) as Squonk2. The ways the inputs are selected (the top part of the dialog) will be specific to Fragalysis
- Post job instance request to the Fragalysis Backend
- Functionality to monitor *job_request* records created/updated by Squonk2.
- Create URLs to the Squonk2 task window (based on the *job_request.squonk_task_id* and *django_context*).
- Automatic results upload functionality (not for phase 1).

Job Launch Pane

Squonk_job_config.yaml

The *squonk_job_config.yaml* file will be defined by IM with Squonk2 job configuration information but held in the Fragalysis frontend repository to enable the job launcher pane to be built and the job to be posted via the Squonk2 POST instance API call.

A precise schema will be defined in the detailed task but here following is an example of the information based on the **Fragmenstein combine job definition.**

The values that will be in the *squonk_job_config.yaml* are shown below in red and blue. These will correspond to the groups in the *job_request* table in fields *inputs* and *options.* The idea is that a check can be made against these fields to see if a job has been run previously.

In red, the definition of the inputs. In this case:

- **fragments** that can be multiple molfiles
- **protein** that is a single PDB file

In fragalysis we would need a configuration that points to this job definition and provides any "missing" information e.g. that the **fragments** must be fulfilled by fragments on the LHS and the **protein** must come from a protein from the LHS. Note, we will probably enhance the definition to allow the multiplicity of inputs to be better defined. e.g. rather than just stating that the **fragments** input is *multiple*, we allow minValues and maxValues to be specified so that Fragalysis knows that there must be between 2 and 20 molecules to be specified.

In blue, the user specified options, defined as a JSON schema document. The options the user chooses are recorded, but Fragalysis may not mandate any particular ones. The UI for these options is rendered automatically using the Forms library.

```
# Host will be an environment parameter so that this definition can be location independent
sdm-host: https://squonk.xchem.diamond.ac.uk/data-manager-api/api/
# These paths are effectively constants so could be hardcoded
# Path to the URL that GETs the full list of job definitions
sdm-jobs-path: job
# The job is launched with a POST request to the instance endpoint
# Executing the job returns a job instance ID
sdm-launch-path: instance
# Details of the job can be obtained from this path
sdm-instance-path: instance/{instance_id}
# Some other endpoints probably needed e.g. file upload
jobs:
  # Each job that we want to be available is defined here
  # This is small subset of the total list of jobs
  fragmenstein-combine:
    collection: fragmenstein
    version: 1.0
    inputs:
      fragments:
        from: lhs-fragments
        format: chemical/x-mdl-molfile
      protein:
        from: lhs-protein-apo-desolv
        format: chemical/x-pdb
    # These are options that Fragalysis can specify
    options:
      # Jobs need to provide the ability for outputs to be placed in a particular directory
      # so that each job is isolated. {job_dir} is a Fragalysis defined location for the
      # job’s results.
      outfile:
        value: fragalysis-jobs/{username}/{job_dir}/merged.sdf
        editable: false
      # For other options we can override the job’s default values
      count:
        value: 5
    # These outputs can be incorporated as RHS datasets
    outputs:
      merged-molecules:
       title: Merged molecules
       # probably some additional parameters for how to load
```

Notes:

- For cases where **multiple combinations** of inputs are needed (e.g. user has selected 3 fragments and wants every pair combination to be executed, this is much better handled by the workflow e.g. user will specify the 3 ligands to submit that to the job. The job will generate the 3 combinations of pairs (1+2 1+3 2+4) and perform an execution for each, resulting in 3 sets of results. This sort of logic is much better handled by the workflow than in Fragalysis.

Fragalysis Backend

Description

The Fragalysis backend will handle the communication between the frontend and Squonk2. It will provide:

- Updated django_context with the URL to the connected Squonk2 (supplied in a Django setting)
- New API: *job_file_transfer* to initiate an asynchronous transfer of snapshot files to Squonk2 (via a celery task). The transfer of files will be tracked by a new table.
- New API: *job_request* to track jobs running in Squonk2 (see below):
- Improved functionality to the upload_cset computed set functionality. Logged on users will be able to delete computed sets that they own (not for phase 1).
- Some improvements to the target upload functionality are required to:
    - Store the *apo-desolv.pdb* file in the Fragalysis database
    - Allow the file transfer processing to check if a target has recently changed.

File Transfer

The frontend POST to the new job_file_transfer API will contain a list of protein_codes (and in future compounds) in a similar way to the existing download_structures API. This will initiate a celery task to copy the files related to the snapshot from Fragalysis to the Squonk2 project (which will be provided to the frontend in the target information). These files are copied to a directory that is specific to that target.

The celery task will do the following:

- From the list of given protein codes, the *viewer_protein* and *viewer_molecule* table will be consulted to find the (latest) version of the following file types:

Type:	Reference:	Target Upload File location:
pdb	viewer_protein.apo_desolve_info	<protein_code>/<protein_code>.apo-desolv.pdb
mol	viewer_molecule.sdf_file	<protein_code>/<protein_code>.sdf
smiles	viewer_molecule.smiles	Extracted from: <protein_code>/<protein_code>.sdf.

- **Important:** For various reasons historical targets may not contain the mol file and the apo_desolv.pdb file will only be available when the target is reloaded (see below). If any of the above files are not present then the *file_transfer* will be updated with a status of FAILED.

- Using the *<Squonk2 Host>/project/<project_id>/file* API with authentication details from the user Squonk2 will be checked to see if the selected files already exist on the project volume..
- For any missing files, the *<Squonk2 Host>/project/<project_id>/file* API will be used to transfer them over file by file. During processing, *file_transfer* record will be updated to a status of

PROCESSING. If there is a problem, then the *file_transfer* record will be updated to a status of FAILED. The status of the file transfer will be recorded in a table and this can be polled for the frontend to track progress.

- If the target is subsequently updated, then - when the POST *job_file_transfer* API is called for the snapshot, the *trasfer_datetime* will be compared to the (new) target *last_updated_datetime* and the files resynchronised if necessary. A resynchronisation will cause all files in the snapshot to be replaced in Squonk2. Once the "Fix data upload" epic has been implemented we may want to improve this by versioning files so that old files versions remain available.

Once the file transfer has finished, the status will be updated to COMPLETE.

Target Upload - apo-desolv

For the proof of concept to work with the Fragmentstein combine job, changes must be made to the target upload code to allow the *<protein_code>.apo-desolv.pdb* file to be uploaded from the aligned directory.

Notes:

1. For historical reasons it is extremely complicated to convert existing datasets (there have been three distinct generations of target-loaders with different characteristics). Some thought has gone into addressing this as part of the fix data upload story. However for now, the only way the *apo-desolv.pdb* files will become available will be via **reloading the target.**
2. At a future date the new file type can relatively easily be added to the *download_structures* api, but this is not currently in scope for this change.

Automatic Upload (Not phase 1)

After the SUCCESS state has been received by Fragalysis, the job request parameters will be consulted to see whether an automatic upload has been requested.

In the case of an automatic upload:

- The status will be changed to 'UPLOADING'
- A celery task will be created that will GET the configured results from Squonk2 and use the existing computed set functionality to upload the file back into Fragalysis as a new computed set.
- On completion of the upload, the status in the job_request will be changed to "UPLOADED" and the table updated with a reference to the computed set.

Data Model

Table: Job_request (new)

Store details of jobs running on Squonk2

- *id*
- *job_name (unique)*
- *snapshot_id (fk)*

- *project_id (fk)*
- *target_id*
- *squonk_project_id*
- *user* : user who requested the job.
- *inputs (JSON)* : Inputs chosen by the user for this job.
- *options (JSON)* : Options chosen by the user for this job.
- *job _status (enum: OPEN, PENDING, STARTED, SUCCESS, FAILURE, RETRY, REVOKED, UPLOADING, UPLOADED)* : The status of the Squonk2 job.
- *squonk_task_id* (string): Used for generating the URL to navigate to the task window.
- *squonk_job_info (JSON)* :
- *code (string - unique)* : UUID generated by Fragalysis and used in the callback URL to enable Squonk2 to update the Job details.
- *computed_set (fk)*: id of the uploaded computed set (if applicable)

Table: Job_file_transfer (new)

Store details of files being transferred to Squonk2 for a job. Note that this is necessary to track the progress of the asynchronous transfer jobs, but also store a snapshot of the files transferred in case they are changed when a target is reloaded.

- *id*
- *snapshot_id (fk)*
- *project_id (fk)*
- *squonk_project_id*
- *proteins* (JSON)*:* A list of of the protein_codes to be transferred
- *compounds* (JSON)*:* A list of of the compounds to be transferred (not phase 1)
- *transfer_status* (enum: *PENDING, STARTED, SUCCESS, FAILURE*)
- *transfer_progress* (percent): Potentially a very approximate progress value if can be sensibly filled (it depends on the details of the backend celery worker implementation)
- *transfer_datetime*

Viewer_target (existing)

- *default_squonk_project_id* (text): This is the squonk project id that the jobs will be run under. It must be configured for jobs to run (How do we do this the least painful way?)
- *last_updated_datetime* (datetime): Updated by the target upload functionality when a target has uploaded successfully. This allows other processes to see when the target data has changed.

Viewer_protein (existing)

- *apo_desolve_info* (file field): Updated by the target upload functionality when a <protein_code>.apo-desolv.pdb file is found in the target directory.

Viewer_computedset (existing)

- *owner_user_id* (FK): This is the id of the user who uploaded the computed set

API Changes

API Job_file_transfer (New)

POST

Fields:

- *snapshot_id*
- *project_id*
- *squonk_project_id*
- *proteins* (JSON)*:* A list of of the protein_codes to be transferred
- *compounds* (JSON)*:* A list of of the compounds to be transferred (not phase 1)

GET

Filter by:

- *id*
- *snapshot_id*
- *project_id*
- *squonk_project_id*
- *status*

ISPYB authorization.

The *job_file_transfer* API will make use of ISPYB safe querysets, so that a User will only be able to transfer files related to protein codes that they are authorised to view.

API job_request

POST/PUT

Fields:

- *job_name*
- *snapshot_id*
- *project_id*
- *inputs*
- *options*
- *job _status*
- *squonk_task_id*
- *squonk_job_info*
- *code*

GET

Filter by:

- *Id*
- *snapshot_id*
- *project_id*
- *user*
- *squonk_project_id*
- *job _status*

- *job_name*
- *inputs*
- *options*
- *squonk_task_id*
- *squonk_job_info*

Authentication

Fragalysis Users will be expected to be logged in when using the *job_request* API.

When a job is launched in Squonk2 from Fragalysis, a security code unique to the job will be generated and supplied to Squonk2 in the form of a callback URL. Squonk2 will use this URL to make status updates in Fragalysis. This means Squonk2 will not need to authenticate with Fragalysis, simplifying the solution.

Fragalyis will be making use of a *lookup_url_kwarg* to check the security code. If the URL contains a kwarg of 'code=<security_code>' then the job related to the security code is updated and no user authentication is performed. In all other cases where the table will be updated, the user will be expected to be logged in.

ISPYB Authorisation

Results in job_request will be filtered by the *project_id* so that jobs in private projects cannot be seen by other users.

API target_molecules

Will be extended with the new *viewer_target* fields: *default_squonk_project_id* and *last_updated_datetime.*

Squonk2

Description

Squonk2 will handle all job related activity and will store detailed information on jobs that are currently processing or completed.

The Fragalysis backend will communicate with Squonk2 in the following cases:
1. Transfer of file information relating to a job that is about to be launched
2. Creation of a job instance
3. Receive job information at the start of a job.
4. Receive status update information from when the job has finished.

The Fragalysis will link directly with Squonk2 in the following cases:
1. Navigating to the Squonk UI task directly to get details of a running job or to see results of a finished job.

Create a Job

Jobs have an ID uniquely identified by three properties: “collection” (str), “name” (str), “version” (str).
The *instance* is responsible for managing the container execution
The *task* is responsible for delivering state updates

The initial POST call to the *instance* API will contain the additional following field:

- A Callback URL of the form: *<Fragalysis Host>/api/job_request/code*=*<security_code>* that will be used by Squonk2 to update Fragalysis with the job status.

To aid tracking and enable concurrent job executions by different users on the same file, Job executions should be run in a specific subvolume as follow:

- ***/<Fragalysis username>/<datetime>***

The datetime will be of format ‘YYYYMMDD HHMMSS’

A successful POST instance call should result in a reply (existing functionality) with the following job information which will be stored in the in the *squonk_job_info* field in the *job_request* table with the status.:

```
{
    “job-execution”: {
        “job”: {
            “collection”: “fragmenstein”,
            “version”: “1.0.0”,
            “name”: “fragmenstein-combine”
        },
        “user_context”: “snapshot-123456789”,
        “project_id”: “project-57930f19-51ee-4d86-a955-52677b84f0b5”,
        “instance_id”: “instance-25ec94c6-ad7b-4d7d-9ac2-0f8c51e7e15b”,
        “task_id”: “task–82608f84-6dae-4a52-97bc-655411fc2298”,
        “stateTransitionTime”: “2022-02-09T17:28:37Z”,
        “state”: “PENDING”
    }
}
```

Call Back

Squonk2 job monitoring functionality should then use the callback URL to make two following PUT calls to Fragalysis:

- The first PUT request will be made when the job is in the *STARTED* state (see below)
- A second PUT request will be made when the job completes hopefully with a SUCCESS state.

Job Processing

A successful state sequence will consist of at least 3 messages (1 POST and 2 PUT): -
PENDING -> STARTED -> SUCCESS
Failures will consist of at least 2 messages for a Job that has failed to start: -
PENDING -> FAILURE
or 3 for a Job that has started and has failed: -
PENDING -> STARTED -> FAILURE

Ultimately Fragalysis needs to wait for PUT messages with a SUCCESS or FAILURE state.

The Squonk2 server will need a **resilient callback delivery mechanism** (with timeout) that stores instance status updates until they've been successfully delivered to the URL. URL delivery (including failures) should be recorded as an "Event" against the Instance "Task".

- If the Squonk2 fails to deliver state updates to the callback after "N-attempts" it will cease using the callback and create a suitable warning "Event".

The number of retries should be **parameterizable** so we can adjust it based on experience.

API Changes

API project/<project_id>/file

No changes will be required to this API in phase 1. It will be used for posting files into a project by the snapshot file transfer process. It may need to change if a more efficient process is required - several suggestions have been made in earlier versions of this document.

API instance (Changed)

POST

Fields:

- Call back URL of the form: '<Fragalysis Host>/api/job_request'

Non-functional Requirements

Environment and Process Changes

As a minimum Squonk2 must be set up in the Diamond development and production clusters - although the cluster available on the development stack only needs to be of a sufficient size to run basic jobs to test the functionality.

The development Squonk2 instance will be pointed to from the following locations:

- Local stacks (on a developers laptop)
- Multiple Development stacks

Outstanding point:

- Do we need a staging Squonk2 AND a production Squonk2?

All URLS should be relative (if available).

We have the Discourse implementation here as a previous example.

For this, it was necessary to create environment variables within the Fragalysis Backend with the applicable URL and provide these to the Front end in the Django context.

Authentication/Authorization

The user will be logged on to Fragalysis via Keycloak.

When a job is launched in Squonk2 from Fragalysis, a security code unique to the job will be generated and supplied to Squonk2 in the form of a callback URL. Squonk2 will use this URL to make status updates in Fragalysis. This means Squonk2 will not need to authenticate with Fragalysis, simplifying the solution.

Through ISPYB authorisation, the user will have access to both public datasets and molecules that they are allowed to view. Hence they can potentially create jobs for “private” molecules.

Performance Criteria And Sizing

The response time for creating a job in Squonk2 in the job launcher pane should be less than approximately 10 seconds.
The response time for enquiring on high level job execution data (such as that shown in the project pane) should not cause lagging in Fragalysis.

Testing and Deployment

Steps to implement to avoid any problems
To be investigated separately

Note that one critical step will be to **reload the target** that will be used for the test so that the apo-desolve.pdb files are added.

Clean up of Old Data

After a period of time, the temporary run data (Nextflow work files etc) can/should be cleaned up, However, the actual results should continue to be available to Fragalysis for enquiry.

Documentation

This document (and the *puml* sequence diagram) will be added to the Fragalysis Backend Repository.

Roadmap

This section contains suggested tickets with rough time estimates. If the scope changes significantly these estimates would have to be revisited:

Ticket	Description	Est. IM	Est. M2M
1	Plumbing: 1. Get Squonk2 set up in the development and production clusters. 2. Configure Fragalysis Backend settings to be able to call Squonk in the development cluster. 1. SSO between Fragalysis/Squonk2 (Alan)	1 day Half day	
2	Job configuration: 1. Which jobs are going to be run initially 2. Definition of yaml for Fragalysis frontend window to be able to run those jobs. This will be stored in the job configuration table. Confirmation of where this is stored and how the change management will work (a yaml file?) 3. Support Fragalysis frontend in creation of job submitter window to be able to enter parameters. (Tim, Boris)	2 days 2 days	
3	Fragalysis Backend changes to be able to configure and run jobs: 1. Finalise detail in design following job config/outstanding questions answered. 2. Configuration of squonk project and set up squonk config parameters in Fragalysis. 3. File transfer 4. Job configuration and job request (Duncan)	Approx 12 days	
4	Squonk2 backend changes: 1. Accept Fragalysis callback URL & test 2. Resilient call-back mechanism to send status updates back to Fragalysis (Alan)	Half day 3 Days	
5	Automatic upload of results in Fragalysis. 1. Changes to computed set functionality to allow deletion of sets for logged in users 2. Kick off import of results sets automatically at the end of a job. (Duncan)	Approx 6 days	

Notes:

- Estimate for 5) includes introduction of basic automatic tests for compound set upload in a similar way to target set upload.

Appendices

Compound Set Deletion

Original Solution Roadmap (as discussed with Ruben on 01/12/2022

– back end only is sufficient at this stage.

1. Change update/upload_cset to be for logged-in users only (like for upload_tset). This will probably involve adding the user_id to the ComputedSet model. Current computed sets will be updated so that they are attached to the anonymous user (part of the migration).
2. On the upload and update cset pages, anonymous computed sets will be visible, but you can only update/delete sets that you own.
3. Check the functionality of update_cset page works. Aim is to add a delete boolean to this screen so you can select a dataset and delete it. If not I'll create a delete_cset page – since it won't need the celery task!
4. Should be sufficient to delete the ComputedSet record for the user.
5. Can add the email notification in if it's free (which it is more or less)

Add User to ComputedSet
Make anonymous computed sets available to be seen by everyone – but no delete available for anonymous users.