Code/Project Documentation

By: Christoffer Vilstrup Jensen
Simon Surland Andersen

February 12, 2020

Contents

1 Setup and installation

To install and run a locally hosted instance of the project, please refer to the document `GFR/doc/install.md`.

1.1 Server management and production installation(s)

For information related to the management and production installation(s) and configuration see: `http://hopper.petnet.rh.dk/wiki/Gfr`.

1.2 Recommended reading

If you're new to working with Django it's recommended that you read:

- Django Getting Started tutorial: `https://docs.djangoproject.com/en/2.2/intro/tutorial01/`
- REST api: `https://en.wikipedia.org/wiki/Representational_state_transfer`
- Samba Share: `https://en.wikipedia.org/wiki/Samba_(software)`

2 Project overview

The project is built using Python 3.6, primarily using the Django web framework. For specific information about functions please refer to their docstrings, the code is written with the intent of following the Google Style Python Docstrings (`https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html`).

The project is made through one Django app, named `main_page`, with a top project directory `GFR/` which contains the following files and directories:

- **Code dirs.:**
  - `GFR/main_page/` - Main Django app directory.
  - `GFR/clairvoyance/` - Django project files e.g.: `settings.py`, `urls.py`, etc.
  - `GFR/doc/` - Files related to documentation of the code and user guide.
- **Files and scripts:**
  - `GFR/manage.py` - Django management script.
  - `GFR/run-tests.sh` - Script for running the automated tests. For more info. run:
    ```
    (venv)> ./run-tests.sh --help
    ```
  - `GFR/requirements.txt` - Python library dependencies.

  - `GFR/db.sqlite3` - SQLite database file.
  - `GFR/uwsgi_params` - Parameters to use when running the uWSGI service.
  - `GFR/clairvoyance_uwsgi.ini` - uWSGI settings file, for setup of uWSGI service.

Most code for the project is found under the directory `GFR/main_page`, which contains:

- `GFR/main_page/fixtures/` - Testing fixtures, i.e. database data to fill out during testing (for more read: `https://docs.djangoproject.com/en/3.0/howto/initial-data/`).
- `GFR/main_page/forms/` - Classes defining Django forms.
- `GFR/main_page/libs/` - Custom utility and wrapper functions:
  - `GFR/main_page/libs/clearance_math/` - Math utility functions for computing clearance, normalized clearance, body surface area, etc.
  - `GFR/main_page/libs/query_wrappers/` - Wrapper functions for simple queries of RIS and PACS.
  - `GFR/main_page/libs/ae_controller.py` - Wrapper library for simplifying the process of making dicom queries using pydicom.
  - `GFR/main_page/libs/dataset_creator.py` - Wrapper library to easily generate empty or partially filled out dicom datasets.
  - `GFR/main_page/libs/dicomlib.py` - Wrapper library for resolving issues related to saving and loading dicom datasets containing private tags.
  - `GFR/main_page/libs/dirmanager.py` - Utility functions for easy and safe creation of directories.
  - `GFR/main_page/libs/enums.py` - Enum definitions used throughout the code-base.
  - `GFR/main_page/libs/formatting.py` - Formatting functions for e.g. converting dates, dicom person names, dots and commas, etc.
  - `GFR/main_page/libs/ris_thread.py` - Background thread executed on startup, used to perform prefetching of registered studies.
  - `GFR/main_page/libs/samba_handler.py` - Wrapper functions for getting data from the Samba Share, for incoming counter files.
  - `GFR/main_page/libs/server_config.py` - General configuration options for the server, e.g. where certain files and directories located, which private tags do we use, etc.
  - `GFR/main_page/libs/status_codes.py` - HTTP and dicom status codes.

- `GFR/main_page/migrations/` - Django migration files for database management.
- `GFR/main_page/static/` - static files, e.g. images, CSS, Javascript, etc.
- `GFR/main_page/templates/` - Jinja templates used to render the site.
- `GFR/main_page/tests/` - Automated testing files for the app.
- `GFR/main_page/views/` - Classes defining Django views, i.e. individual site endpoints.
  - `GFR/main_page/views/api/` - Class-based views for REST api endpoints
- `GFR/main_page/log_util.py` - Utility for handling of logging to multiple files, s.t. the `ris_thread` and main thread of the server can log to separate files without overlap.
- `GFR/main_page/apps.py` - Code to be ran once on startup of the Django server (this is responsible for e.g. starting the background RIS thread for retrieving historic studies).
- `GFR/main_page/backends.py` - Defines the authentication process for simple User login.
- `GFR/main_page/models.py` - Defines database models.
- `GFR/main_page/urls.py` - Defines available endpoints and their corresponding views.

2.1 Changelog

The changelog describing added, changed, removed and/or fixed features of previous versions can be found under: `GFR/doc/CHANGELOG.md`.

2.2 Adding new libraries, frameworks, etc.

If adding any new libraries or frameworks becomes a necessity don't <u>ever</u> use external CDNs, as the application must still be able to run internally, even if connection to the outside internet is down.

3 Structure of stored data

3.1 DICOM files

This project stores dicom objects in nested directories based on the accession number of a study, e.g. accession number is "REGH12345678":
`GFR/active_dicom_objects/REGH12345678/REGH12345678.dcm`

The reason for using nested directories, is so we can store any history gathered from PACS as dicom files next to the currently active study.

This project generates four directories for temporarily storing dicom objects:

- `GFR/active_dicom_objects` - Dicom objects related to currently active studies, displayed on `<SITE_URL/list_studies`.
- `GFR/deleted_studies` - Dicom objects related to studies which have been deleted from `<SITE_URL>/list_studies`. Dicom ojects in this directory are displayed on `<SITE_URL>/deleted_studies`.
- `GFR/control_studies` - Dicom objects related to filled out studies, displayed on `<SITE_URL>/control_list_studies`, that are to be checked through by a second person.
- `GFR/search_dir` - For temporary storage of retrieved search files. This directory should usually be empty, as search files are only stored for a brief period after they have been received.

3.2 Samba Share files

Alongside the Django server we're running a Samba Share, i.e. a shared directory accessible by Windows machines, which stores csv data files from the counter wizards located at each hospital. E.g. on the production server named `gfr`, the Samba Share is located under `/data/`, with two directories:

- `/data/Samples/<HOSPITAL_SHORTNAME>/` - Sample counts for the current day, here `<HOSPITAL_SHORTNAME` is the abbreviated name for the hospital, e.g. "Rigshospitalet" has shortname "RH", "Glostrup" has "GLO", etc.
- `/data/backup/<HOSPITAL_SHORTNAME>/` - Stores backup of old sample files. A sample file is moved to backup, once it's more than 1 day old.

4 DICOM

The full specification of the DICOM standard can be found at:
`https://www.dicomstandard.org/current/`.

Working with dicom objects/dataset throughout this project is done via. two Python libraries:

- `pydicom` (`https://pydicom.github.io/pydicom/stable/getting_started.html`) - for viewing and manipulating dicom files, objects and datasets, etc.
- `pynetdicom` (`https://pydicom.github.io/pynetdicom/stable/`) - for any networking related to DICOM, such as querying dicom databases like RIS and PACS.

4.1 DICOM tags

DICOM tags are data elements stored in a DICOM dataset and are defined by a unique tag, typically represented by a hexadecimal value, a VR (Value Representation), and value to be stored.

For a full list of available VR's see: `http://dicom.nema.org/dicom/2013/output/chtml/part05/sect_6.2.html`

For information regarding specific DICOM tags and description of their usage, please refer to the search tools, where you can search by tag:

- `https://www.dicomlibrary.com/dicom/dicom-tags/`
- `https://dicom.innolitics.com/ciods`

4.2 Private DICOM tags

In the code private DICOM tags are defined in `GFR/main_page/libs/server_config.py`. The private tags are as follows:

Tag (hex)	**Attribute name**	**VR**	**Description**
`0x00231001`	GFR	LO	GFR (Normal, Moderat Nedsat, Svært nedsat)
`0x00231002`	GFRVersion	LO	GFR Version
`0x00231010`	GFRMethod	LO	GFR Method
`0x00231011`	BSAmethod	LO	Body Surface Method
`0x00231012`	clearance	DS	clearance
`0x00231014`	normClear	DS	normalized clearance
`0x00231018`	injTime	DT	Injection time
`0x0023101A`	injWeight	DS	Injection weight
`0x0023101B`	injbefore	DS	Vial weight before injection
`0x0023101C`	injafter	DS	Vial weight after injection
`0x00231020`	ClearTest	SQ	Clearance Tests (Sequence)
`0x00231021`	SampleTime	DT	Sample Time (Sequence item)
`0x00231022`	cpm	DS	Count Per Minuts (Sequence item)
`0x00231024`	stdcnt	DS	Standart Counts Per
`0x00231028`	thiningfactor	DS	Thining Factor
`0x00231032`	ExamStatus	US	Examnation Status
`0x0023103F`	clearancehistory	SQ	Clearance History

5 Formatting specifications

5.1 Commas and Dots (presenting floats)

5.2 Dates and timestamps

6 `ae_controller.py` library

7 API

For dynamic content on the site, such as search, admin panel, and retrieval of old sample data, we use a set of custom defined api endpoints, which uses a REST design to make it easy to work with on the Javascript frontend.

7.1 Available API endpoints

For precise information about which views are used for each endpoint, please refer to the file `GFR/main_page/urls.py`.

- `<SITE_URL>/api/user`: (Requires admin privileges)
  - `GET`: list information about all users.
- `<SITE_URL>/api/user/<user_id>`: (Requires admin privileges)
  - `GET`: list information about specific user based on unique user id.
  - `PATCH`: update information about specific user.
  - `DELETE`: delete specific user.
  - `POST`: create new user.
- `<SITE_URL>/api/`:

7.2 Defining new endpoints

Comment: Something about the custom serializer for Django model instances. Describe how to use the RESTEndpoint class and it's subclasses for defining endpoints with only specific actions.

7.3 Non-REST endpoints

Currently there are only two endpoints which haven't yet been updated to use the REST design of all other endpoints, these endpoints are:

- `<SITE_URL>/ajax/login`:
  - `POST` - attempt to login with the provided request parameters.
- `<SITE_URL>/ajax/update_thining_factor`: (Login required)

– `POST` - update the saved thining factor for the department of the currently logged in user.

8 Database design scheme

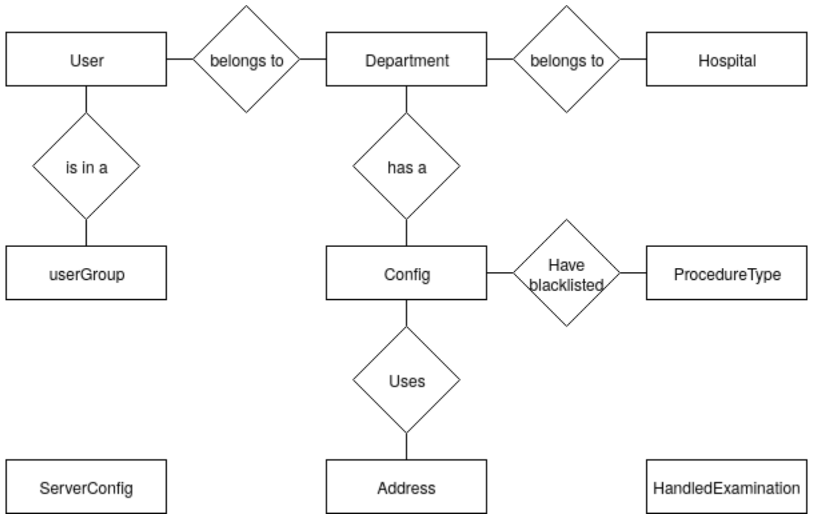

Figure 1: Database relationship diagram

9 Add a panel to admin panel / Database

To enable database manipulation / editting without using tools like DB broswer and their like. There's a admin panel, that allows editting of DB entries. The following is a guide on how to edit the codebase s.t. it can display a new table

1. Add the database schema by creating a model class in *main_page/models.py* file.
2. Update the database by calling the django commands make migration and migrate.
3. Create the class **Edit<model name>Form** corosponding to the edit form and the class **Add<Model name>Form** corosponding to the adding form in the *main_page/forms.py* file
4. Add an Endpoint by adding the class **<model name>EndPoint** in the *main_page/views/api/api.py* file.
5. Update *main_page/url.py* with the endpoint created in step 4, remember to create 2 endpoints for displaying and for accessing an object.
6. In the classes **AdminPanelEditView** and **AdminPanelAddview** found in the *main_page/views/admin_panel.py* file, add an entry with to the dictionary **MODEL_NAME_MAPPINGS** with the model from step one in both classes. Add an Entry to the dictionary in the:
   EDIT_FORM_MAPPINGS / **ADD_FORM_MAPPING** in the respective classes with the form you created in step 3.
7. Update the javascript in the files *main_page/static/main_page/js/admin_panel.js*, *main_page/static/main_page/js/admin_panel_ add.js* and *main_page/static/main_page/js/admin_panel_ edit.js*
8. Add the option to the button in the admin

Total list of files that require to be edited

main_page/

- forms/
  - model_add_forms.py
  - model_edit_forms.py
- models.py
- urls.py
- views/:
  - admin_panel.py
  - api/

        · api.py

static/main_page/js/

- admin_panel.js
- admin_panel_add.js
- admin_panel_edit.js

template/main_page/admin_panel.html