Open Astro Archive

Florian Merges

Grado de Ingeniería de Informática

Ingeniería del Software

Oriol Martí Girona

Santi Caballe Llobet

June, 2019

Universitat Oberta de Catalunya

Open Astro Archive

Florian Merges
Grado de Ingeniería de Informática
Ingeniería del Software
Oriol Martí Girona
Santi Caballe Llobet
June, 2019

Universitat Oberta de Catalunya

Abstract

This thesis covers the design and development of a general purpose astronomical data archive. In astronomy, instruments attached to telescopes, are the main data producers. Astronomical data archives as used to provide access to this vast amount of data, which keeps increasing day by day. These archives are usually developed as part of a sponsored collaborative project between institutions, or as an in-house development. As such, they tend to be tailor made for an institution, a telescope, or its instruments. But, not all of them have an archive that enables astronomers to search, inspect and download data. Therefore, a new software was created, by following an agile software development methodology, in order to fill this gap. The result obtained is a fully functional general purpose astronomical data archive. One that can be used for public data, private data, or both, supporting raw as well as reduced data, and easy to integrate with existing infrastructure.

To my parents

Acknowledgments

I am particularly grateful for the assistance given by my thesis supervisor Oriol Martí Girona by solving all my questions along the way. To Prof. Dr. Santi Caballé Llobet for his clarification that tilted the balance towards a thesis in software engineering. And to all the people of the UOC community that I had the pleasure to interact with.

I would also like to offer my thanks to my work colleagues, present and past, specially to Saskia Prins for sharing her knowledge in astronomy, and to Prof. Dr. Hans Van Winckel for giving me the opportunity to work for the Mercator Telescope project.

Last but not the least, I would like to thank my family, for always believing in me, for their love, support, and understanding.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Background

Instruments, attached to telescopes, are the main scientific data producers in the astronomical field. Most of these instruments use Charge-Coupled Device (CCD)[1] detectors to capture the light-energy from the objects in the sky, and after an analog to digital conversion, store this pixel data to a file along with some metadata. This metadata may consist of: sky coordinates, date and time, instrument settings, and meteorological information, among others. The sum of both, pixel data and metadata, is known as raw data, and in consequence a file with raw data is called a raw file[2].

This raw data by itself is not of much use to an astronomer, it needs to be reduced first. Data reduction is very much instrument specific. Custom made software pipelines create reduced data by processing raw and calibration data together. At least three steps are necessary to reduce raw data: bad pixel removal, bias and dark current subtraction, and flat fielding.[2]

The standard file format to store this data is the Flexible Image Transport System (FITS)[3] format. Regardless of being a dated format, and attempts to use others like HDF[3], the majority still rely on FITS. The metadata is stored in what is known as Header Data Units (HDU) consisting of keyword/value pairs. The FITS format requires at least one HDU, from now on simply called header, with some mandatory keywords. The remaining keywords, and the nature of their values, are very much unregulated and up to its creator.

Due to this liberty, headers from two instruments, even from the same telescope, might be significantly different. Moreover, the format of the header for a given instrument might change as well with time, for instance, new keywords are added, a data unit of a keyword is modified, etc. Thus, without a special keyword in the header that identifies the version of its format, the heuristics as to how to extract and normalize them may become somewhat tricky.

The vast amount of data produced by sky surveys and observations need to be cataloged and made accessible to astronomers, and here is where astronomical data archives come into play.

[1] "A semiconductor device that is used especially as an optical sensor and that stores charge and transfers it sequentially to an amplifier and detector."[1]

[2] A simple analogy could be a raw photo from a consumer digital camera.

[3] "HDF5 is a data model, library, and file format for storing and managing data. It supports an unlimited variety of datatypes, and is designed for flexible and efficient I/O and for high volume and complex data."[4]

Although there exists historical data archives dating back to earlier centuries made up of drawings and notes, and more recently of photographic plates, our concern is about the data archives of the digital era.

Astronomical data archives are usually developed as part of a sponsored collaborative project between institutions, or as an in-house development. One way or the other, these archives tend to be tailor made for an institution, a telescope, or its instruments—but not really with a general purpose in mind.

Althought there is an alliance, the International Virtual Observatory Alliance (IVOA)[5] founded in 2002, with the mission to facilitate the interconnection and interoperability between data archives and tools—through the definitions of standards. (Archives supporting these standards can be queried by third parties in a transparent way, by means of a common language, supported by a growing list of tools.) However, they do not provide a data archive solution, one, ready to be used by a telescope or astrophysical observatory.

1.2 Objectives

The aim of this project is to design and develop a general purpose astronomical data archive. In short, a system that enables users to search, access, and download astronomical data. In addition, enabling them to manage the access to the data, hence rendering it suitable for collaborative environments with multiple research groups.

This archive can be a solution for small to medium sized telescopes, professional astronomers, and amateurs willing to offer access to their data but, due to resource limitations are unable to develop a custom solution.

1.3 Procedure Statement

The output of an initial research[4] suggest that there are no off-the-shelf astronomical archive solutions available. As mentioned before, see Section 1.1, archives are tailor made for a given institution or telescope. This means there may be a demand for a product like this, accordingly, a new product is developed.

Furthermore, the research brought to light several similarities—taken into account during design—among the reviewed astronomical archives:

- most of them only provide access to public data.
- have a search facility.
- display a subset of the metadata stored in a raw header[5].
- allow inspection of the raw header.
- link the files with their respective observing program.
- provide a means to retrieve data, some directly, others per request.

Next, the development method or approach in order to create the product is selected. In software development there are basically two big methodology families. On one side

[4] An initial research is always important in order to prevent reinventing the wheel.

[5] The raw header as it is enclosed in the FITS file.

we have those that follow the classical waterfall method, and on the other we have the so called agile methods.

Agile methods follow an iterative process in which the classical steps of a waterfall method (business requirements → technical design → coding & testing → launch) are repeated in iterations for the duration of the project; and by continuous inspection and adaptation, avoid loosing track of the project's objective. Thus, contrary to those following the classical method, agile methods embrace change. The initial idea is to have a product at the end of the project, therefore the choice is to use an agile method, otherwise, too much precious time is invested in the analysis and design of it.

The Scrum framework is used for the development of this project. "Scrum is a framework for organizing and managing work. The Scrum framework is based on a set of values, principles, and practices that provide the foundation to which your organization will add its unique implementation of relevant engineering practices and your specific approaches for realizing the Scrum practices. The result will be a version of Scrum that is uniquely yours."[6, page 13]

As this project is the bachelor thesis of the author, and because it is not a team effort, the framework was adapted based on time constraints and personal preferences in order to meet the allotted time frame. The adaptations are:

- keep low ceremony.
- fusing the three roles: Product Owner, ScrumMaster, and the Development Team.
- no meetings, nor daily scrum sessions.
- simplified sprint[6] actions: planning, inspections, scrum, and retrospectives.
- simplified sprint backlog.

In addition, due to the time constraints mentioned earlier, an entirely Test Driven Development (TDD)[7]—even less a Behavior Driven Development (BDD)[8]—seem unfeasible. In consequence, only unit tests for essential components are implemented. Nonetheless, the author strives to follow most of agile's best principles and practices, making it a constant to remind himself of: "Do the Simplest Thing That Could Possibly Work", "You Aren't Gonna Need It", and "Don't repeat yourself (DRY)".

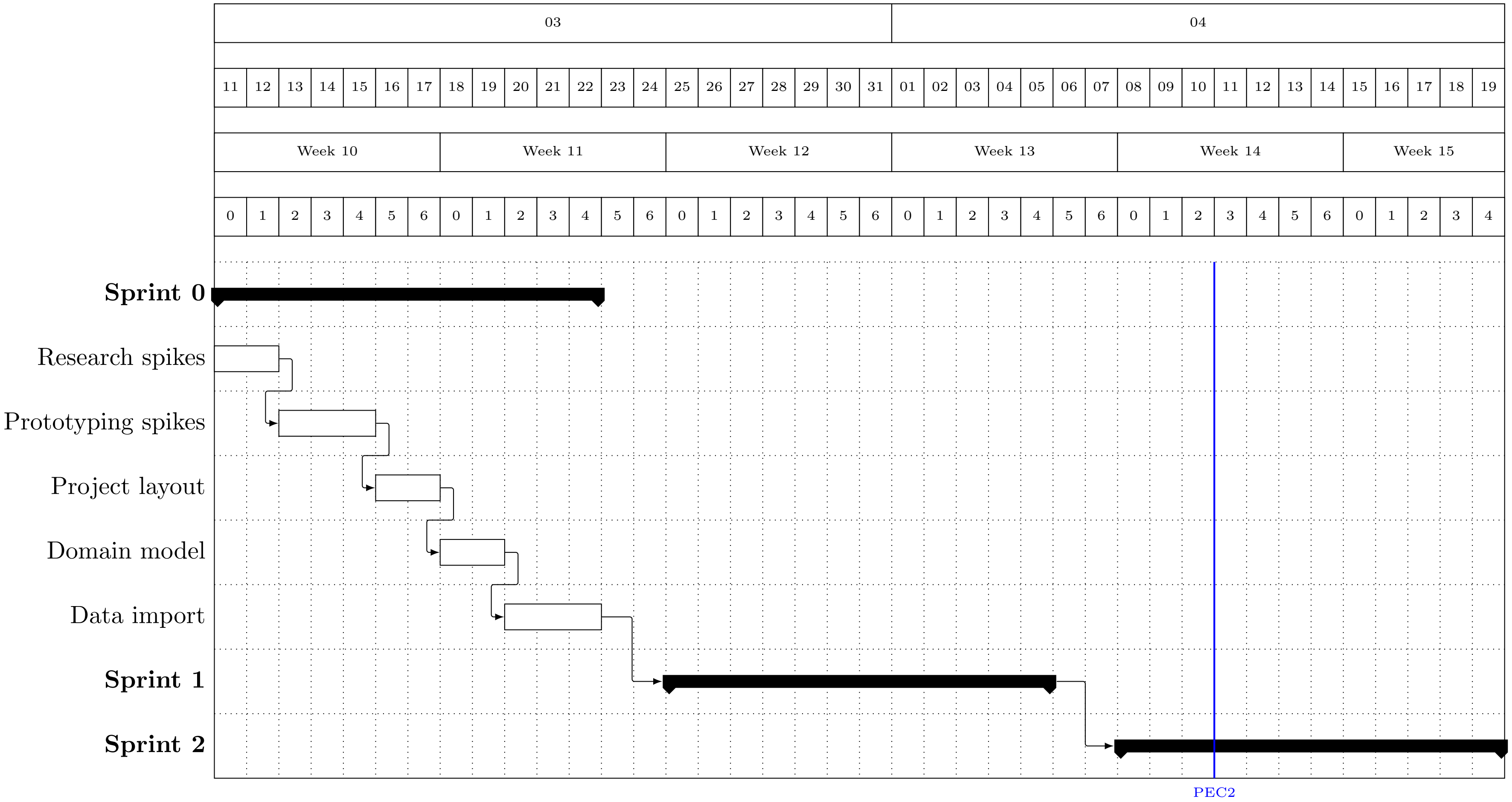
1.4 Time Management

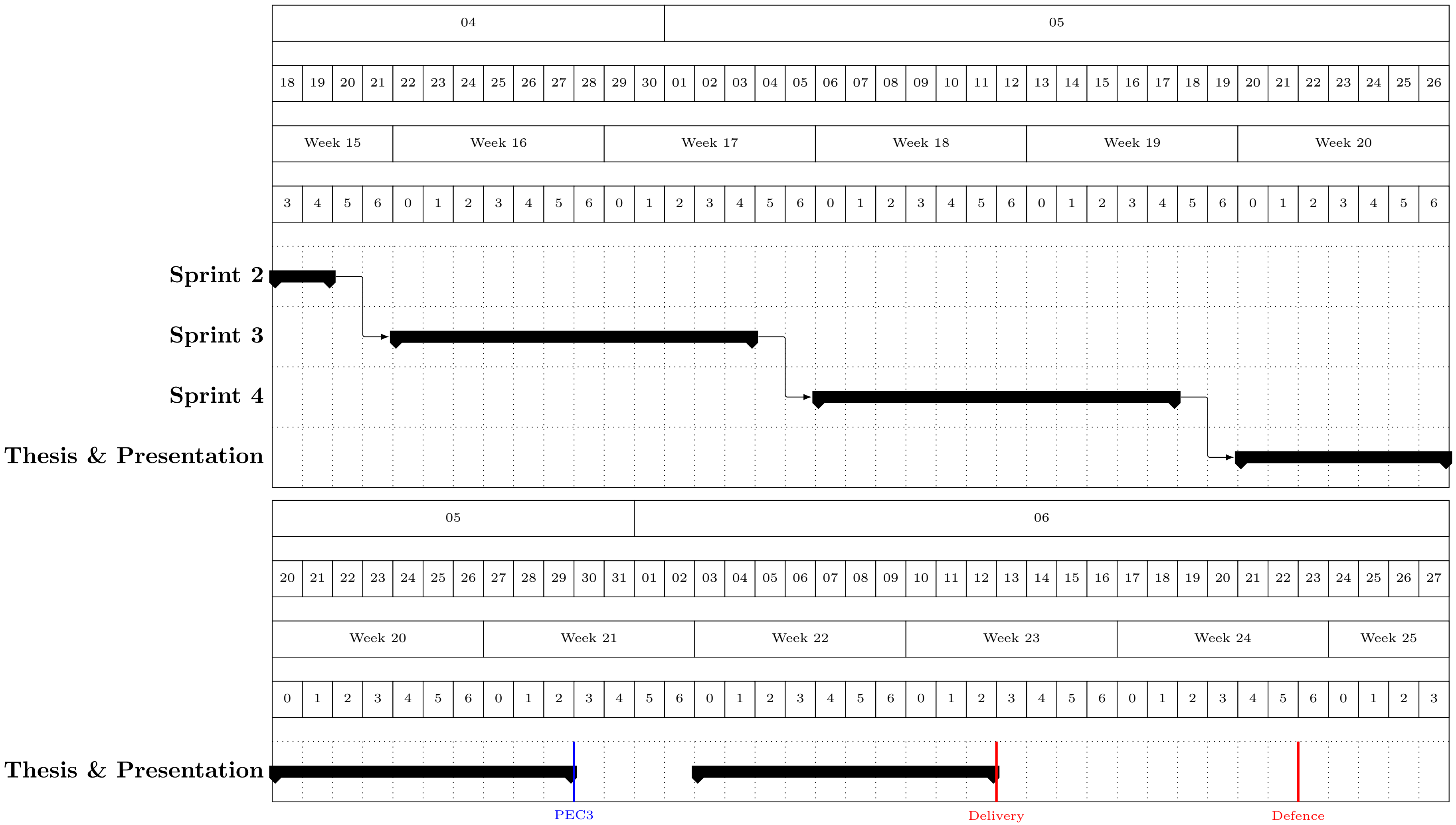
The following Gantt chart covers the period between the start of the first sprint until the thesis's defence day. It displays the month, the days, the weeks, the day of the week (zero for Monday), and also highlights some important dates. After each sprint there are two slack days, used to recover and gain perspective. Notice also that only the first sprint is shown ungrouped—the first sprint is always somehow special.

[6] "Scrum organizes work in iterations or cycles of up to a calendar month called sprints. [...] They are timeboxed, have a short and consistent duration, have a goal that shouldn't be altered once started, and must reach the end state specified by the team's definition of done."[6]

[7] TDD is an approach in which tests are written first, followed by just enough code to pass those tests.

[8] BDD is an approach, emerged from TDD, in which technical and non-technical people can work together using a domain specific language in order to express the behavior of a system, and consequently test it.


03
04
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19
Week 10
Week 11
Week 12
Week 13
Week 14
Week 15
0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4
Sprint 0
Research spikes
Prototyping spikes
Project layout
Domain model
Data import
Sprint 1
Sprint 2
PEC2


04
05
18 19 20 21 22 23 24 25 26 27 28 29 30 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
Week 15
Week 16
Week 17
Week 18
Week 19
Week 20
Sprint 2
Sprint 3
Sprint 4
Thesis & Presentation
05
06
20 21 22 23 24 25 26 27 28 29 30 31 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
Week 20
Week 21
Week 22
Week 23
Week 24
Week 25
Thesis & Presentation
PEC3
Delivery
Defence

1.5 Results

The result obtained is a fully functional general purpose [9] astronomical data archive. An archive suitable for all environments, public or private or both. It supports raw as well as reduced data. Data is kept safe as it only requires read access to it. Further, standard command line tools are provided for an easy integration with existing infrastructure.

In addition, a simple but powerful web application is provided, enabling users to:

- search data by different criteria, including cone search.
- export search results into different output formats.
- access the data's most relevant information.
- inspect raw data headers.
- access to observing night reports, and thus understand the observing conditions of the data.
- directly plot and inspect reduced data.
- link to the most popular astronomical objects catalog and its finding charts.
- select and download data.
- access the download history.
- see the details of a program and its data.
- change the details of a program
- upload attachments to a program.
- modify the user permissions for a program.
- track the progress of a program, for instance, see when data was taken and how many.
- subscribe to different notifications.
- receive e-mail notifications.
- sign up in the system.

[9] General purpose because it is not tied to any given telescope or instruments.

1.6 Chapter Summaries

This document covers the design and development of a general purpose astronomical data archive, using an agile methodology. As such, it tries to engage the reader into the iterative process followed from inception to conclusion.

Booch et al. in [7, page 494] write: "Sadly, there is no commonly agreed-upon way to quantitatively represent an arbitrary architecture." But, this is not necessarily something bad. After all, there is a component of art in engineering, and akin to real life a lot of times there is more than one path leading to the same destination.

The remaining chapters are:

Chapter 2 focuses on the product requirements. Gives an overview of the main stakeholders of the system, introduces the product backlog—which is a prioritized list of requirements or desired functionality—, the product constraints, and the template used for the user stories—a user story is a format to express product backlog items.

Chapter 3 is about the architecture and technology of the system. Explains the project layout (or skeleton), introduces the core domain model entities, and the intricacies of the data import.

Chapter 4 details the general user interface design, digs into the security and permission strategy implemented in the system, and how user management is done.

Chapter 5 delves into data search, and how raw and reduced data are displayed. It also covers briefly some external services.

Chapter 6 examines the features related to observing programs and how program's data access is handled. In addition, this chapter unfolds the details of data download.

Chapter 7 is all about user notifications, the different options available, and how users can manage their notification subscriptions.

Chapter 8 exposes the conclusions and future visions of the project.

Appendix A contains source code snippets referenced in the document.

Appendix B goes step by step through the installation process for a production server and a development environment.

Chapters 3–7 correspond to the sprints layed out in Chapter 2. They all follow a similar structure: first the user stories of the sprint are introduced; followed by a more detailed exposition of them and how they were implemented in the system; and finally, a short retrospective, a review, or both.

Chapter 2

Requirements

2.1 Stakeholders

One of the first steps in any project consists in identifying the main stakeholders:

Principal Investigator (PI), and co-investigator (Co-I): Commonly, astronomical research is grouped into programs or proposals; each program has at least a PI[1], and most of the times one or more collaborators—also known as Co-Is.

Administrator: The actor in charge of the application, the data, and its storage.

User: Any actor, besides an administrator who wants access to the archive to search, inspect, and download data.

2.2 Product Backlog

The project development uses the Scrum framework, its tools and strategies. In Scrum, the central artifact around everything evolves is the product backlog. "The product backlog is a prioritized list of desired product functionality. It provides a centralized and shared understanding of what to build and the order in which to build it. It is a highly visible artifact at the heart of the Scrum framework that is accessible to all project participants."[6, pages 99–100]

"As with XP[2], in Scrum it is not important for the product owner to identify all of the requirements up front. However, there is often a benefit to jotting down as many of them as possible at the outset. Scrum has no prescribed, or even recommended, approach to initially stocking the product backlog."[9]

Table 2.2 illustrates the initial product backlog with the user stories identified. The table format is inspired by Cohn's Spreadsheet-Based Product Backlog[10]. The unique ID is omitted because it doesn't add any relevant information for the case statement—and it is generated by the Source Configuration Management (SCM) system anyways.

[1] A PI is "the scientist in charge of an experiment or research project."[8]

[2] Extreme Programming (XP), another agile software development method.

2.3 Product Constraints

Product constraints are the limitations and restrictions that apply to a product. Non-functional requirements are requirements that address a variety of system needs, and thus can be considered to be product constraints as well.

Therefore, the non-functional requirements of the system are grouped into product constraints expressed by user stories. It might seem unusual at first, but the strong point of using stories for non-functional requirements is that they allow to keep track of who raised the requirement[3].

As a/an	**I want to...**
Product Owner	support multiple instruments and data products
Product Owner	be easy to use
Product Owner	not impose special software requirements
Product Owner	be compatible with mobile devices and tablets
Product Owner	be extensible by third parties
Product Owner	make it easy to internationalize the software if needed

Table 2.1: Requirements: Product Constraints

2.4 User Story Template

"User stories are a convenient format for expressing the desired business value for many types of product backlog items, especially features. User stories are crafted in a way that makes them understandable to both business people and technical people. They are structurally simple and provide a great placeholder for a conversation. Additionally, they can be written at various levels of granularity and are easy to progressively refine."[6, page 83]

Figure 2.1 shows the user stories template used in this document. The template adheres to the recommendations made by Cohn[9], Rubin[6], and others. Each story has a title, a flag indicating if the story is a constraint, a short description, and eventually a confirmation—also known as conditions of satisfaction. Furthermore the user stories are not numbered, as: "[...] numbering story cards adds pointless overhead to the process and leads us into abstract discussions about features that need to be tangible."[9]

Name	C
Description	
Confirmation	

Figure 2.1: Requirements: User story template

[3]For a longer discussion about non-functional requirements and user stories see [11]

The user stories in this document do not include story points. Story points are useful in a team environment in order to prioritize the work to be done. Due to the lack of a team, gut feeling and delivering something of value with each sprint—and not just a potentially shippable product increment—are used instead. Related to this, Kniberg[12] writes:

> “Once we have a preliminary list of stories to be included in the sprint I do a “gut feeling” check. I ask the team to ignore the numbers for a moment and just think about if this feels like a realistic chunk to bite off for a sprint. If it feels like too much, we remove a story or two. And vice versa. At the end of the day, the goal is simply to decide which stories to include in the sprint. Focus factor, resource availability, and estimated velocity are just a means to achieve that end.”

Table 2.2: Requirements: Product Backlog

As a/an	I want to...	so that...	Priority	Sprint
Admin	import new data into the system	users can access it	High	0
Admin	register new users	they can use the system	High	1
Admin	enable or disable users	I can deactivate unused accounts	High	1
User	to register myself into the system	I don't need to contact an administrator	Medium	1
System	provide security measures for data access	I can be used for private, and public data	High	1
User	search existing data	I can see what is available	High	2
User	see the detail of a raw file		High	2
User	have access to the end of night report for a file	I can see the conditions of the observation	Low	2
User	have access to the finding chart for any given raw file		Low	2
User	have access to the header of a raw file		High	2
User	analyze the reduced spectrum		Low	2
PI/Co-I	define details of my programs		Medium	3
PI/Co-I	manage user access to my programs	I choose who can see my data	Medium	3
PI/Co-I	upload the proposal to my programs	users can download it	Low	3

Continued on next page

Table 2.2: (continued)

As a/an	**I want to...**	**so that...**	**Priority**	**Sprint**
PI/Co-I	see the progress of my programs	I can see the health of it	Low	3
Admin	administrate user access	I can give rights to certain users	Medium	3
User	select data to download		High	3
User	download data as a tarball	I don't have to download individual files	High	3
User	receive notification when new data is available	I don't have to check every so often	Low	4
User	receive a notification when a tarball is ready		Low	4
User	I want to choose the user interface language		Low	4

Chapter 3

Sprint Zero

3.1 User Stories

Data support	C
The system should support multiple instruments, and data products	
Design takes into account different instruments Design takes into account different data products	

No special software requirements	C
The system should not impose special software requirements to the final user	
The user doesn't need to install any specialized software to use the application	

Responsive design	C
The system should be compatible with mobile and tablet devices	
With a desktop computer With a tablet With a smartphone	

Extensibility	C
The system should be extensible and customizable by third parties	
The system can be easily adapted to a given telescope, its instruments, and its data products	

Architecture and technology	
Define the architecture and technology to be used for this project	
Architecture is scalable Follows current best practices	

Project layout	
Create the general project layout	
Follows recommendations of the technology used	

Initial domain model	
Should only focus on the data import	
Only data import domain objects are considered	

Data import	
As an Administrator, I want to be able to import new data into the system	
The system supports data produced by different instruments The system supports raw and reduced data The system supports different header versions	

3.2 Architecture

After an initial analysis of the requirements laid out in Section 2.2, it seems logical, to opt for a web application.

> "One of the biggest changes to enterprise applications in the last few years has been the rise of Web-browser-based user interfaces. They bring with them a lot of advantages: no client software to install, a common UI approach, and easy universal access. Also, many environments make it easy to build a Web app." [13, Ch. 4]

Accordingly, the system is composed of: a web application for the user interaction, some tools for the astronomical data import, a database, and a task queue for asynchronous processes (like tarball[1] generation, e-mails, and notifications). Figure 3.1 shows a simplified diagram of the system.

The system is developed following an object oriented approach, which allows for a higher degree of abstraction, modularity, and easier maintenance. Additionally, the system follows a layered architecture [15, page 1]; to be precise a 3-tier architecture (not counting the service layer) as it combines the business layer with the persistence layer.

[1] "Tarball is a frequently used jargon term for an archive that has been created with the tar command." [14]

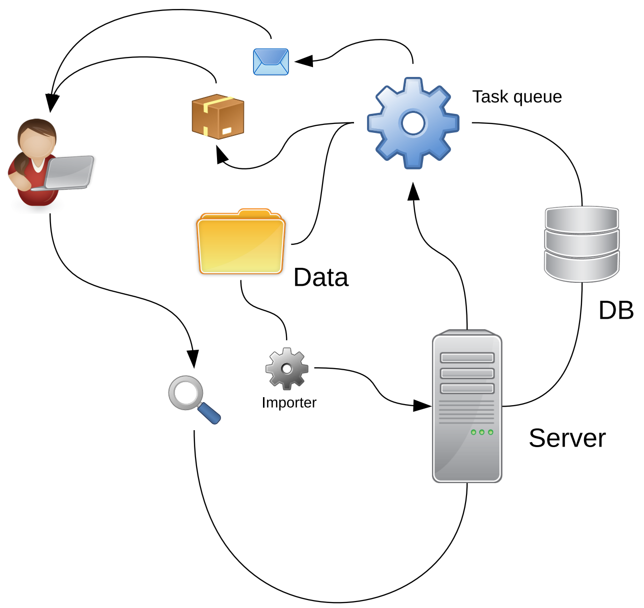


Figure 3.1: Sprint Zero: System diagram

A framework is used for the web application, it follows the Model-View-Controller (MVC) or Model-Template-View (MTV) design pattern[2]. A short digression: although the pattern runs under the same name as the classical MVC pattern [13, Ch. 14], there are some small but important differences. They are equal in sense that both call for separation of concerns, which we could say is the most important aspect of them. (Separation of concerns is the ability to change one of the component of the pattern without affecting another.) But, while the classical MVC pattern—specially suited for GUIs, and dating back the times of Smalltalk—is located at the presentation layer in a tiered architecture and only communicating with the layer(s) below, the Controller part of the *new* MVC pattern is located outside and communicates with both layers: presentation layer and business layer. The web frameworks available in Python's ecosystem follow this *new* MVC pattern[3]. Figure 3.2 shows a diagram of how the layered architecture looks like.

The web application uses the traditional stateless page-redraw model despite recent trends towards single page applications (SPA). Single page applications perform well whenever a desktop application experience is to be simulated in a browser, or for highly interactive web applications, but this is not necessarily the case for an online data archive. Besides, little is gained by adding more complexity to the presentation layer.

In general terms, the data import consists of: reading in raw or reduced data files, extraction and normalization of the information retrieved from the metadata located in the headers, and its storage into a database for query and retrieval. As the system is not responsible for the storage of the imported data files, and only requires access to them, the data can be located anywhere, on the same server, a shared resource, or even the cloud[4].

The extracted information from the data files is persisted into a relational database. This type of database uses the relational model—consisting of tuples and relations—to implement our Entity-Relation (E-R) domain model[5]. A relational database in combina-

[2] A "[...] pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."[16]

[3] See [17] and [18] for a longer discussion.

[4] "This metaphor represents the intangible, yet universal nature of the Internet."[19]

[5] See [20, Ch. 2–3]

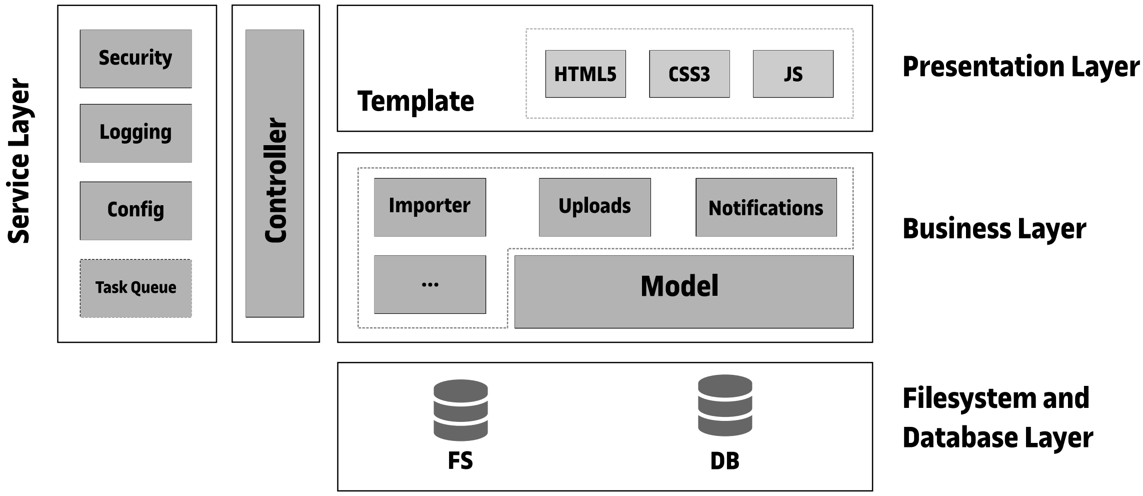


Figure 3.2: Sprint Zero: Layered architecture

tion with a Relational Database Management System (RDBMS) offers many benefits: the Structured Query Language (SQL)[6], the ACID[7] properties, scalability, and compatibility with many programming languages and tools.

Furthermore, a job queue is used to carry out asynchronous tasks, those not fitting the standard stateless request-response pattern of a web application, or those exceeding the maximum response timeout of a HTTP[8] request, like for instance: generating a tarball or sending an e-mail notification.

3.3 Technology

The main programming language of choice is Python. It is object oriented, dynamic, multiparadigm, multiplatform, with an easy learning curve, and with wide acceptance in academia, specially in astronomy. "Python is, as of this writing, poised to become the dominant language used in software systems for astronomy, for pre-post observing, and for during-observing software at the middle and higher layers, and for lower level prototypes."[22]. Thanks to the latter, there exists a significant number of scientific, numeric, and graphical libraries available.

The MVC framework *Flask*, along with several extensions, is used for the development of the web application. The framework extensions provide common features like database management, security, forms, mail, and internationalization among others. Although being a microframework, it is robust and scalable enough to manage an application of this size.

The domain model makes use of the Object Relational Mapper (ORM)[13, Ch. 10] provided by *SQLAlchemy*[23]. *SQAlchemy* is heavily inspired by the patterns mentioned in [13], among them, one is crucial to understand how an ORM functions: the Unit Of Work (UOW). By using an ORM we avoid most of the risks associated with SQL injections[9]

[6] See [20, Ch. 4]
[7] Stands for Atomicity, Consistency, Isolation, Durability. See [13, Ch. 5]
[8] "Stands for *Hypertext Transfer Protocol*. HTTP is the protocol used to transfer data over the web. It is part of the Internet protocol suite and defines commands and services used for transmitting webpage data."[21]
[9] "A SQL injection attack consists of insertion or *injection* of a SQL query via the input data from the client to the application."[24]

as the library handles the escaping for us—using placeholders otherwise when raw SQL queries are demanded. Moreover, *Alembic*, a lightweight database migration tool, is used to version the changes to the domain model, thus translating in easier software upgrades. *SQLAlchemy* and *Alembic* are well integrated with *Flask*.

In addition to the aforementioned, many Python libraries are used, some standing out by themselves like: *astropy* for processing data files and coordinates calculations, *numpy* for manipulating data matrices, *astroquery* and *requests* to communicate with external services, *py.test* for units tests, and *Sphinx* for documentation.

The relational database management system *PostgreSQL* with *Q3C*[25] extension is used. *Q3C* provides spatial indexing on a sphere, and several functions for different types of coordinate search.

For the asynchronous tasks the system relies on *Celery* and *Redis*. The former is a distributed task queue, while the latter is an in-memory data structure storage which can be used for caching purposes as well. *Redis* is used as the message broker for *Celery*.

The user interface client side code is written in HTML5, CSS3, and JavaScript. *JQuery*, a widely used cross-platform JavaScript library, is used for the client side scripting. The responsive design and the homogeneous look'and'feel are achieved through *Twitter Bootstrap*.

The web application is deployed to a WSGI[10] application server called *uWSGI*, which in turn communicates with the chosen web server: *NGINX*. To finish it all of, *NGINX*, and all the other servers and services are deployed on a *Linux* platform.

3.4 Project Layout

The project layout follows Flask's recommendations for large applications[27].

The root folder contains the `open_astro_archive` package. All the modules required by the application are part of it: constants, domain model, application factory, importer, configuration, and blueprints[11]. Further, the application is split into the following blueprints: *main*, *program*, *admin*, and *api*. The `static` folder contains the images, stylesheet, and JavaScript code; the `template` folder, all the templates used by the presentation layer. Blueprints may have their private `static` and `templates` folders.

Returning to the root folder of the application, there is a migrations folder containing files for the migration tool, another for unit tests, and two special files: `Pipfile` with the list of software requirements of the application, and `wsgi.py`, which provides the application instance and extends *flask* script with the command line tools for data import, database creation, etc.

The project layout described can be seen in Figure 3.3.

[10] "The Web Server Gateway Interface (or *WSGI* for short) is a standard interface between web servers and Python web application frameworks."[26]

[11] "Flask uses a concept of blueprints for making application components and supporting common patterns within an application or across applications. Blueprints can greatly simplify how large applications work and provide a central means for Flask extensions to register operations on applications."[28]

```
/
├── migrations
├── open_astro_archive
│   ├── admin/
│   ├── api/
│   ├── constants.py
│   ├── database.py
│   ├── extensions.py
│   ├── factory.py
│   ├── importer
│   │   ├── core.py
│   │   ├── filters.py
│   │   ├── __init__.py
│   │   └── mercator.py
│   ├── __init__.py
│   ├── main/
│   ├── models.py
│   ├── notifications.py
│   ├── permissions.py
│   ├── program/
│   ├── settings.py
│   ├── static/
│   ├── tasks.py
│   ├── templates/
│   └── uploads.py
├── Pipfile
├── tests
└── wsgi.py
```

Figure 3.3: Sprint Zero: Project layout

3.5 Domain Model

The domain model is quite simple, as this sprint focuses only on the initial project layout and data import.

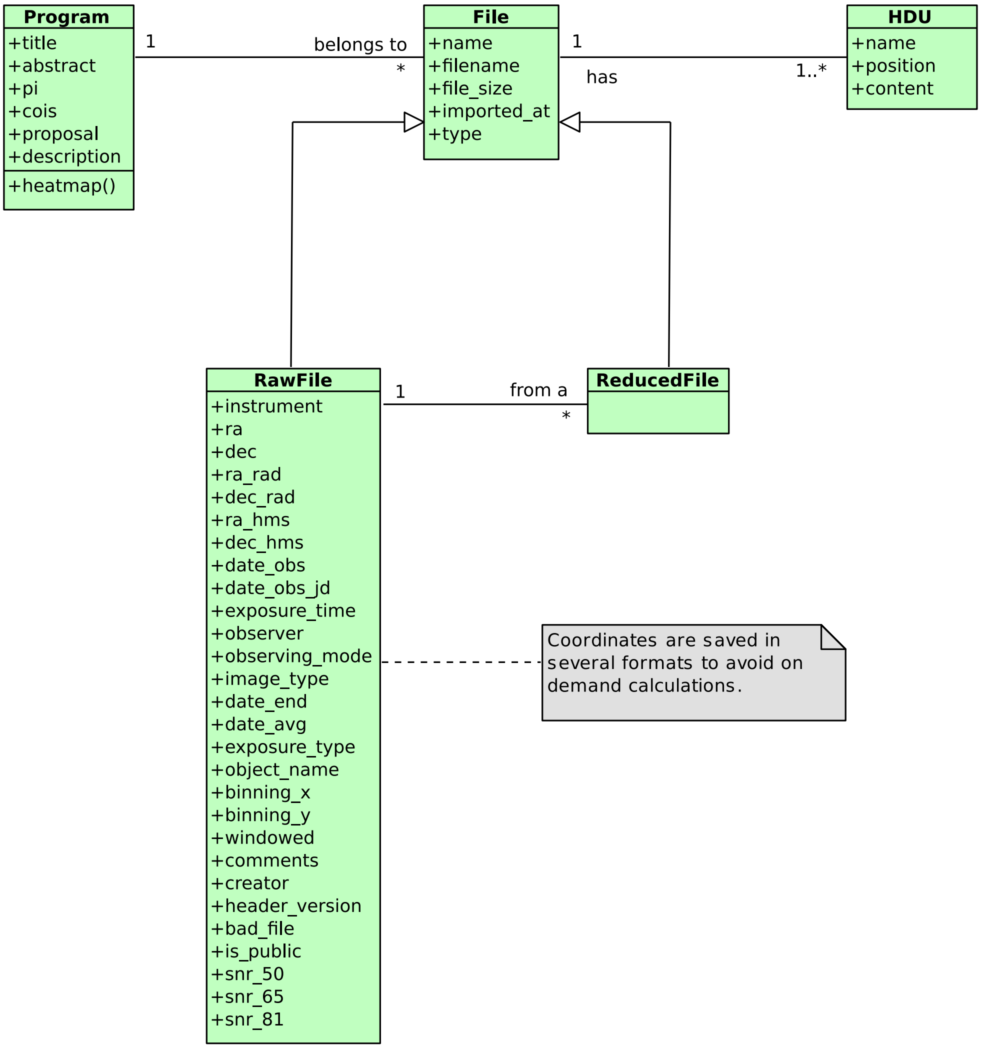


Figure 3.4: Sprint Zero: Domain model class diagram

The `File` class is the parent class of `RawFile` and `ReducedFile`, they represent raw and reduced data respectively. The reasoning behind the `File` class is not only due to the fact that both data types are files, but because by doing so they can be handled the same way from the model's point of view—something handy when dealing with files and downloads.

The `Program` class represents an observing program. An observing program, from now on just program, can be defined as a study which is composed of astronomical objects or targets to be observed under certain conditions[12]. Once an observing program is approved by a time allocation committee, actual observing time on a telescope is granted and scheduled. As such, most of the time, data product of an observation, be it raw or reduced, is associated with a program[13].

[12]The requested observing conditions is part of what is called Phase II information. It is not relevant for archiving, but for scheduling observations.

[13]Particular cases can be dealt by each specific telescope's importer module.

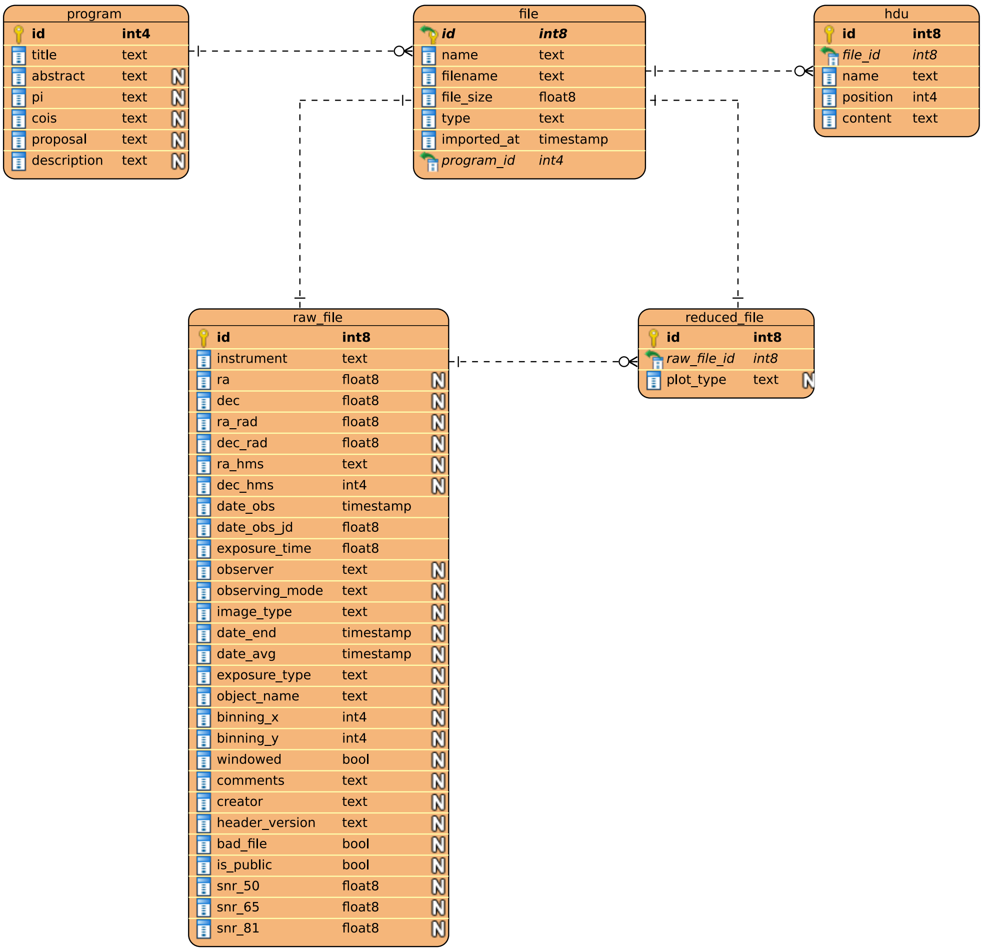


Figure 3.5: Sprint Zero: Data model relational diagram

The `HDU` class, an acronym that stands for Header Data Units, models the FITS file headers. The model is designed in such a way that any `File` object can have many associated `HDU` objects. In practice however, only raw file headers are saved[14]; reduced file headers are mostly a copy of each respective raw header. The string representation of headers are stored entirely along with its position and name. Figure 3.5 shows the initial relational diagram displaying the entities related to the data import with its corresponding data types.

3.6 Data Import

Data import is the action of bringing into the system new data in order to make it available to users. Some of the requirements of the data import are: to accept different sources or instruments, support raw and reduced data, and be compatible with different header versions. The header information between instruments or even institutions, specially the latter, can be significantly different. On top of that, the data acquisition system of any given instrument might suffer modifications during its lifetime, thus the information contained in the data headers possibly changes as well, for example: addition of new fields, or modification of a data unit.

As depicted in Figure 3.6, the data import process consists of a loop that processes

[14]In case this requirement may change, the data model won't be affected.

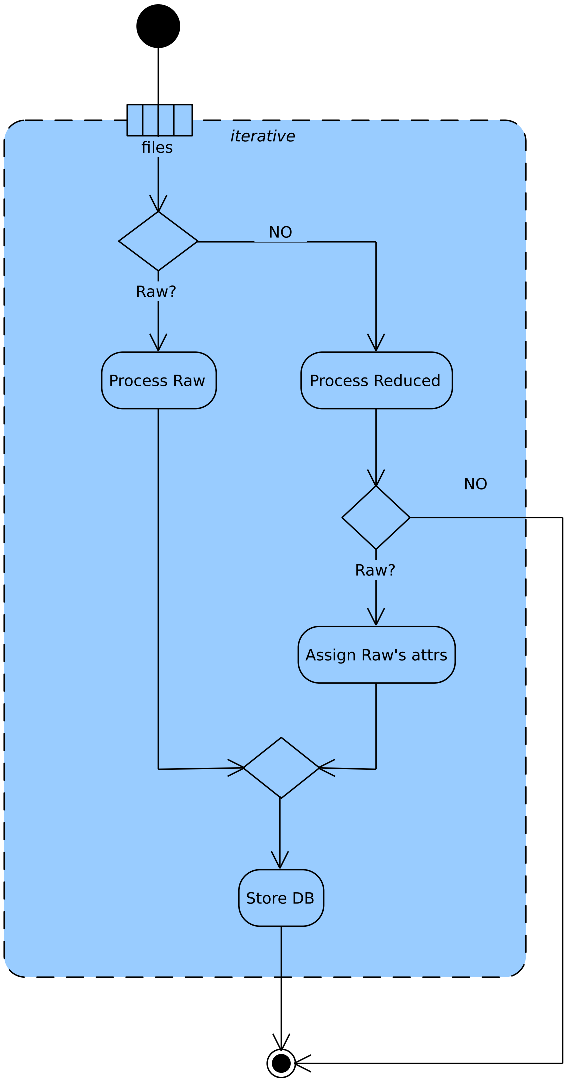


Figure 3.6: Sprint Zero: Data import activity diagram

each input file found in a source folder. Based on the type of data, raw or reduced, it will select the appropriate importer. Once the file has been processed and the objects created, they are stored in the database.

From the FITS format specification it is obvious that the data import component has to be flexible enough to accommodate the most diverse scenarios: incomplete headers, missing keyword entry, wrong data unit, different header versions, no header version, etc. One solution is to have a dictionary to lookup based on the data source or telescope, the instrument, and the header version, how a certain attribute (that will be stored in the database) needs to be processed. Further, a general template or default configuration can be defined per telescope or instrument or both. By means of filters, special cases can be dealt with. Figures 3.8, 3.9, 3.10, 3.11 show the problematic in action, four different keyword naming convention for the same information, and on top, the coordinates of Figure 3.11 use B1950 instead of J2000.

A filter can be used to normalize data, ensure a certain measurement unit, or to generate an attribute value for a missing header keyword. All filters share a common interface, and each filter has access to the full data header, in case an attribute has to be derived from other keywords.

Generally, software originates from a concrete problem, and through an abstraction process, evolves into a generic solution. This case, by no means is any different, and in consequence, the first version of the import tool is written for a given data set. The data set contains raw and reduced data for different instruments including imaging and spectroscopy; with single and multiple headers. It has been anonymized, object names and coordinates changed, and several other steps, all to avoid disclosing any private data. This data set is used for testing purposes as well.

As already mentioned in Section 3.4, the import tool is made available through the `flask` script. While a command line tool might seem a bit simplistic at first, it is a good choice over a daemon[15] as it offers a lot of integration flexibility. (This approach is in line with Raymond's[30] Rule of Composition.) Section A.1 shows the source code of the import tool, and Figure 3.7 shows the tool in action.

```
(open-astro-archive) fmerges@vegas:~/programming/open-astro-archive$ flask import-files data/hermes/raw/
Importing mercator raw data
Processing directory data/hermes/raw/:
  [####################################]  100%
(open-astro-archive) fmerges@vegas:~/programming/open-astro-archive$ flask import-files --reduced data/hermes/reduced/
Importing mercator reduced data
Processing directory data/hermes/reduced/:
  [####################################]  100%
```

Figure 3.7: Sprint Zero: Data import tool screenshot

3.7 Retrospective

The general architecture, technology, project layout, and initial domain model are defined.

The data import works and passes all the unit tests produced for the test-dataset. Yet, as mentioned in Section 3.6, the data import is closely related to the test-dataset available; this should change an become a generic solution, in result, a new story was added to the backlog.

[15] "[...] a computer daemon is a constantly running program that triggers actions when it receives certain input."[29]

```
DATE    = '31/10/97'           / Date file was written (dd/mm/yy) 19yy
ORIGIN  = 'CEA/SSL UC Berkeley' / EUVE Science Archive
CREATOR = 'STWFITS '           / Fitsio version 11-May-1995
TELESCOP= 'EUVE    '           / Extreme Ultraviolet Explorer
INSTTYPE= 'DS/S    '           / Instrument type (DS/S, SCANNER)
OBJECT  = 'NGC 4151'           / Name of observed object
RA_OBJ  =     182.635454000001 / R.A. of the object (degrees)
DEC_OBJ =     39.4057280000001 / Declination of the object (degrees)
RA_PNT  =     182.988000000001 / R.A. of the pointing direction (degrees)
DEC_PNT =              39.5477 / Declination of the pointing direction (degrees)
RA_PROC =     182.637910000001 / R.A. used to process data (degrees)
DEC_PROC=             39.41343 / Declination used to process data (degrees)
```

Figure 3.8: Sprint Zero: FITS Header example (one)

```
DATE-OBS= ' 2/07/96          ' / UT date of start of observation (dd/mm/yy)
TIME-OBS= '14:03:54          ' / UT time of start of observation (hh:mm:ss)
EXPSTART=       50266.58605108 / exposure start time (Modified Julian Date)
EXPEND  =       50266.58949003 / exposure end time (Modified Julian Date)
EXPTIME =       297.1250000000 / exposure duration (seconds)--calculated
EXPFLAG = 'NORMAL            ' / Exposure interruption indicator

        / TARGET & PROPOSAL ID
TARGNAME= 'NGC4151           ' / proposer's target name
RA_TARG =   0.1826357541667E+03 / right ascension of the target (deg) (J2000)
DEC_TARG=   0.3940567500000E+02 / declination of the target (deg) (J2000)
```

Figure 3.9: Sprint Zero: FITS Header example (two)

```
TELESCOP= 'Gemini-North'       / Telescope
PARALLAX=                   0. / Parallax of Target
RADVEL  =                   0. / Heliocentric Radial Velocity
EPOCH   =                2000. / Epoch for Target coordinates
EQUINOX =                2000. / Equinox for Target coordinates
TRKEQUIN=                2000. / Tracking equinox
SSA     = 'Walls   '           / SSA
RA      =           105.915125 / RA of Target
DEC     =          10.77030556 / Declination of Target
ELEVATIO=     55.9472013888889 / Current Elevation
AZIMUTH =     259.745640277778 / Current Azimuth
CRPA    =     19.8191902202926 / Current Cass Rotator Position Angle
HA      = '+02:16:26.42'       / Telescope hour angle
LT      = '03:34:28.3'         / Local time at start of observation
```

Figure 3.10: Sprint Zero: FITS Header example (three)

```
ORIGIN  = 'UIT/GSFC'            / WHERE TAPE WRITTEN
ASTRO   =                     2 / ASTRO MISSION NUMBER
FRAMENO = 'b0582   '            / ANNOTATED FRAME NUMBER
CATHODE = 'CSI     '            / IMAGE TUBE PHOTOCATHODE
FILTER  = 'B1      '            / CAMERA/FILTER IDENTIFIER
PDSDATIM= '06-JUL-1995  07:20'  / MICRODENSITOMETRY DATE & TIME
PDSID   =                    21 / MICRODENSITOMETER IDENT
PDSAPERT=                    20 / MICROD. APERTURE, MICRONS
PDSSTEP =                    10 / MICROD. STEP SIZE, MICRONS
PIXELSIZ=        8.0000000E+01  / CURRENT PIXEL SIZE, MICRONS
EQUINOX =        2.0000000E+03  / EQUINOX OF BEST COORDINATES
NOMRA   =             182.0044  / 1950 I.P.S.  R.A., DEGREES
NOMDEC  =              39.6839  / 1950 I.P.S.  DEC., DEGREES
NOMROLL =             323.9500  / I.P.S. ROLL ANGLE
NOMSCALE=        5.6832500E+01  / NOMINAL PLATE SCL (ARCSEC/MM)
CALIBCON=          5.00000E-16  / PREFLIGHT LAB CALIB FOR CAMERA
FEXPTIME= '8355    '            / EXPOSURE TIME, APPLICABLE FRM
DATE-OBS= '13/03/95'            / DATE OF OBSERVATION (GMT)
```

Figure 3.11: Sprint Zero: FITS Header example (four)

Chapter 4

Sprint One

4.1 User Stories

Easy to use system	C
The system should be easy to use	
User interface design patterns are used The User Interface (UI) is simple and intuitive	

Responsive design	C
The system should be compatible with mobile and tablet devices	
With a desktop browser With a tablet With a smartphone	

Provide security measures for data access	
Several levels of security should be implemented	
User needs to sign in to access the archive Roles support Administrator role Program permissions: user and admin	

Register new users	
As an Administrator, I want to be able to register new users	
Administrator user can register new users	

Enable or disable users	
As an Administrator, I want to be able to enable or disable a user	
Administrator can enable an account Administrator can disable an account	

Self registration	
As a User, I want to be able to register myself	
A user can register through the web application Only e-mail and password are required	

Refactor importing component	
Generic, and pluggable	
It is easy to add support for new telescopes Simple mapping between attributes, header keywords and filters It can be adapted to the needs of any given telescope	

4.2 User Interface Design

Although Kernighan and Pike are right when they say: "First, graphical user interfaces are hard to create and make 'right' since their suitability and success depend strongly on human behavior and expectations. Second, as a practical matter, if a system has a user interface, there is usually more code to handle user interaction than there is in whatever algorithms do the work."[31, page 114]. Still, in the end, the success of a web application depends in no small measure on the user satisfaction by means of its interface.

Therefore, a clean and simple design is followed. Agreement with some of the Gestalt[32] laws can be identified, for instance, the law of good Gestalt, similarity, proximity, continuity or simplicity. To increase the chances of a satisfactory user experience, only the most relevant information is shown—in contrast with a cluttered user interface with an excessive use of widgets.

In addition, the design strives to be consistent and to follow standard conventions. In words of Don Norman[33, page 149]: "Consistency in design is virtuous. It means that lessons learned with one system transfer readily to others. On the whole, consistency is to be followed. If a new way of doing things is only slightly better than the old, it is better to be consistent. But if there is to be a change, everybody has to change. Mixed systems are confusing to everyone." And: "People invariably object and complain whenever a new approach is introduced into an existing array of products and systems. Conventions are violated: new learning is required. The merits of the new system are irrelevant: it is the change that is upsetting."

For this reason, several user interface design patterns are used: navigation menu with highlighted active section, notification area, user profile and logout actions, modal dialogs, and special widgets for text editing, passwords, calendar, and so on. Furthermore, other patterns are used to communicate with the user, from using colors and hiding information

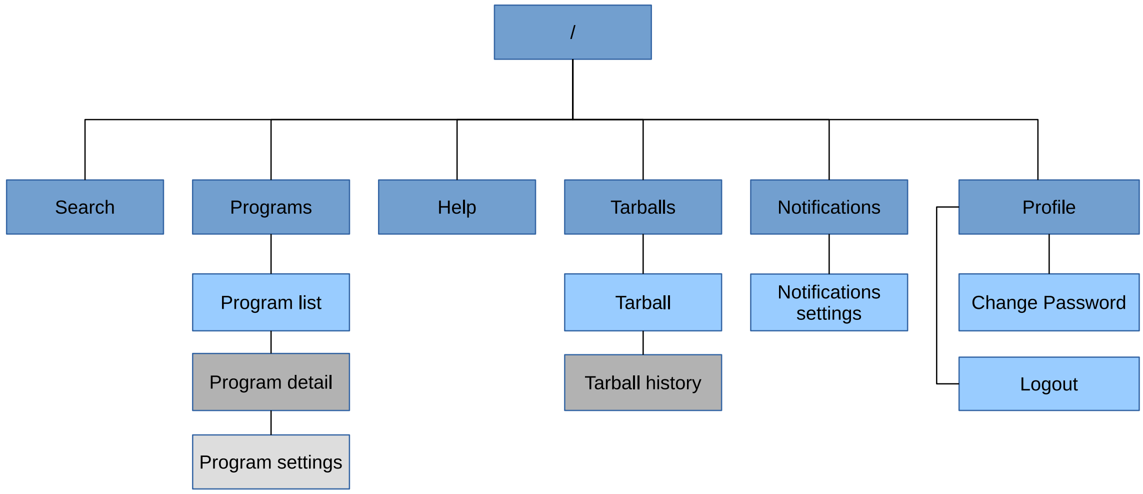


Figure 4.1: Sprint One: Sitemap

to changing the states of widgets. Figures 4.2 and 4.3 displays some of these patterns in use.

A sitemap provides the navigability through a website or web application; and a rough estimation of the number of pages or templates in use. Figure 4.1 shows the web application sitemap.

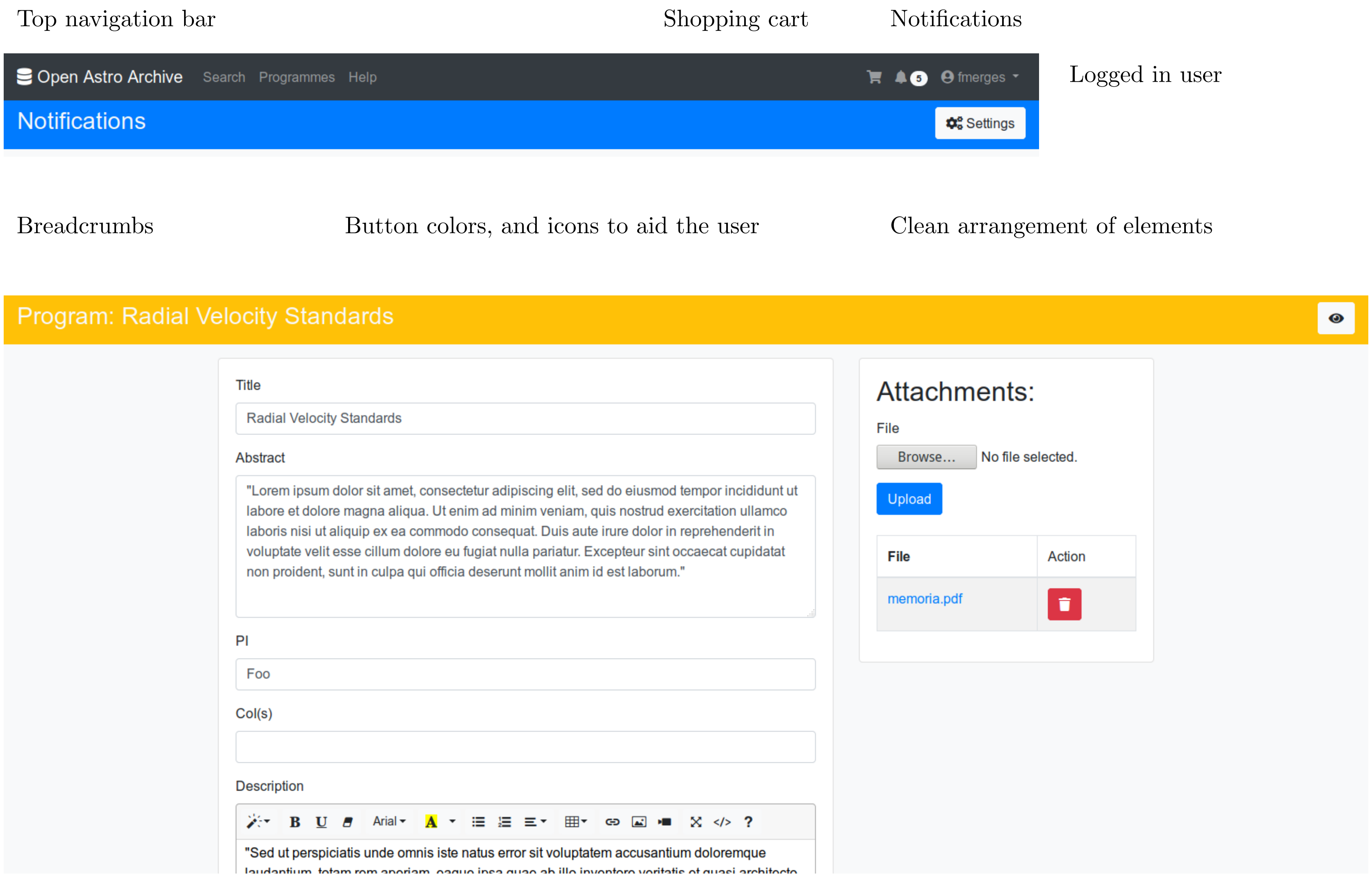
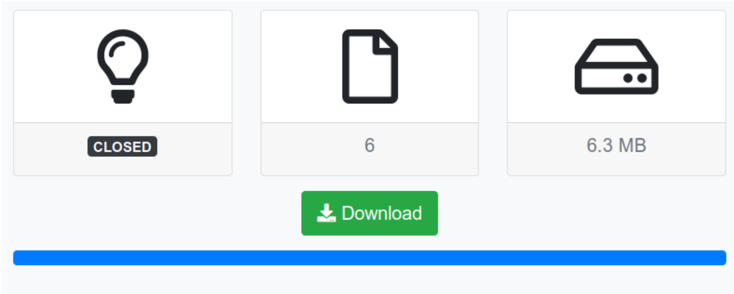
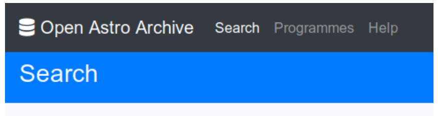
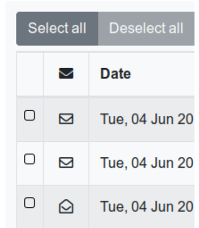


Figure 4.2: Sprint One: User interface design elements (one)

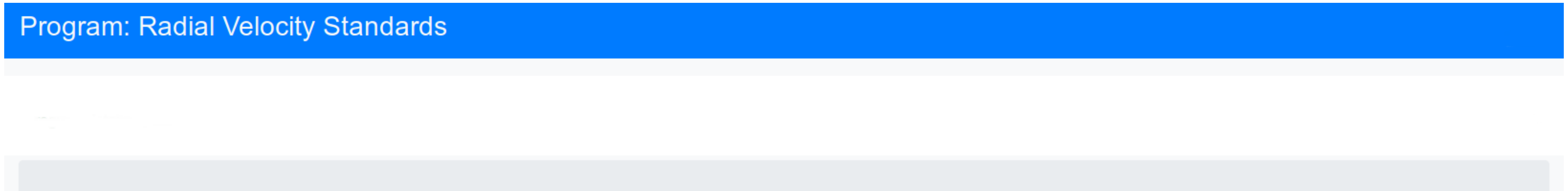




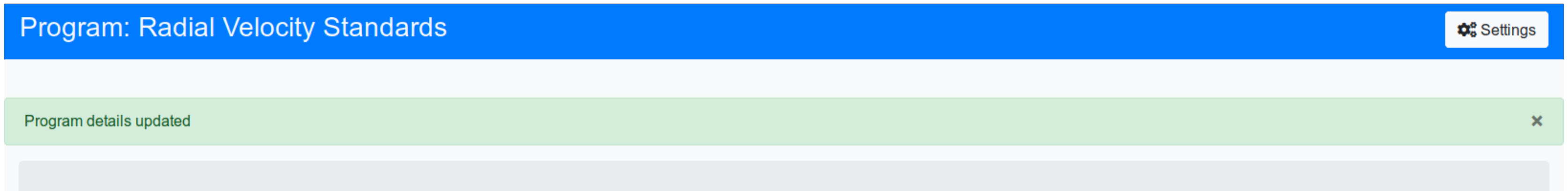
Navigation bar with active section



Usage of icons, colors, and dynamic elements



Show or hide elements based on user permissions



Notifications with different colors to indicate the status of the action

Figure 4.3: Sprint One: User interface design elements (two)

4.3 Permissions and Security

Permissions and security are better defined early on, otherwise it can be more difficult to get it right. The system's requirements are to support public and private data, and to allow PIs to manage their programs permission. Table 4.1 summarizes the access levels identified.

Level	Permission	Description
4	Admin	All of the levels below, plus user management.
3	Program Admin	Given a program, can manage its permissions and settings.
2	Program User	Given a program, can access its details and data.
1	User	User can login and search public data.

Table 4.1: Sprint One: Permission Levels

In order to accommodate the different needs identified, the system follows the Party-Role[34] pattern. The Role archetype[35] is justified by the web application (even do it doesn't have any behavior associated) to restricts the user interface, data access, and actions.

The web application uses an extension, *Flask-Security*[36], to handle authentication and authorization—also user registration, a topic for another section. This extension builds on top of several others much like a façade[37] in order to provide:

“ 1. Session based authentication
2. Role management
3. Password hashing
4. Basic HTTP authentication
5. Token based authentication
6. Token based account activation (optional)
7. Token based password recovery / resetting (optional)
8. User registration (optional)
9. Login tracking (optional)
10. JSON/Ajax Support ”[36]

Flask-Security imposes some constraints on the domain model. To begin with, it requires the model to have two classes, one representing users and another representing roles. In addition, these two classes need to include some behavior via mixin[1]. And finally, the users class needs to have two attributes for the login procedure: email and password.

The basic role support provided by *Flask-Principal*, the extension used by *Flask-Security*, is suitable for the *Admin* role but not for the other two. In order to provide support for these, *Program User* and *Program Admin*, two new classes are required by *Flask-Principal*, one defines the need that can be requested for, and a second one for the permission itself. Furthermore, a new class is required in order to express and persist this new user permission between users and programs.

See Section A.2 for a detailed explanation on how the permissions can be checked at the controller level.

[1] Mixin “[c]lasses [...] are designed to provide concrete methods to other classes via multiple inheritance.”[38]

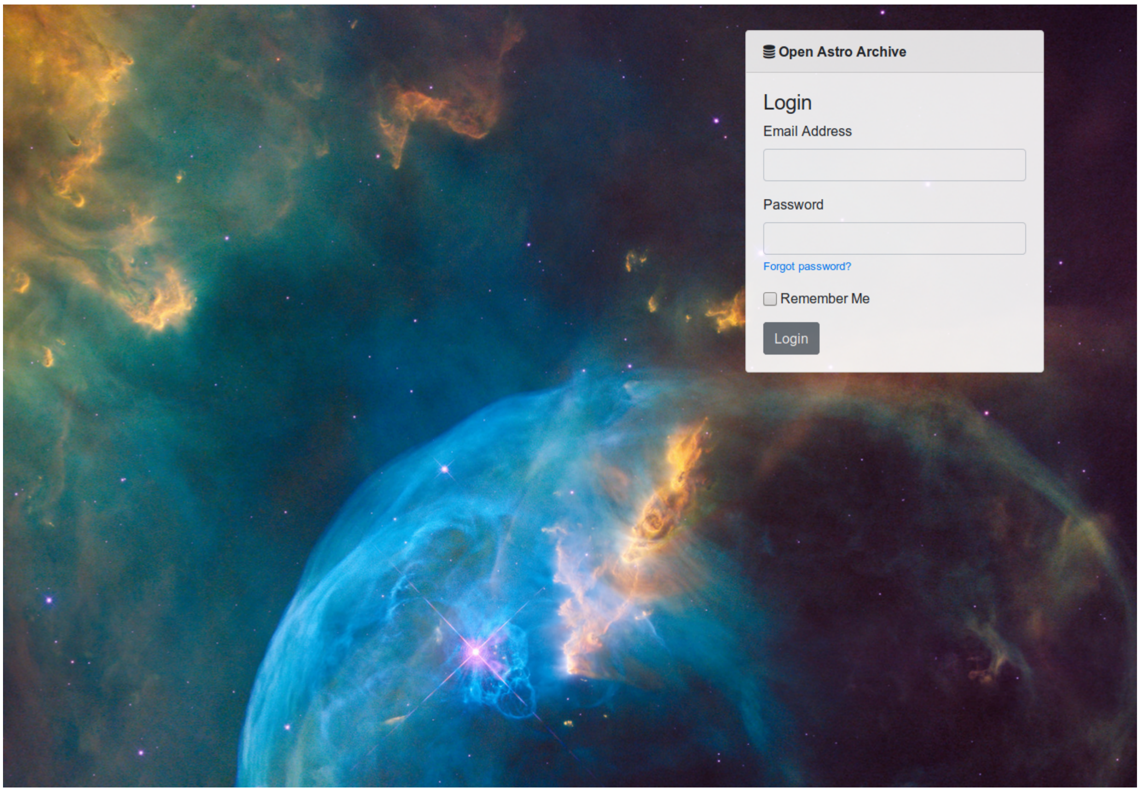


Figure 4.4: Sprint One: User login screenshot

4.4 User Management

User management is handled through the *Flask-Security* extension. Two user registration options are available. First option is self registration via the web application. Second, using the command line *flask* script. The later can be used for enabling and disabling users as well.

Flask-Security ships with a basic set of templates that were overridden in order to fit the overall web application design. Figure 4.4 shows the customized login screen.

4.5 Domain Model

Three classes were added:

- `User`
- `Role`
- `UserProgramPermission`

Figure 4.5 shows an excerpt of the domain model class diagram with the user, and role classes required to support the different user access levels defined in Section 4.3.

4.6 Data Import

After some refactoring, a new data import component is available. Previously, it was closely tied to the test-dataset, now, it consist of a minimalistic framework.

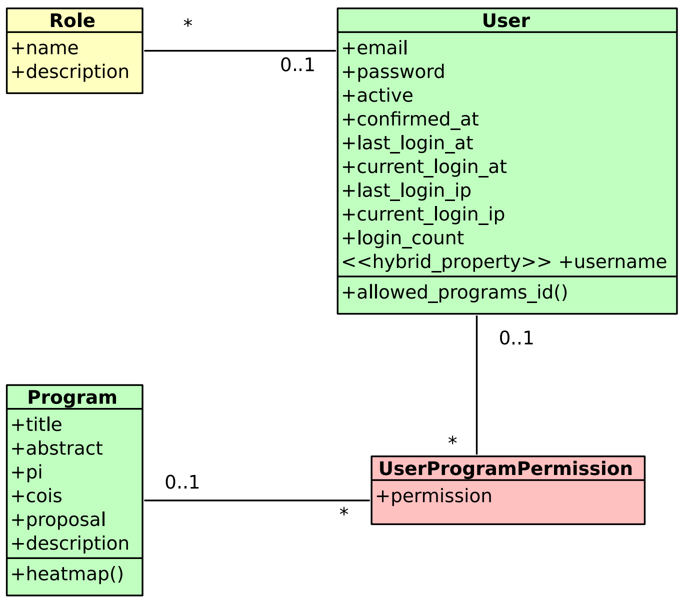


Figure 4.5: Sprint One: Domain model, user and roles class diagram

The framework orchestrates the overall data import leaving some margin for each telescope[2] module to suit its particular needs. By means of inheritance, and the use of the template method[37] pattern, it provides several hooks to tailor the data import process for each telescope, and its instruments. Figures 4.6, and 4.7 illustrate this new design by showing the most relevant aspects of it.

Adding support for a given telescope is straightforward. First, an abstract class needs to be implemented for the particular data type—raw or reduced. Second, the class needs to be registered into the system via the plugin registry. (The benefit of using a plugin registry is that the core of the importing system doesn't need to be changed when a new telescope module is added to the system.) The module registry can be done in two ways, either by decorating[39] the class, or by calling the registry directly.

In conclusion, thanks to this new design and the flexibility it offers, even the most tricky scenarios shall be supported.

4.7 Retrospective

The general user interface, navigation, and sitemap are defined. Users can register themselves through the web user interface; only a valid e-mail address and password are required. Administrators can register new users and activate them using the command line tool. The same tool can be used to disable an existing user. The data import component was completely refactored, althought its external interface was kept, thus the importing tool did not require any changes.

[2]The term telescope is used although in reality it should be seen as a data source.

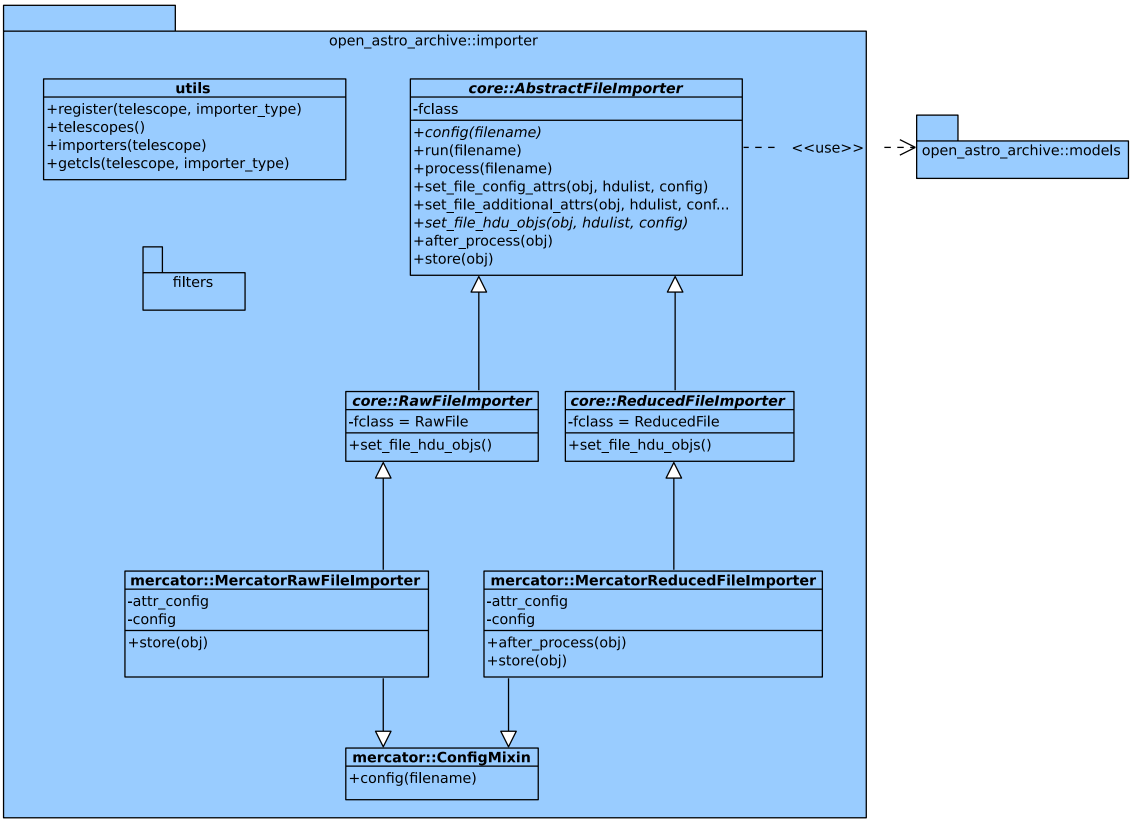


Figure 4.6: Sprint One: Data import class diagram

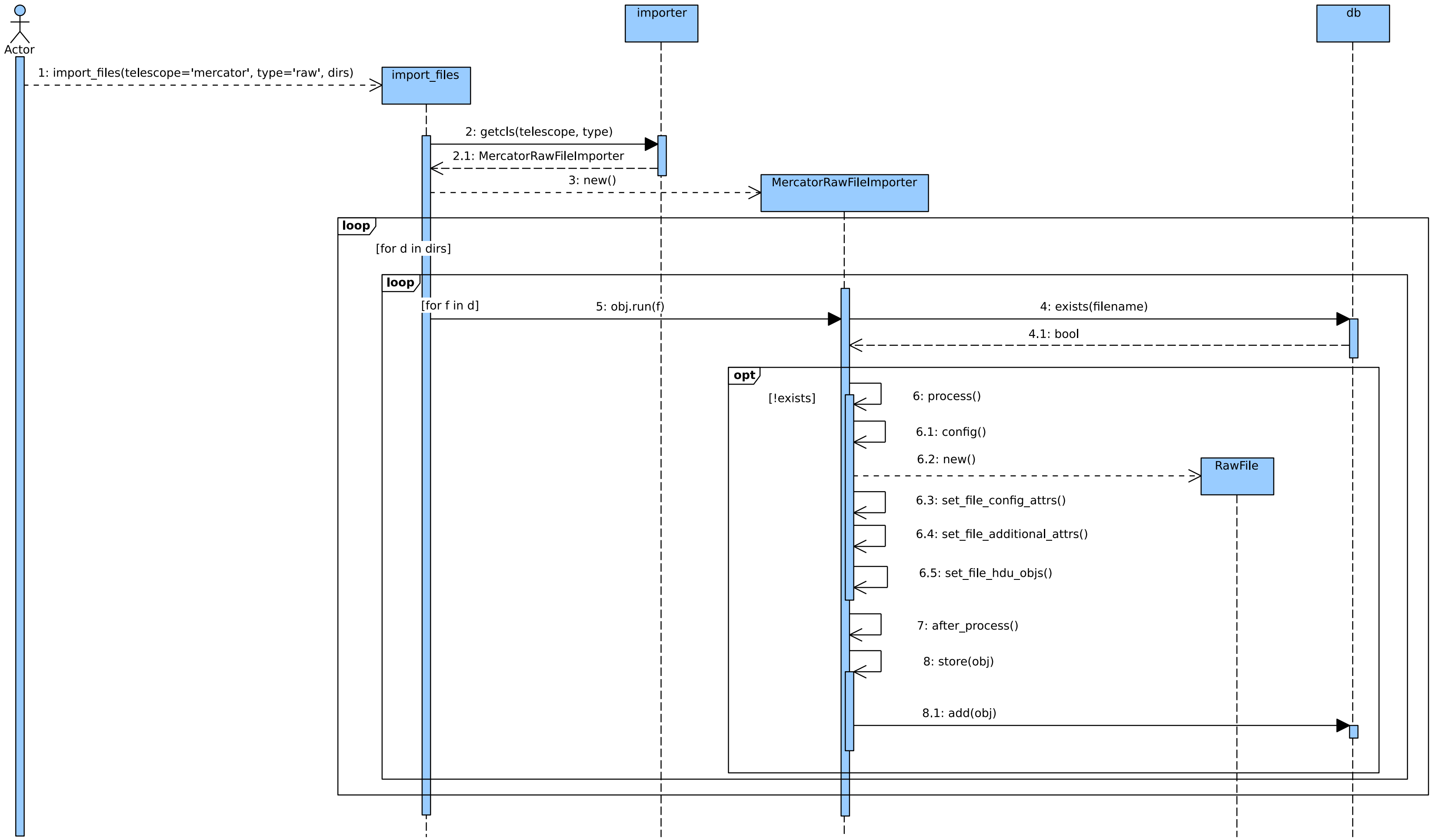


Figure 4.7: Sprint One: Data import sequence diagram

Chapter 5

Sprint Two

5.1 User Stories

Data search	
As a User, I want to be able to search for data by object name, coordinates, instrument, and program	
User can search through public data The list of programs only include those for which the user has permissions The results can be ordered by different attributes The results can be exported to different formats	

Cone search	
As a User, I want to be able to make a cone search	
Cone search returns only objects inside the coordinates radius	

Raw file view	
As a User, I want to see the detail of a raw file	
All the attributes are displayed	

End of Night Report	
As a User, I want to have access to the end of night report for a raw file	
Handles correctly if an end of night is not available	

Finding Chart	
As a User, I want to have access to the finding chart for a given raw file	
The finding chart matches the astronomical object A link to *Simbad* is provided	

Raw header display	
As a User, I want to be able to see the raw header for a raw file	
Multi-header support	

Analyze Reduced Spectrum	
As a User, I want to be able to analyze the reduced spectrum, if available, for a given raw file	
The speed is acceptable Gzip and deflate is used Zoom, pan and reset options are available	

5.2 Data Search

Searching the archive is the most relevant action a user wants to do, downloading data being second. Hence, it is very important to provide a user friendly search interface.

Users want to search data by object name[1], coordinates, date range, instrument, program, or by all of them. Searching by date range, instrument, or program is straightforward. But, it turns out that the object name is not a reliable search criteria because:

1. an object can have several names. It may have been cataloged more than once, and each catalog has its own naming conventions.
2. the object name in a FITS header is assigned by the observer during the observation, or by the PI as part of the submitted observing proposal. Meaning, it can be literally anything.

A solution is to rely on the object's coordinates[2]. The coordinate system most widely used in astronomy to identify objects is the equatorial system, it uses two angles to locate the astronomical object on the celestial sphere: right ascension (RA), and declination (DEC).

However, providing only coordinate search is a bit cumbersome because it forces the user to look up the coordinates of the object. In order to make the search experience more pleasant, a service was added to lookup in *Simbad*[3] by object name. The service returns a list of possible object names, and as soon as the user confirms one, the coordinates fields of the search form are populated—thus avoiding the tedious task just mentioned.

[1]The name of a star or stellar object
[2]The system assumes the object coordinates are in Epoch J2000.
[3]A database containing information for several million astronomical objects. See [40]

	Data Search
Goal Level:	Sea Level
Primary Actor:	User
Preconditions:	User is logged in

Main Success Scenario:

1. The system provides a search form to the user.
2. The user submits the search.
3. The system searches the archive taking the user's permission into account.
4. The system returns the results of the search.

Extensions:

1.a User enters an object name:

1. The system queries *Simbad* catalog.
2. The system returns the astronomical objects options.
3. The user selects the astronomical object from the list.
4. The browser fills the object coordinates of the search form.

Figure 5.1: Sprint Two: Data search use case

"A cone search extracts all the objects within a user specified radius around a single point in the sky"[41]. The search form has already fields to introduce the coordinates, thus by extending it, both type of searches can be combined into one; provided a small initial radius is set. Figure 5.1 shows a summary of the data search use case.

The search form is made up of the following fields: a name field for object coordinates lookup; RA, DEC, and radius for coordinates search, and cone search; from and through dates to restrict the time interval; and some multi fields for instruments and programs. The list of available programs to search upon depend on the user's role and permissions. See Figure 5.2.

Once the search request is received on the backend, and the form data is validated, an archive search is carried out. The search takes into account: user roles, program permissions, and the public flag of files. In case the form data includes RA and DEC coordinates, the database query uses a function provided by the *Q3C* extension to perform a cone search. A cone search requires some spherical trigonometry to calculate the distance between the user provided coordinates and the coordinates of all the available files in the archive; doing this as part of a database query is not so efficient. *Q3C* uses indexes to speed things up, and at the same time it hides the mathematics required for the search. Figure 5.3 shows the results of a search. Figure 5.4 shows an activity diagram of the search process.

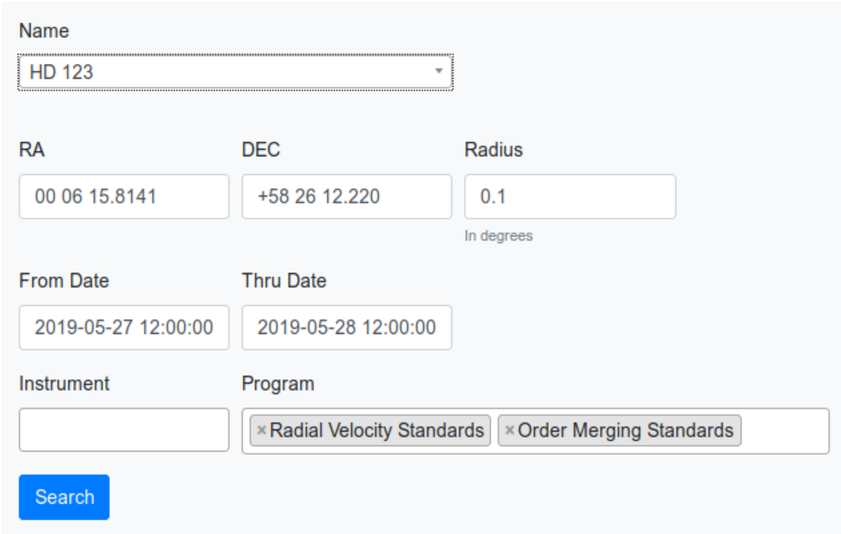
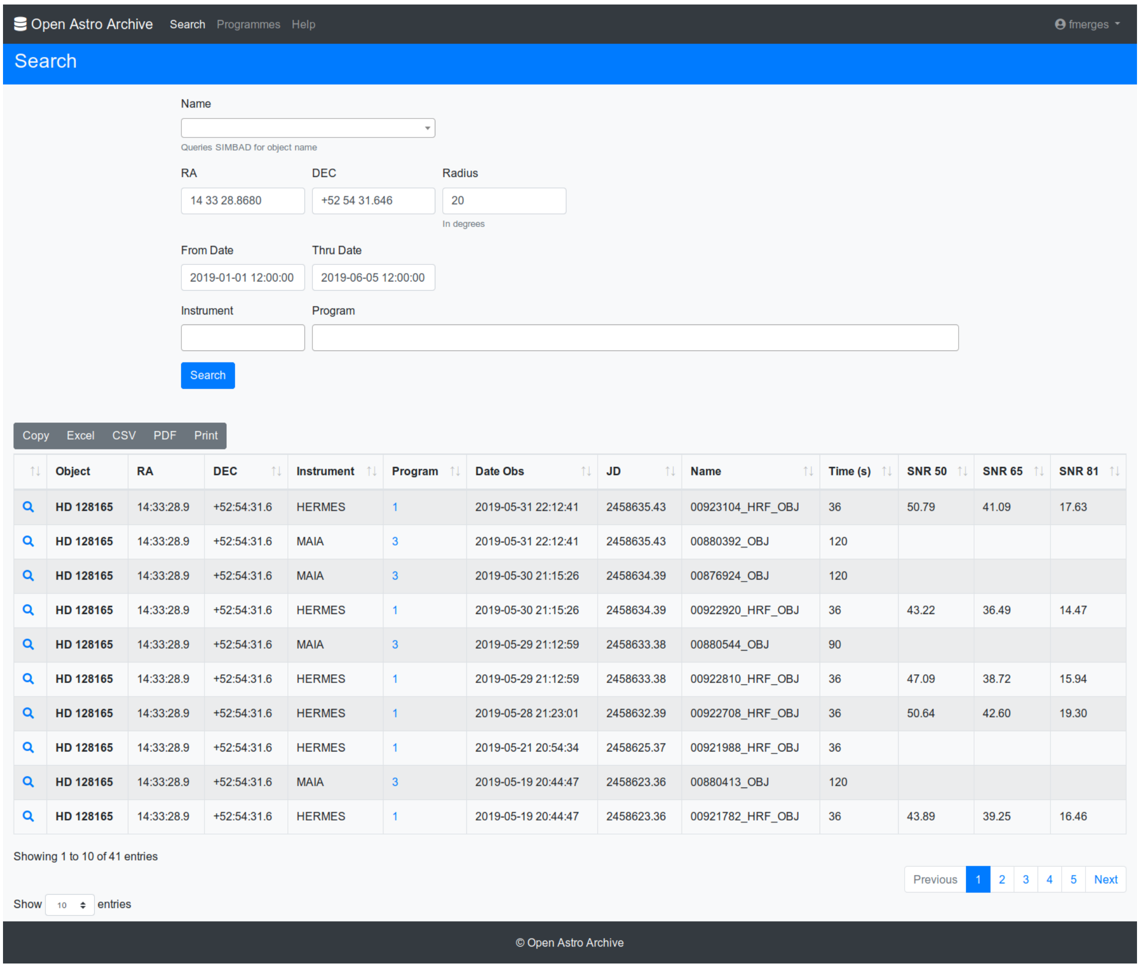


Figure 5.2: Sprint Two: Search form screenshot



	Object	RA	DEC	Instrument	Program	Date Obs	JD	Name	Time (s)	SNR 50	SNR 65	SNR 81
	HD 128165	14:33:28.9	+52:54:31.6	HERMES	1	2019-05-31 22:12:41	2458635.43	00923104_HRF_OBJ	36	50.79	41.09	17.63
	HD 128165	14:33:28.9	+52:54:31.6	MAIA	3	2019-05-31 22:12:41	2458635.43	00880392_OBJ	120			
	HD 128165	14:33:28.9	+52:54:31.6	MAIA	3	2019-05-30 21:15:26	2458634.39	00876924_OBJ	120			
	HD 128165	14:33:28.9	+52:54:31.6	HERMES	1	2019-05-30 21:15:26	2458634.39	00922920_HRF_OBJ	36	43.22	36.49	14.47
	HD 128165	14:33:28.9	+52:54:31.6	MAIA	3	2019-05-29 21:12:59	2458633.38	00880544_OBJ	90			
	HD 128165	14:33:28.9	+52:54:31.6	HERMES	1	2019-05-29 21:12:59	2458633.38	00922810_HRF_OBJ	36	47.09	38.72	15.94
	HD 128165	14:33:28.9	+52:54:31.6	HERMES	1	2019-05-28 21:23:01	2458632.39	00922708_HRF_OBJ	36	50.64	42.60	19.30
	HD 128165	14:33:28.9	+52:54:31.6	HERMES	1	2019-05-21 20:54:34	2458625.37	00921988_HRF_OBJ	36			
	HD 128165	14:33:28.9	+52:54:31.6	MAIA	3	2019-05-19 20:44:47	2458623.36	00880413_OBJ	120			
	HD 128165	14:33:28.9	+52:54:31.6	HERMES	1	2019-05-19 20:44:47	2458623.36	00921782_HRF_OBJ	36	43.89	39.25	16.46

Figure 5.3: Sprint Two: Search results screenshot

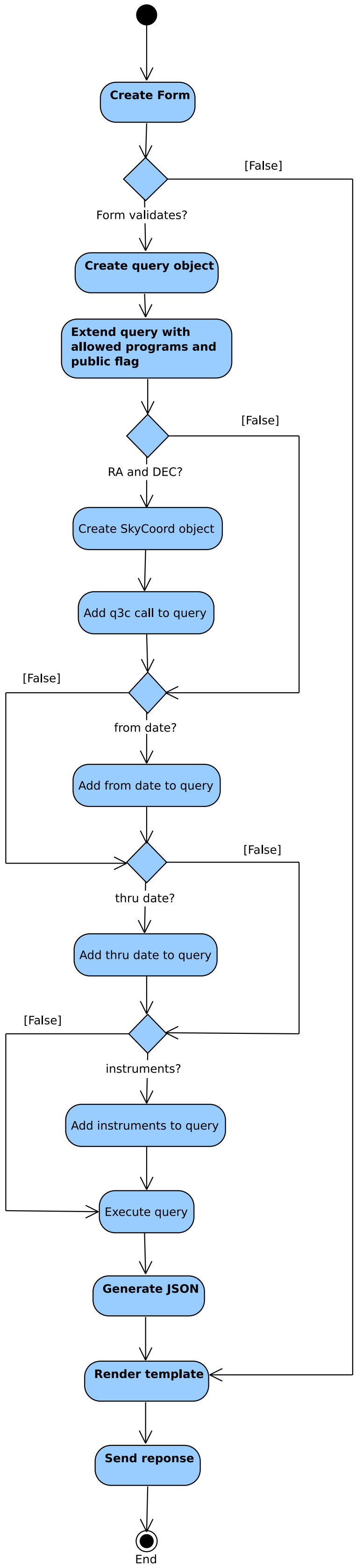


Figure 5.4: Sprint Two: Search activity diagram

	Raw File View
Goal Level:	Sea Level
Primary Actor:	User
Preconditions:	User is logged in

Main Success Scenario:

1. The user selects a raw file to view from the search results.
2. The system lookups the given raw file.
3. The system checks if the user can view file.
4. The system returns the results.

Extensions:

2.a Raw file does not exists:

1. The system sends back the error to the client.

3.a No permissions:

1. The system sends back the error to the client.

Figure 5.5: Sprint Two: Raw file view use case

5.3 Raw File Details

Typically after a data search, a user is interested in knowing the details of a raw file. See if there is reduced data available, inspect the FITS header, read the observer's comments, check the finding chart to confirm it's the right object, or have access to the end of night report submitted by the observer in order to have a better idea of the observing conditions when the data was taken. Figure 5.5 shows a summary of the raw file view use case.

From an User Experience (UX) point of view, all relevant information and actions were combined in a single view, from which the user can[4]:

- see the extracted metadata of a raw file.
- see the finding chart with a link to *Simbad*.
- access to the observing program.
- inspect the reduced spectrum.
- see the raw FITS header.
- see the list of reduced files.
- access the end of night report.

Figure 5.6 and Figure 5.7 shows a screenshot of the raw file view.

The spectrum plot uses a widget allowing the user to pan and zoom. All the data points of the spectrum are send, meaning the data is not averaged. Several different plotting libraries were tested until one was found with an acceptable performance, as a typical spectrum contains many datapoints. By enabling compression on the webserver, with the current internet access speeds, the download time required to visualize a plot is acceptable. The data itself for the plot is downloaded on demand using AJAX[5]. For the time being, only the reduced spectrum plot is available. In future, the idea is to move the plotting responsibility to the data source or telescope module, this way each module controls how its data is plotted.

Having access to the full header allows a user to find a value that might not be captured already as part of the extracted metadata. In case the header has several header units, they are made available to the user through tabs.

5.4 Retrospective

General search, coordinate, and cone search are implemented. Raw file view is implemented, including finding chart, plot, headers, reduced files and the end of night report.

The end of night report and the reduced spectrum plot are tied to the telescope or data source of the test-dataset. In future, by means of the strategy[37] pattern, each data source will have it own logic.

The velocity at the end of this sprint suffered a little due to public holidays. The initial time planning contemplated already two slack days between this sprint and the upcoming one, but at the end they turned into four.

[4] Not all of the options are available for every raw file, for instance, an imaging raw file is not a spectrum, or there might not be reduced data for a given raw file.

[5] "AJAX stands for Asynchrnonous JavaScript And XML [...]. The basic idea behind AJAX is that your application can send and receive information to other computers without reloading the web page."[42]

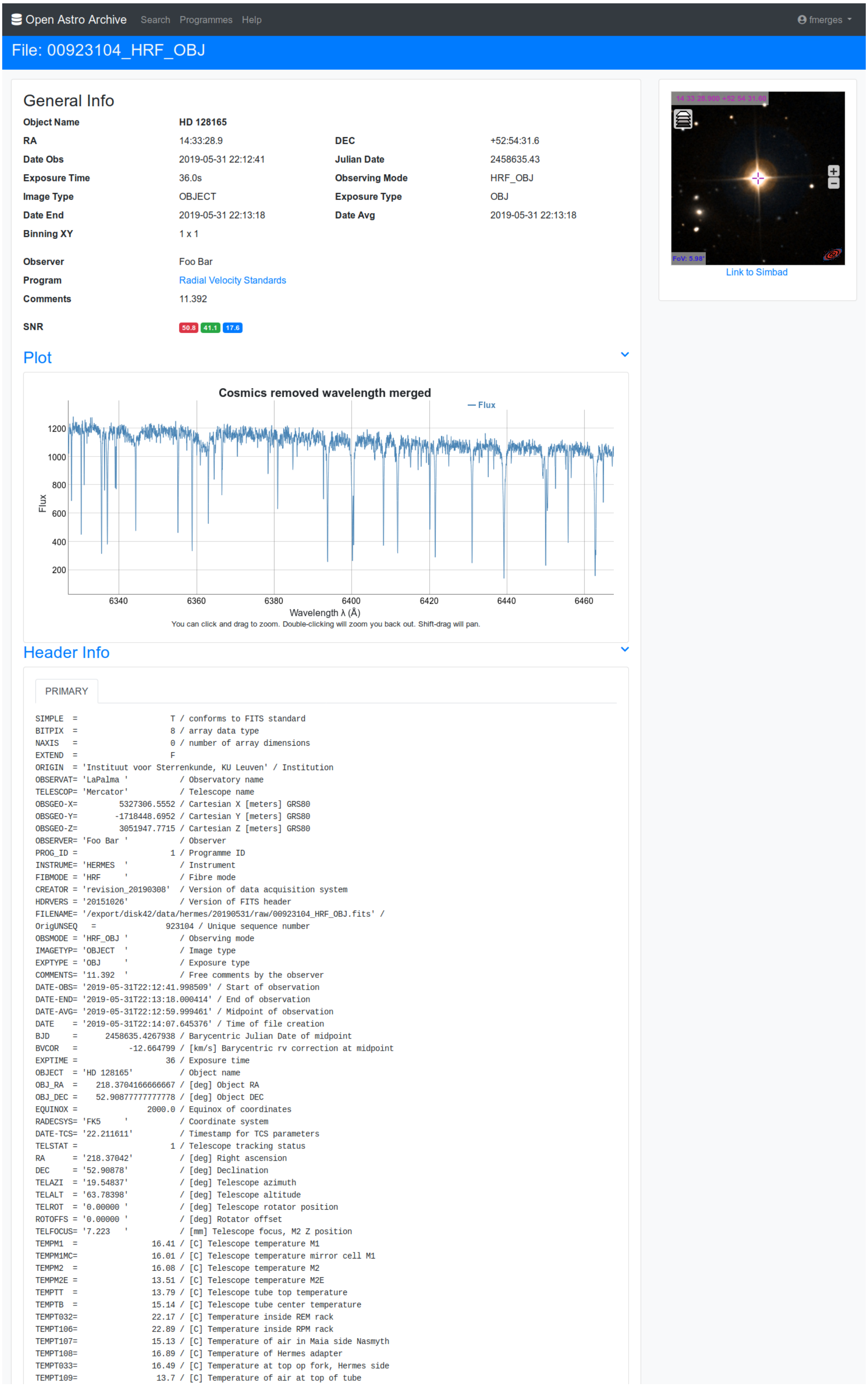
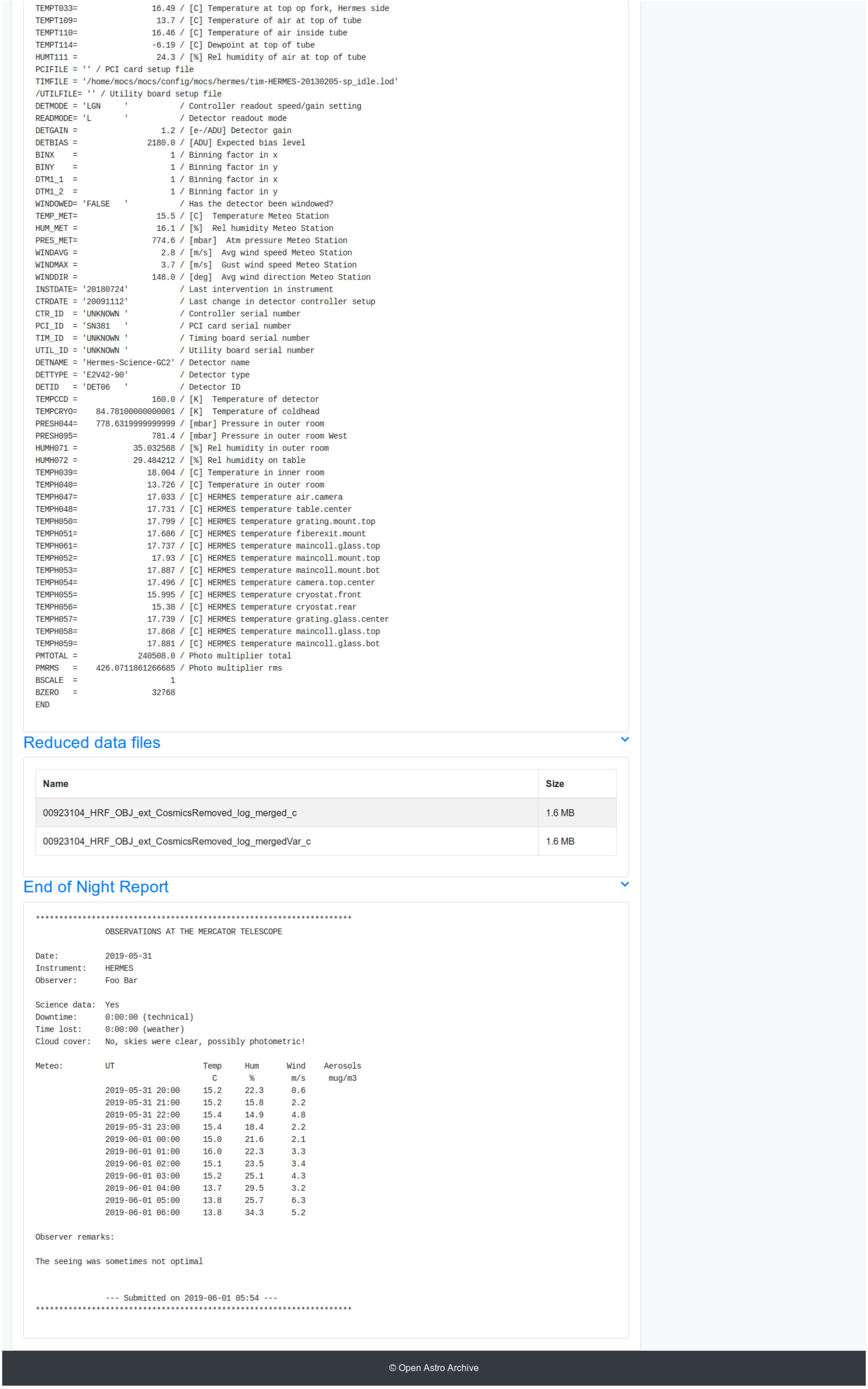


Figure 5.6: Sprint Two: Raw file view screenshot (one)



```
TEMPT033=                16.49 / [C] Temperature at top op fork, Hermes side
TEMPT109=                 13.7 / [C] Temperature of air at top of tube
TEMPT110=                16.46 / [C] Temperature of air inside tube
TEMPT114=                -6.19 / [C] Dewpoint at top of tube
HUMT111 =                 24.3 / [%] Rel humidity of air at top of tube
PCIFILE = '' / PCI card setup file
TIMFILE = '/home/mocs/mocs/config/mocs/hermes/tim-HERMES-20130205-sp_idle.lod'
/UTILFILE= '' / Utility board setup file
DETMODE = 'LGN     '           / Controller readout speed/gain setting
READMODE= 'L       '           / Detector readout mode
DETGAIN =                  1.2 / [e-/ADU] Detector gain
DETBIAS =               2180.0 / [ADU] Expected bias level
BINX    =                    1 / Binning factor in x
BINY    =                    1 / Binning factor in y
DTM1_1  =                    1 / Binning factor in x
DTM1_2  =                    1 / Binning factor in y
WINDOWED= 'FALSE   '           / Has the detector been windowed?
TEMP_MET=                 15.5 / [C]  Temperature Meteo Station
HUM_MET =                 16.1 / [%]  Rel humidity Meteo Station
PRES_MET=                774.6 / [mbar]  Atm pressure Meteo Station
WINDAVG =                  2.8 / [m/s]  Avg wind speed Meteo Station
WINDMAX =                  3.7 / [m/s]  Gust wind speed Meteo Station
WINDDIR =                148.0 / [deg]  Avg wind direction Meteo Station
INSTDATE= '20180724'           / Last intervention in instrument
CTRDATE = '20091112'           / Last change in detector controller setup
CTR_ID  = 'UNKNOWN '           / Controller serial number
PCI_ID  = 'SN381   '           / PCI card serial number
TIM_ID  = 'UNKNOWN '           / Timing board serial number
UTIL_ID = 'UNKNOWN '           / Utility board serial number
DETNAME = 'Hermes-Science-GC2' / Detector name
DETTYPE = 'E2V42-90'           / Detector type
DETID   = 'DET06   '           / Detector ID
TEMPCCD =                160.0 / [K]  Temperature of detector
TEMPCRYO=    84.78100000000001 / [K]  Temperature of coldhead
PRESH044=    778.6319999999999 / [mbar] Pressure in outer room
PRESH095=                781.4 / [mbar] Pressure in outer room West
HUMH071 =            35.032588 / [%] Rel humidity in outer room
HUMH072 =            29.484212 / [%] Rel humidity on table
TEMPH039=               18.004 / [C] Temperature in inner room
TEMPH040=               13.726 / [C] Temperature in outer room
TEMPH047=               17.033 / [C] HERMES temperature air.camera
TEMPH048=               17.731 / [C] HERMES temperature table.center
TEMPH050=               17.799 / [C] HERMES temperature grating.mount.top
TEMPH051=               17.686 / [C] HERMES temperature fiberexit.mount
TEMPH061=               17.737 / [C] HERMES temperature maincoll.glass.top
TEMPH052=                17.93 / [C] HERMES temperature maincoll.mount.top
TEMPH053=               17.887 / [C] HERMES temperature maincoll.mount.bot
TEMPH054=               17.496 / [C] HERMES temperature camera.top.center
TEMPH055=               15.995 / [C] HERMES temperature cryostat.front
TEMPH056=                15.38 / [C] HERMES temperature cryostat.rear
TEMPH057=               17.739 / [C] HERMES temperature grating.glass.center
TEMPH058=               17.868 / [C] HERMES temperature maincoll.glass.top
TEMPH059=               17.881 / [C] HERMES temperature maincoll.glass.bot
PMTOTAL =             240508.0 / Photo multiplier total
PMRMS   =    426.0711861266685 / Photo multiplier rms
BSCALE  =                    1
BZERO   =                32768
END
```

Reduced data files

Name	Size
00923104_HRF_OBJ_ext_CosmicsRemoved_log_merged_c	1.6 MB
00923104_HRF_OBJ_ext_CosmicsRemoved_log_mergedVar_c	1.6 MB

End of Night Report

```
********************************************************************
              OBSERVATIONS AT THE MERCATOR TELESCOPE

Date:         2019-05-31
Instrument:   HERMES
Observer:     Foo Bar

Science data:  Yes
Downtime:     0:00:00 (technical)
Time lost:    0:00:00 (weather)
Cloud cover:  No, skies were clear, possibly photometric!

Meteo:        UT                    Temp     Hum      Wind    Aerosols
                                     C        %        m/s     mug/m3
              2019-05-31 20:00      15.2     22.3     0.6
              2019-05-31 21:00      15.2     15.8     2.2
              2019-05-31 22:00      15.4     14.9     4.8
              2019-05-31 23:00      15.4     18.4     2.2
              2019-06-01 00:00      15.0     21.6     2.1
              2019-06-01 01:00      16.0     22.3     3.3
              2019-06-01 02:00      15.1     23.5     3.4
              2019-06-01 03:00      15.2     25.1     4.3
              2019-06-01 04:00      13.7     29.5     3.2
              2019-06-01 05:00      13.8     25.7     6.3
              2019-06-01 06:00      13.8     34.3     5.2

Observer remarks:

The seeing was sometimes not optimal


              --- Submitted on 2019-06-01 05:54 ---
********************************************************************
```

© Open Astro Archive

Figure 5.7: Sprint Two: Raw file view screenshot (two)

Chapter 6

Sprint Three

6.1 User Stories

Program Settings	
As a PI/Co-I, I want to define details of my programs	
All the programs attributes are updated Text editor works correctly Program permissions are checked	

Program Permissions	
As a PI/Co-I, I want to manage the user access to my programs	
Verify the user can search and select an user The list of users should not include: • users with administrator role • the current user • users already with any of both program permissions Program permissions are checked for changing permissions	

Program Progress	
As a PI/Co-I, I want to see the progress of my programs	
The sum aggregate for the days and data observed is correct The calculation is fast enough or if caching is needed Program permission are checked	

Program Attachments	
As a PI/Co-I, I want to be able to attach files to a program	
The user can upload and delete attachments The user can upload a file with the same name Permissions are checked when uploading or downloading attachments	

Select data to download	
As a User, I want to select the data to download	
The download data set can be chosen Files can be added and removed from a cart The cart can be emptied Permissions are checked when adding data	

Data Tarball	
As a User, I want to download the data as a tarball	
The tarball is created correctly The tarball is send as an attachment Permissions are checked for downloading the tarball	

6.2 Program Details

One of the epics of the project besides being able to search and retrieve data is, to provide a facility for a PI (and Co-Is) of a program to manage the access to their program and its data. Part of the epic includes as well to be able to: define details of a program, and keep track of its progress. In addition, an authorized user should be able to add attachments to a program.

From an UX point of view, similar to the raw file details view, see Section 5.3, all relevant information and actions were combined in a single view.

A PI can engage with the users by uploading attachments and create content for the program page, e.g, uploading some publication data, plots, news, etc. There is no limitation on the number of attachments possible, only the size of each to avoid misuse.

By keeping track of a program is meant, to be able to see when data has been taken for the program in an intuitive an simple way. This is accomplished by a simple heatmap widget. It uses colors and labels to indicate the sum aggregate of the data available for a given program for a certain time period.

Figure 6.1 shows the use case diagram for a program. The authorized user corresponds to a user that satisfies the *Program User* role. See Section 4.3.

Figures 6.2, and 6.3 give a short summary of the written specification for the *Update Program Details*, and *Add Program Permission* use cases.

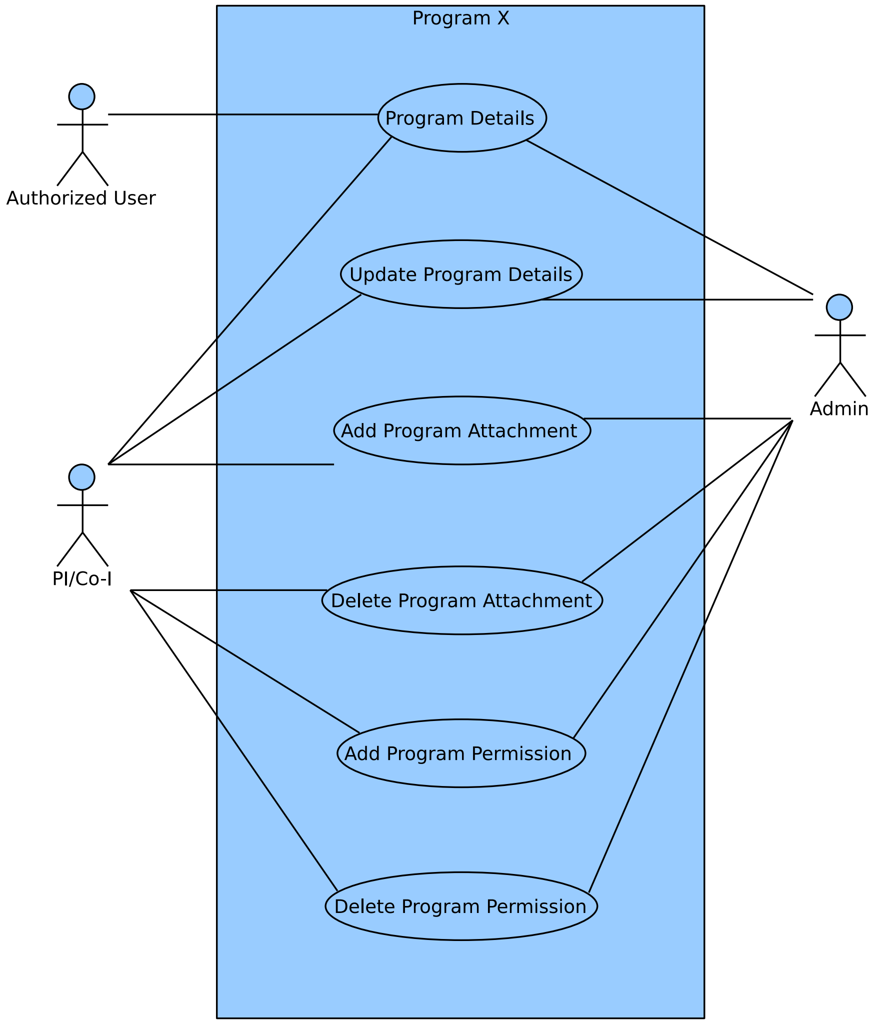


Figure 6.1: Sprint Three: Program use case diagram

	Update Program Details
Goal Level:	Sea Level
Primary Actor:	PI, Co-I, or Admin
Preconditions:	User is logged in

Main Success Scenario:

1. The user follows the link from the program details view.
2. The system looks up the given program.
3. The system checks the user can update the program.
4. The system provides an edit form with the relevant content.
5. The user makes the changes and submits the edit.
6. The system looks up the given program.
7. The system checks the user can update the program.
8. The system updates the program details based on the user submitted data.
9. The system notifies the user about the changes.

Extensions:

2.a Program does not exists:

1. The system sends back the error to the client.

3.a No permissions:

1. The system sends back the error to the client.

6.a Program does not exists:

1. The system sends back the error to the client.

7.a No permissions:

1. The system sends back the error to the client.

8.a Form validation fails:

1. The system provides an edit form with the relevant content, and errors.

Figure 6.2: Sprint Three: Update program details use case

	Add Program Permission
Goal Level:	Sea Level
Primary Actor:	PI, Co-I, or Admin
Preconditions:	User is logged in

Main Success Scenario:

1. The user follows the link from the program details view.
2. The system looks up the given program.
3. The system checks the user can update the program.
4. The system provides an select form with a list of users, excluding the user itself, and the users that have already any permission on the program.
5. The user select the user and the permission, and submits the edit.
6. The system looks up the given program.
7. The system checks the user can update the program.
8. The system updates the program permissions based on the user submitted data.
9. The system notifies the user about the changes.

Extensions:

2.a Program does not exists:

  1. The system sends back the error to the client.

3.a No permissions:

  1. The system sends back the error to the client.

6.a Program does not exists:

  1. The system sends back the error to the client.

7.a No permissions:

  1. The system sends back the error to the client.

8.a Form validation fails:

  1. The system provides a select form with the relevant content, and errors.

Figure 6.3: Sprint Three: Add program permission use case

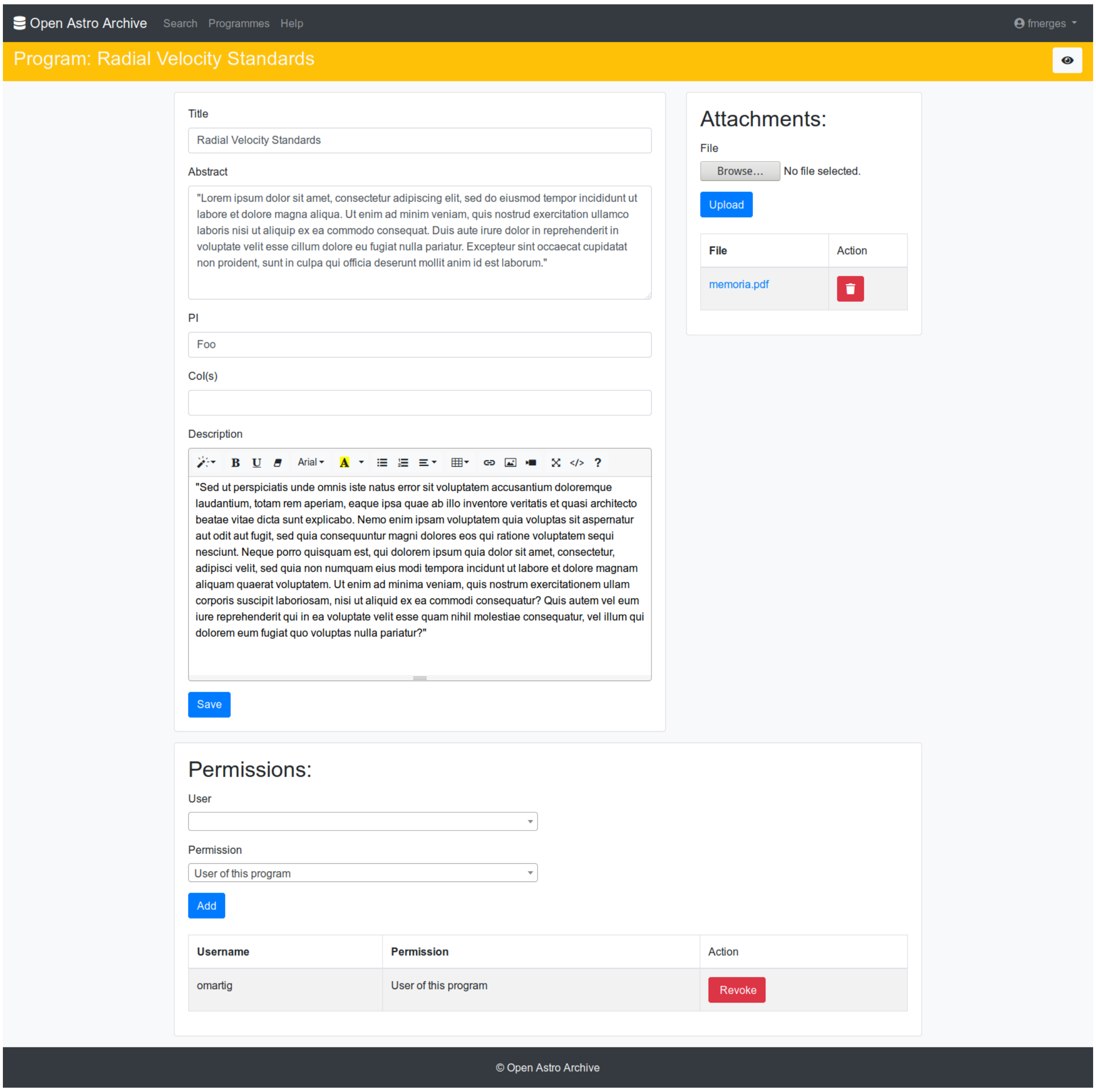


Figure 6.4: Sprint Three: Update program details screenshot

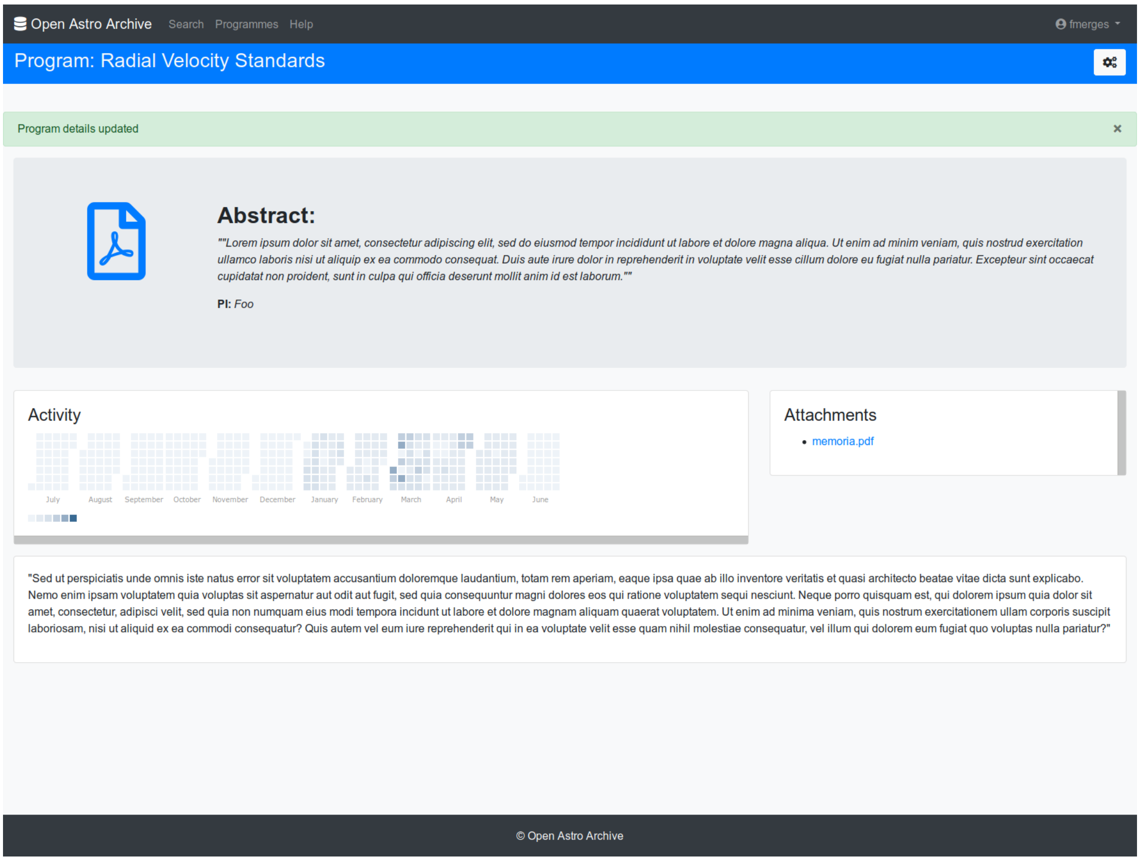


Figure 6.5: Sprint Three: Program details screenshot

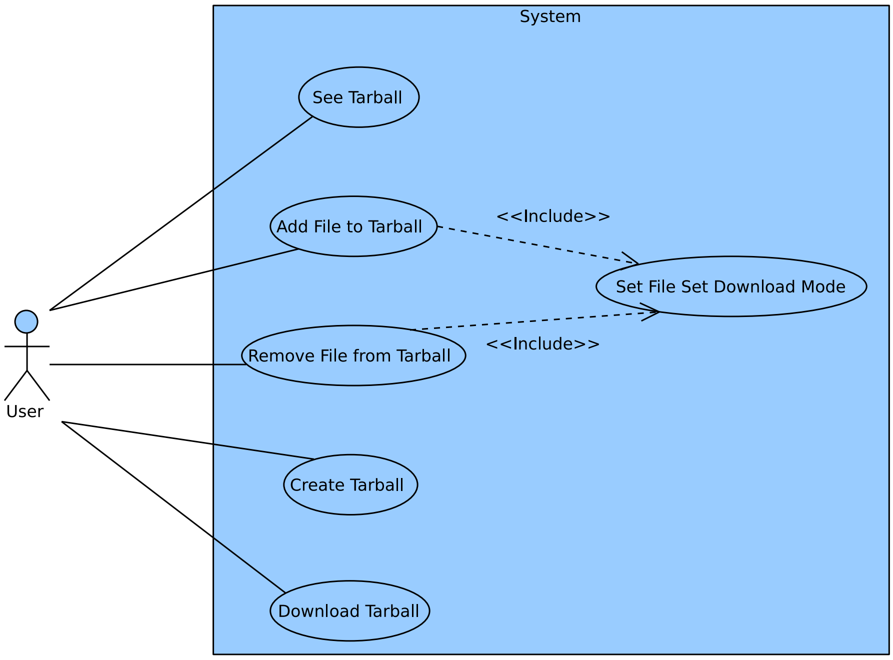


Figure 6.6: Sprint Three: Download use case diagram

6.3 Data Download

Being able to download data is one of the key aspects of any archive. Astronomical data tends to be of considerable size, thus the user requirement to be able to download the selected files packed into a tarball. Files are added in sets to a tarball.

The file sets are determined by the download mode. Table 6.3 reflects the possible download modes for a given raw file. The size of a tarball is determined by the number of files it contains, and indirectly by the characteristics of the instrument(s) that produced the data. Figure 6.6 shows a use case diagram related to data download.

Mode	Flag	Tarball Action
Raw file	F	Raw file is added
Reduced files	R	Reduced files are added
Raw and reduced files	F+R	Raw and reduced files are added
Remove or reset		Raw and reduced files are removed

Table 6.3: Sprint Three: Download Modes

The data download feature uses the Shopping Cart[43] design pattern in an attempt to follow Krug's first law of usability: "Don't make me think!"[44]. Shopping Cart is a pattern everybody is used to, hence it is the perfect match for this purpose. A user simply adds file sets to the cart, afterwards creates the tarball, and in the end downloads it. See Figure 6.7

Figure 6.8 gives a short summary of the written specification for the *Add File to Tarball* use case. The other use cases are quite similar, all of them ensure the user is logged in, the user is the owner of the tarball, and in case of adding files to a tarball it filters the files based on the user's permissions.

Generating a tarball takes some time, it doesn't fit the request-response pattern of web applications, hence they are generated asynchronously by means of a task queue. When a user confirms a tarball, a new task is enqueued, and as soon as a worker is available

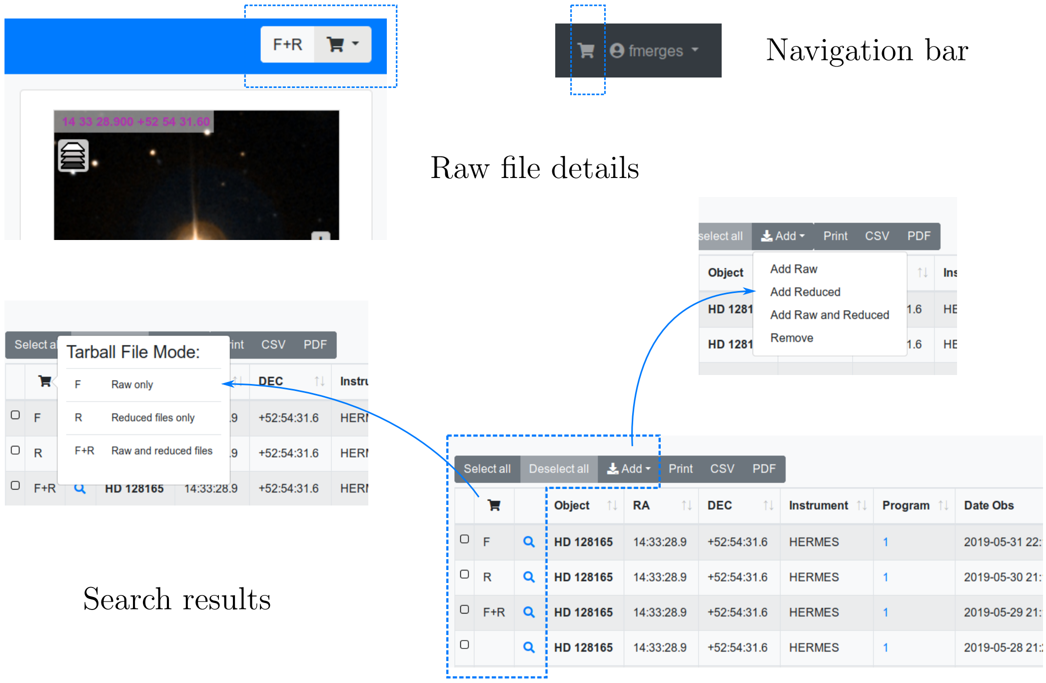


Figure 6.7: Sprint Three: Data download design elements

the task is processed. The task queue identifies each task with an unique identifier. The identifier can be used to query the status of the task, show its progress, and relaunch in case of problems. Figure 6.10 despicts the tarball creation.

The stored tarballs may take up some considerable space. The development of a tool is planned to control the disk quota, wich can be run periodically using *cron*.

A tarball goes through several states as seen in Figure 6.9. Each of these states are recorded with the time the state transitioned. And, a user can have many closed tarballs but only one open at a time. Figure 6.11 shows the tarball page, in which the contents of the tarball are shown with the download mode, the status of the tarball, the total number of files it contains and the sum of the files. It shows 6 files in the tarball because each reduced set, for the given test-dataset, is made up of two files: $2Raw + 2Reduced \cdot 2 = 6Files$

	Add File to Tarball
Goal Level:	Sea Level
Primary Actor:	User
Preconditions:	User is logged in

Main Success Scenario:

1. The user selects the raw file(s), and mode that identifies the set, to download.
2. The system get the files taking the user's permissions into account.
3. The system adds to the user's active tarball the resulting files from the previous step with the given mode.
4. The system notifies the user about the changes.

Figure 6.8: Sprint Three: Add file to tarball use case

Figure 6.9: Sprint Three: Tarball status

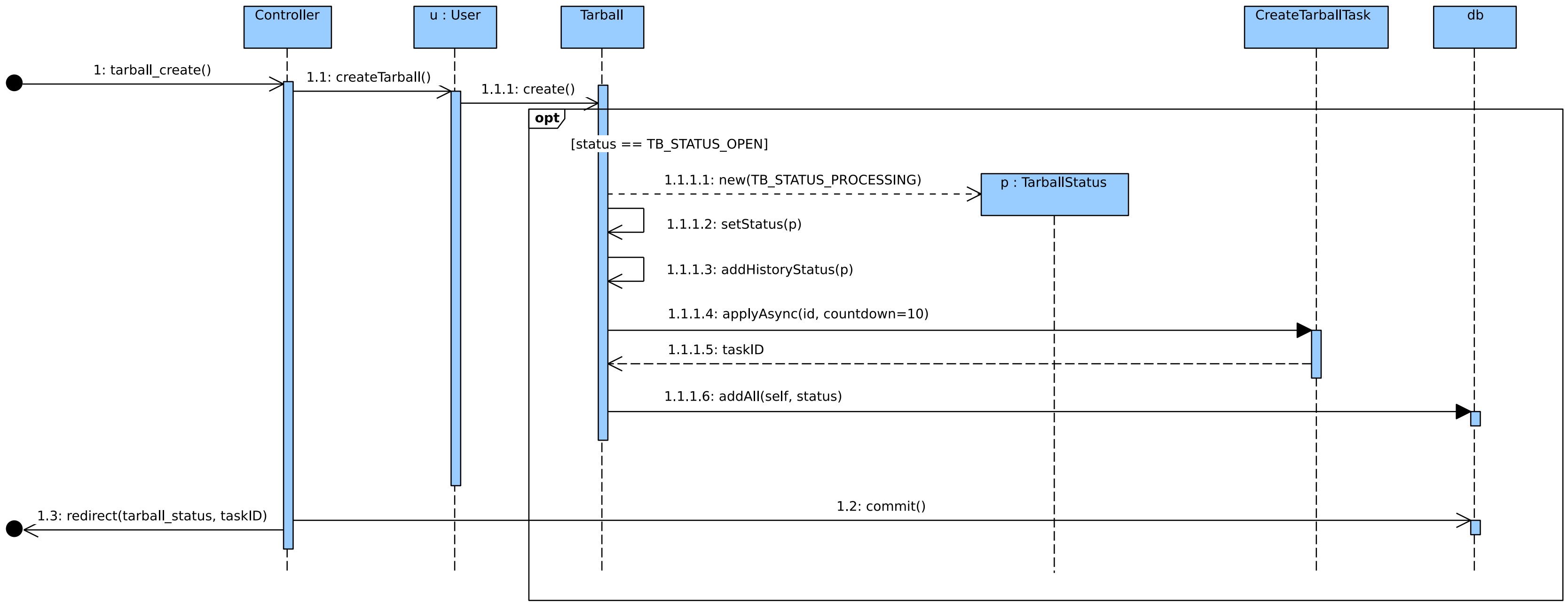


Figure 6.10: Sprint Three: Tarball creation sequence diagram

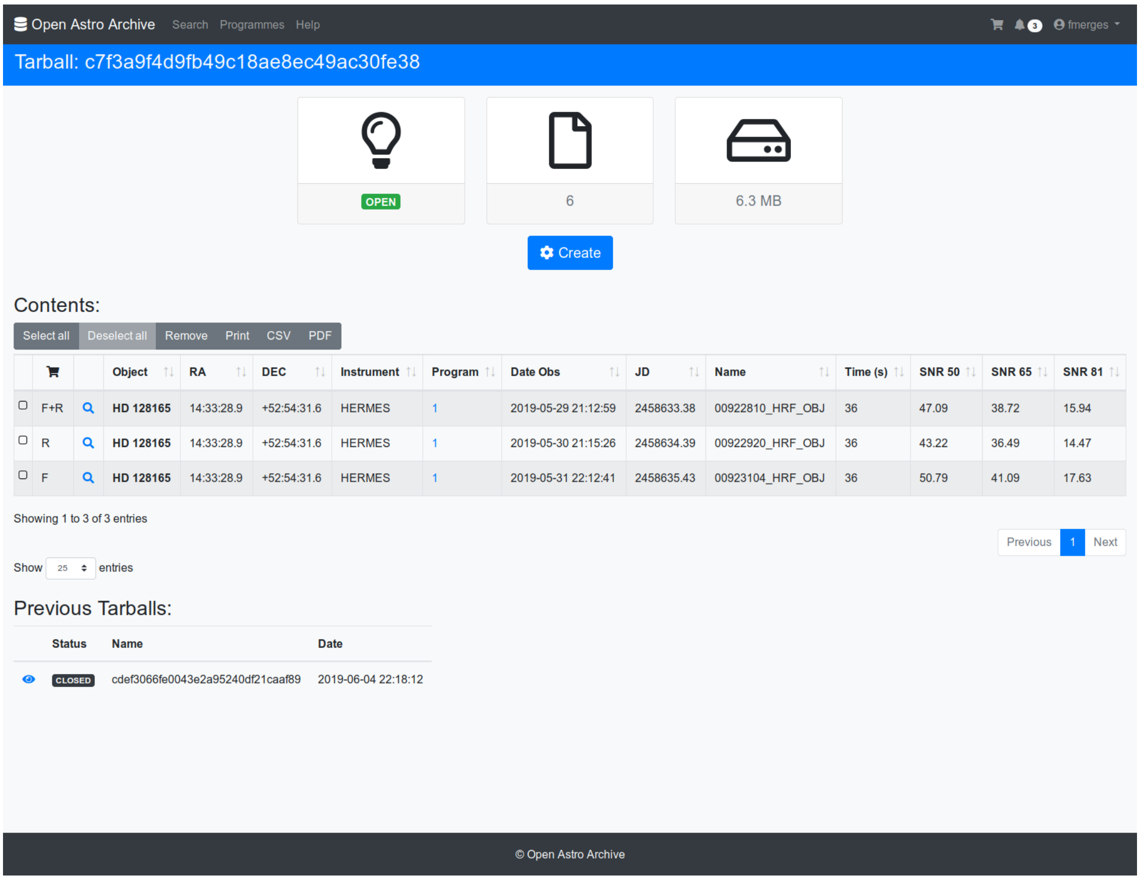


Figure 6.11: Sprint Three: Tarball detail view screenshot

6.4 User Management

The *Flask-Security* extension provides a datastore that abstracts users management. But, it only knows about users and roles, not tarballs. Consequently, since the introduction of the tarball epic, the user management stopped working. To fix it, a new datastore was created which relies on the application model instead, thus handling correctly all the tarball intricacies.

6.5 Domain Model

Several changes to the domain model were required:

- a new `Attachment` class is associated to `Program` class
- three new classes were added for the data download and tarball generation: `Tarball`, `TarballFile`, and `TarballStatus`. Both `Tarball` and `TarballStatus` implement the Historic Mapping association pattern.[45, Ch. 15]
- new relationships are created with the existing `User` class

Figure 6.12 shows an excerpt of the domain model class diagram.

6.6 Retrospective

Program and download stories are implemented. Pending for a future sprint: add tarball compression option to the tarball view; at this moment it is part of the application settings.

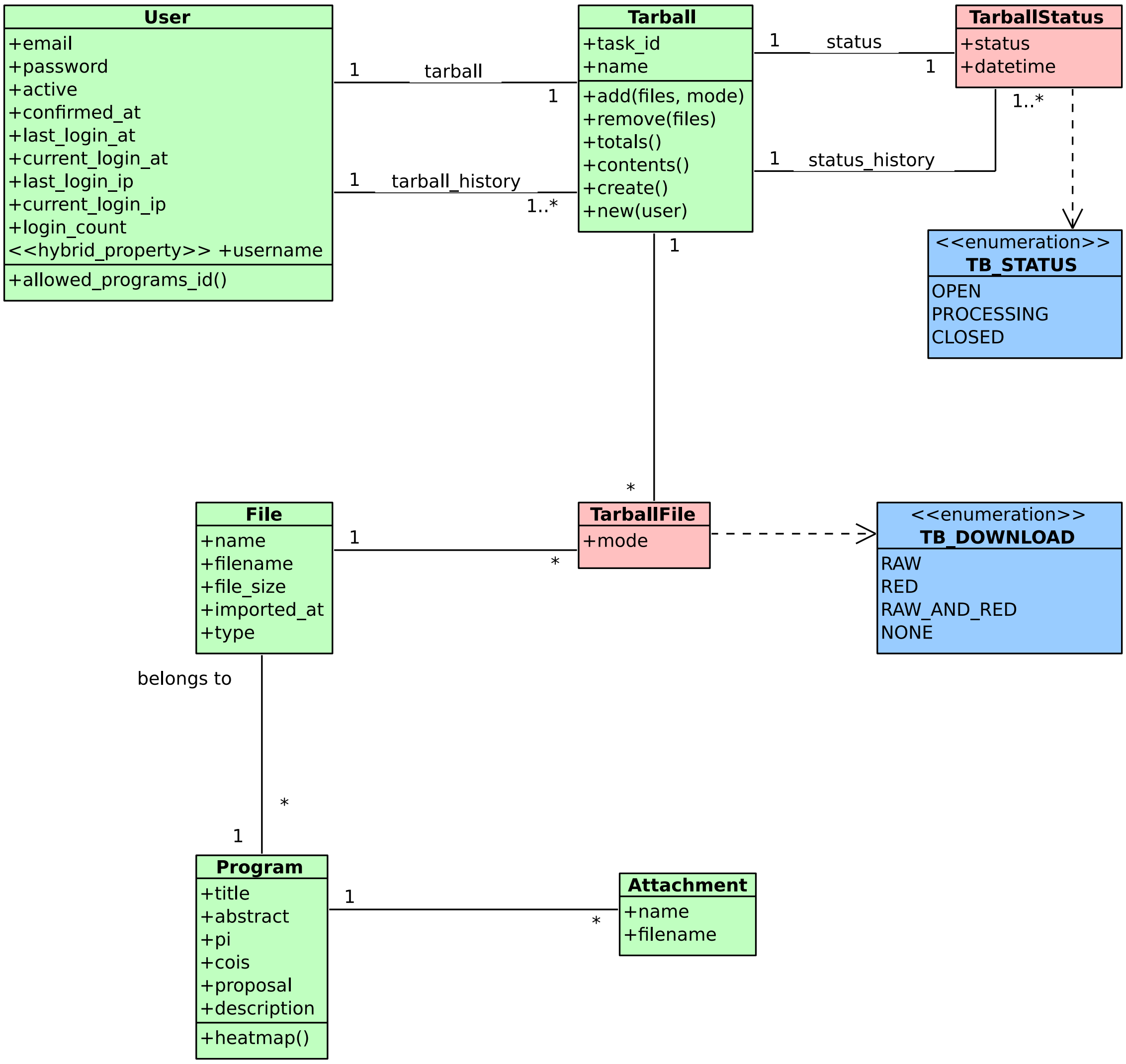


Figure 6.12: Sprint Three: Domain model, tarball and attachment class diagram

Chapter 7

Sprint Four

7.1 User Stories

Tarball notification	
As a User, I want to receive a notification when a tarball is ready for download	
Only receive notification if subscribed	

New data notification	
As a User, I want to receive a notification when new data is available for a program	
Only receive notification for subscribed programs	

Manage tarball notification subscription	
As a User, I want to be able to manage my tarball notification subscription	
Only receive notification if subscribed	

Manage data notification subscriptions	
As a User, I want to be able to manage my notification subscriptions	
Should only allow to programs for which the user has permissions Can only select between internal, or internal and e-mail	

Internationalization	C
As a User, I want to choose the user interface language	
User can change profile settings interface language changes	

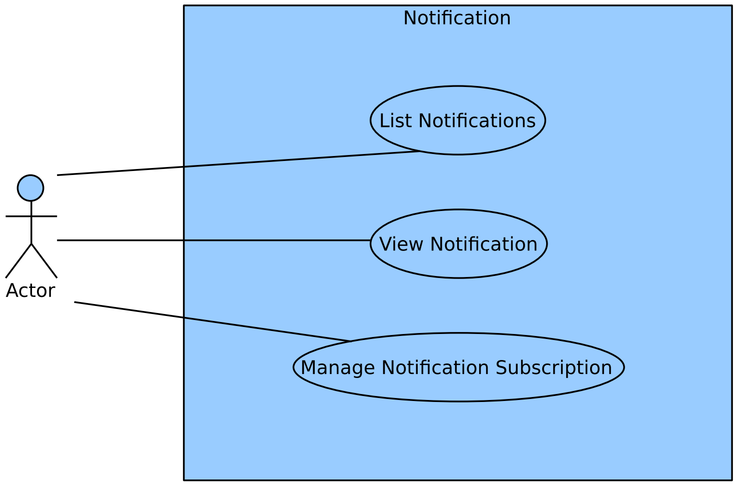


Figure 7.1: Sprint Four: Notifications use case diagram

7.2 Notifications

Early on, it was already apparent that a general notification system for the application was certainly very convenient. Initially, two cases were identified: a notification when a tarball is ready, and a notification when new data for a program is available. Soon after, new notification types were identified, like for instance, a notification when program's permissions were modified. Figure 7.1 shows a use case diagram with the actions a user can do.

Notification saves the user from waiting that a certain event happens, or even checking if it did so. Notifications are handy for long running actions and asynchronous ones. Checking every day if there is new data available for a program is certainly not appealing. By using a notification system, the user only needs to wait for the notification to arrive.

The notification system does not rely on any external service, e.g., Google Cloud Messaging (GCM). Furthermore, users can manage their own subscriptions, by selecting for each notification type between internal notifications only, or both internal and e-mail. Figure 7.2 gives a short summary of the written specification for the *Manage Notification* use case. Figure 7.3 and 7.4 show details of the user interface of the notification system.

In essence, a notification consists of a message that is addressed to a user. By and large, a notification is rarely personalized to the extend to be only of use to a single user. As such, a notification may consist of a template that is filled on demand. Thus, the only thing required to be tracked is if the user opened the notification or not, and when. In consequence the notifications in the system are shared among users, thus avoiding a lot of redundant data.

There may be several strategies to trigger a notification. In the case of a new tarball one, the natural choice is to assign the responsibility to the worker that creates the tarball. However, in the case of a program notification, the responsibility might be assigned to the importing tool, or to a utility that checks if there is new data since the last time it was executed—as it turns out, the later solution is the more robust one, specially when several observing nights are imported at once, otherwise duplicate notifications can be triggered.

A service module was created, following the Pure Fabrication[46] or Service pattern, to handle all the notification actions. The service is used by the controller, the workers, and the utility tools. Figure 7.5 shows a class diagram of it. Figure 7.6 shows a sequence diagram of the actions done by the program data notification handler, triggered by the command line utility.

	Manage Notification
Goal Level:	Sea Level
Primary Actor:	User
Preconditions:	User is logged in

Main Success Scenario:

1. The user accesses the notification subscription.
2. The system returns a form prefilled with the notification subscriptions.
3. The user changes the subscription settings and submits.
4. The system updates the notification subscriptions.
5. The system notifies the user about the changes.

Figure 7.2: Sprint Four: Manage notification subscriptions use case

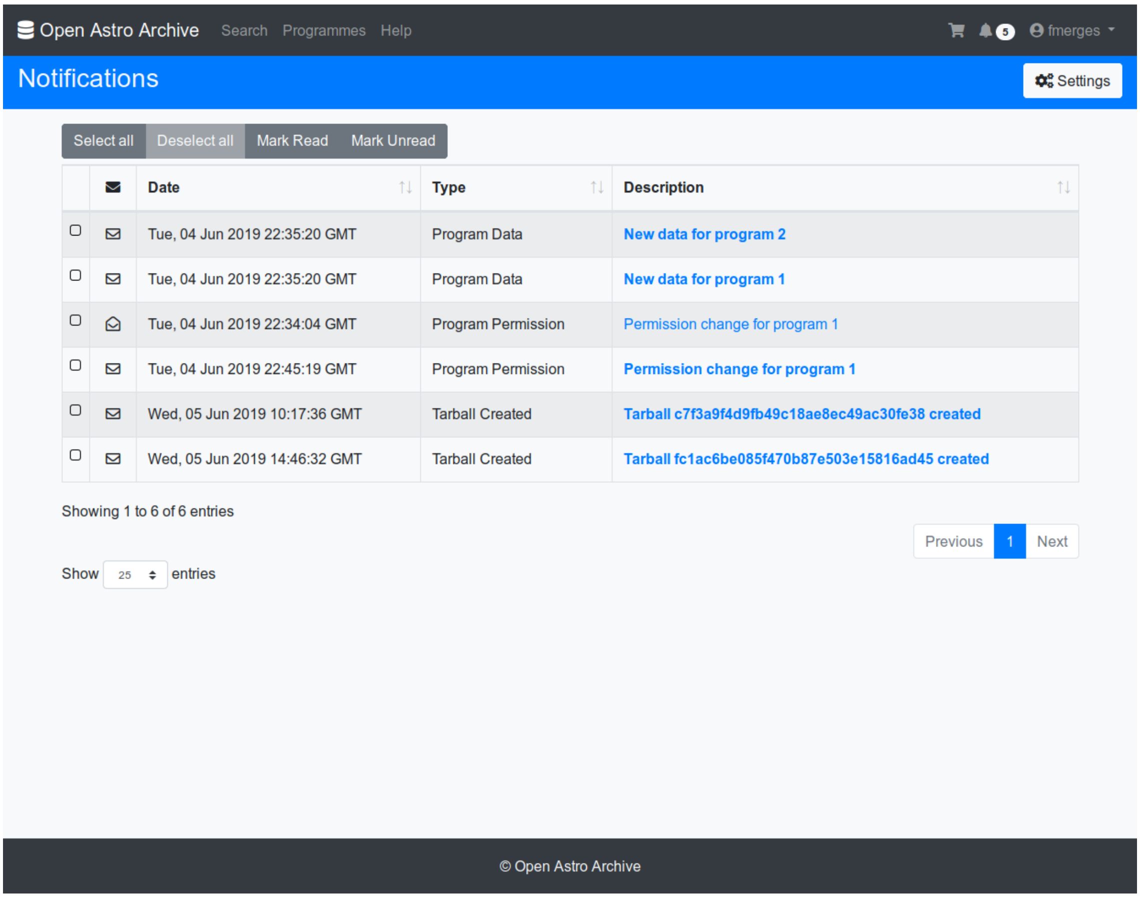


Figure 7.3: Sprint Four: Notifications display screenshot

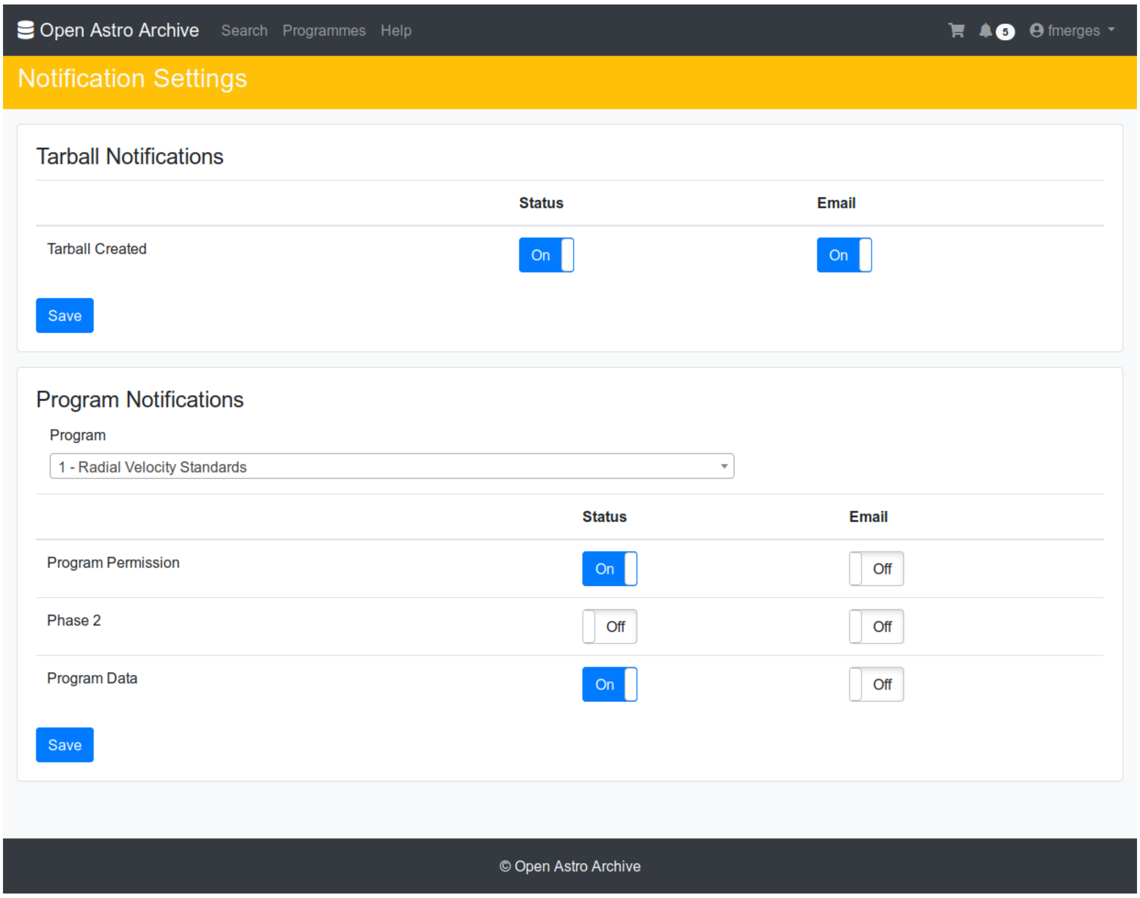


Figure 7.4: Sprint Four: Notifications settings screenshot

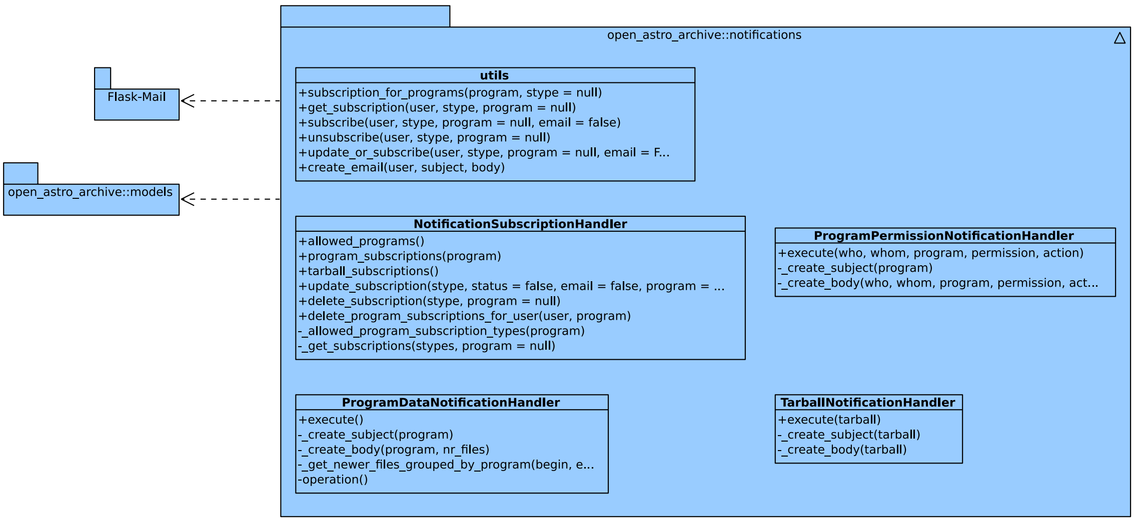


Figure 7.5: Sprint Four: Notifications service class diagram

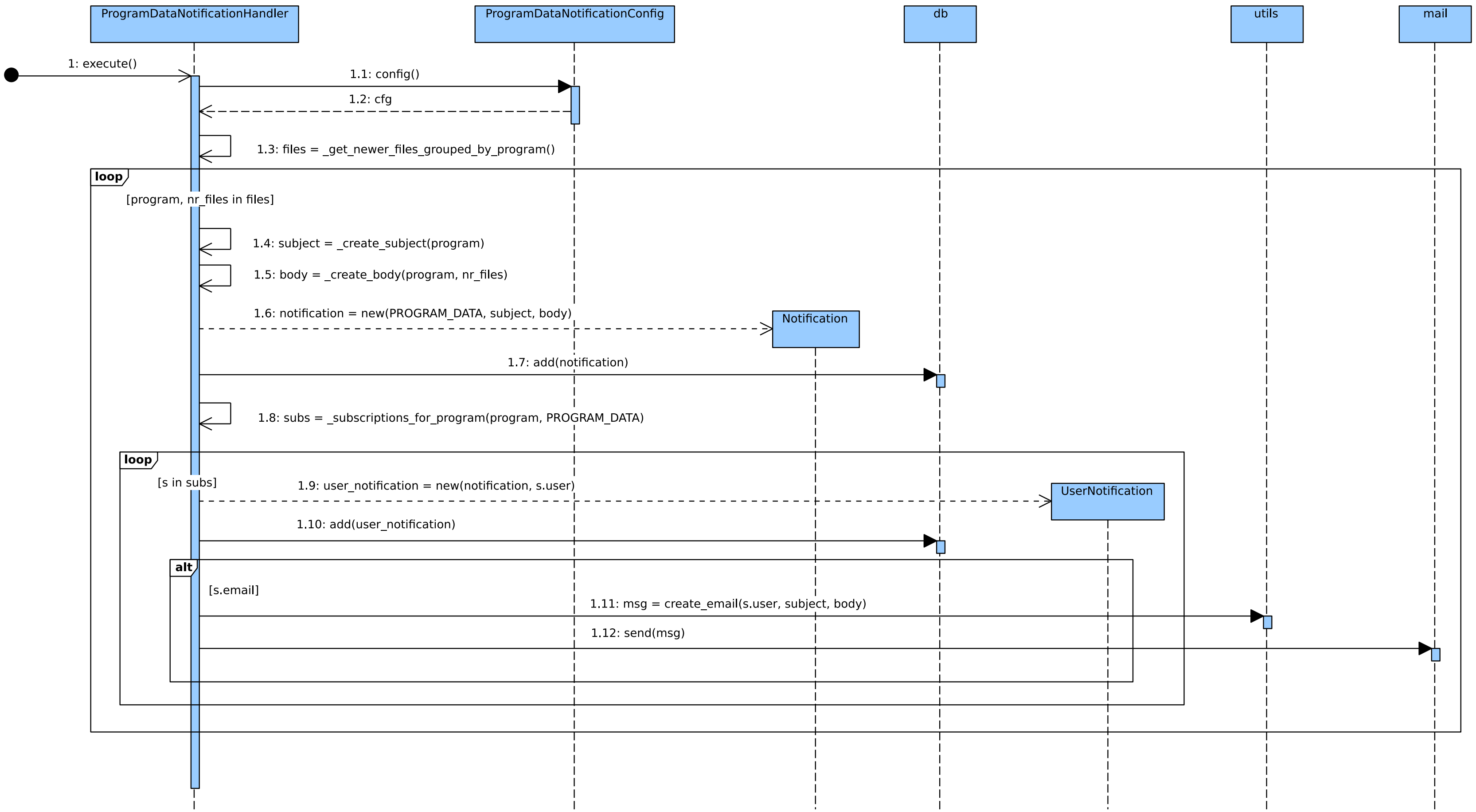


Figure 7.6: Sprint Four: Program data notification handler sequence diagram

7.3 Domain Model

Once more, new classes were added to the domain model:

- `Notification`
- `NotificationSubscription`
- `UserNotification`
- `ProgramDataNotificationConfig`

Figure 7.7 shows an excerpt of the domain model class diagram.

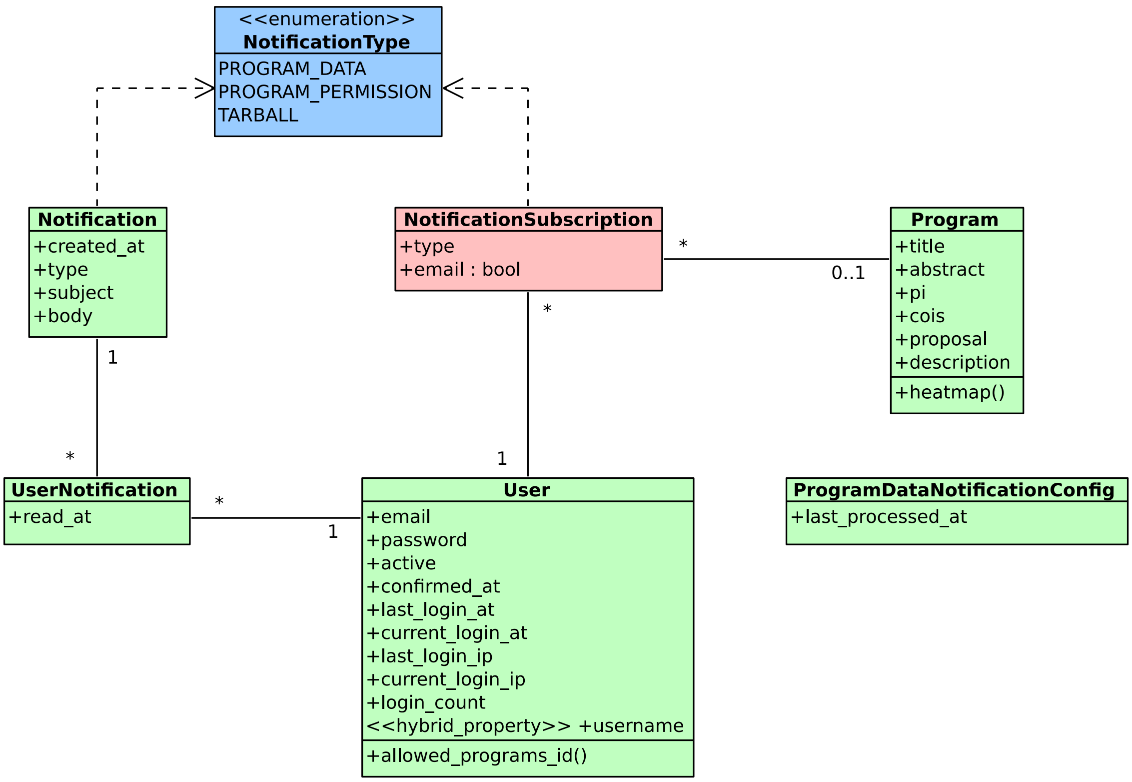


Figure 7.7: Sprint Four: Domain model, notifications class diagram

7.4 Retrospective

Notification user stories were implemented. The user can subscribe to different types of notifications and choose between internal delivery, or e-mail and internal delivery.

Due to time constraints, the user story regarding internationalization could not be completed. Albeit, the foundation is in place for an upcoming sprint.

Chapter 8

Conclusions

Software engineering demands continuous learning and practice, lots of practice indeed. Learning the theory is not enough, it is the practice that makes the skill. And one of the most important ones—may the author be granted the right to introduce an opinion—is our ability to recognize patterns. Not patterns in a purely software sense, but in a greater one. The "Gang of Four"[37], inspired by the magnificent book *A Pattern Language. Towns. Buildings. Construction*[16], made the term *pattern* popular in software.

But somehow, the essential idea got lost: recognizing patterns and the use of a language to describe them. People at first, including the author, started blindly to apply these so called *software patterns* having the wrong idea that, by doing so, the end product would be better or in business terms, more enterprise. But with practice, you come to realize that the essential aspect is to recognize them, and moreover to discern when it is a good idea to apply the proposed solution or not.

When confronted to a problem that demands a software solution, the first step is to gain context, analyze and understand the problem in hands, and be able to name it. Gaining context in many fields provide perspective. And it is through perspective that our abilities to recognize patterns are enhanced. The lessons learned from this project strengthen this idea, as well as the recognition of the vast amount of knowledge waiting ahead.

The main objectives proposed for this project were accomplished. There is a fully working software that complies with the requirements initially laid out, see Section 1.5. (With one exception, a low priority requirement, half way implemented, that had to be postponed for later: internationalization.)

The methodology used, although not in its full extend, is the reason that this final product is available. The classic waterfall approach would have certainly provided more formal documentation, but given the time frame, little or no software at all would have been available. In addition, it is the author's believe, that if a standard project management approach for software development would have been strictly followed, even an agile one, it would have had a detrimental effect, given the time frame, on the final outcome.

There has been many stories and topics that were not covered during this project. It is a work in progress, and it is the author's plan to continue its development—embracing whatever it brings along. Following is a short list of some of those stories:

- add more unit tests.
- add more command line tools.
- add support for more telescopes and instruments.
- finish internationalization.
- create an administration web user interface.
- package the application.
- abstract the telescope support further. Importer, plot, and other services as extensions of the framework.
- use *Docker* to containerize the application, thus allowing easier deployment.
- make use of *marshmallow*, a simplified object serialization, and share a common service for the web view and the REST-API.
- develop a REST-API, and a module for *astroquery*, thus allowing third parties to access the system.
- create a dashboard for users and administrators.
- use *cookiecutter*. Once setup, the system can be easily customized.
- provide support for different user interface themes. Add branding features.
- add Virtual Observatory (VO) support.
- add support for *Jupyter Notebooks*.
- consider to add a functionality to support a Phase II process.
- eventually add *S3* support.

In the end, this project was a very enriching experience. It is with surprise just how much knowledge was acquired for the production of this document.

Bibliography

[1] Merrian Webster. *Definition of Charged-Coupled-Device*. 2019. URL: https://www.merriam-webster.com/dictionary/charge-coupled%5C%20device (visited on 06/10/2019).

[2] D. Scott Birney, Guillermo Gonzalez, and David Oesper. *Observational Astronomy. Second Edition*. Cambridge University Press, 2006.

[3] FITS Working Group. *Definition of the Flexible Image Transport System*. 2016. URL: https://fits.gsfc.nasa.gov/standard40/fits_standard40aa-le.pdf.

[4] The HDF Group. *HDF5*. 2019. URL: https://portal.hdfgroup.org/display/HDF5/HDF5 (visited on 06/10/2019).

[5] International Virtual Observatory Alliance. *IVOA.net*. 2002. URL: http://ivoa.net/ (visited on 06/10/2019).

[6] Kenneth S. Rubin. *Essential Scrum*. Pearson Education, Inc., 2013.

[7] G. Booch et al. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, 2007.

[8] Princeton University "About WordNet." *WordNet*. URL: https://wordnet.princeton.edu/ (visited on 06/10/2019).

[9] Mike Cohn. *User Stories Applied: For Agile Software Development*. Addison-Wesley, 2004.

[10] Mike Cohn. *A Sample Format for a Spreadsheet-Based Product Backlog*. 2011. URL: https://www.mountaingoatsoftware.com/blog/a-sample-format-for-a-spreadsheet-based-product-backlog (visited on 06/10/2019).

[11] Mike Cohn. *Non-functional Requirements as User Stories*. 2008. URL: https://www.mountaingoatsoftware.com/blog/non-functional-requirements-as-user-stories (visited on 06/10/2019).

[12] Henrik Kniberg. *Scrum and XP from the Trenches: How we do Scrum*. C4Media Inc, 2007.

[13] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

[14] The Linux Information Project. *Tarball Definition*. 2005. URL: http://www.linfo.org/tarball.html (visited on 06/10/2019).

[15] Mark Richards. *Software Architecture Patterns*. O'Reilly, 2015.

[16] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. *A Pattern Language. Towns. Buildings. Construction*. Oxford Univeristy Press, 1977.

[17] WardCunningham et al. *Whatsa Controller Anyway*. 2013. URL: http://wiki.c2.com/?WhatsaControllerAnyway (visited on 06/10/2019).

[18] Django Software Foundation. *FAQ: General*. 2013. URL: https://docs.djangoproject.com/en/dev/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names (visited on 06/10/2019).

[19] P. Christensson. *Cloud Computing Definition*. 2009. URL: https://techterms.com/definition/cloud_computing (visited on 06/10/2019).

[20] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill College, 1997.

[21] P. Christensson. *HTTP Definition*. 2015. URL: https://techterms.com/definition/http (visited on 06/10/2019).

[22] Albert R. Conrad. *Software Systems for Astronomy*. Springer, 2014.

[23] Michael Bayer. "SQLAlchemy". In: *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks*. Ed. by Amy Brown and Greg Wilson. aosabook.org, 2012. URL: http://aosabook.org/en/sqlalchemy.html.

[24] Open Web Application Security Project. *SQL Injection*. 2016. URL: https://www.owasp.org/index.php?title=SQL_Injection&oldid=212863 (visited on 06/10/2019).

[25] S. Koposov and O. Bartunov. "Q3C, Quad Tree Cube – The new Sky-indexing Concept for Huge Astronomical Catalogues and its Realization for Main Astronomical Queries (Cone Search and Xmatch) in Open Source Database PostgreSQL". In: *Astronomical Data Analysis Software and Systems XV*. Ed. by C. Gabriel et al. Vol. 351. Astronomical Society of the Pacific Conference Series. July 2006, p. 735.

[26] Kenneth Reitz and Tanya Schlusser. *The Hitchhiker's Guide to Python*. O'Reilly, 2016.

[27] Pallets Team. *Larger Applications*. 2010. URL: http://flask.pocoo.org/docs/1.0/patterns/packages/ (visited on 06/10/2019).

[28] Pallets Team. *Modular Applications with Blueprints*. 2010. URL: http://flask.pocoo.org/docs/1.0/blueprints/#modular-applications-with-blueprints (visited on 06/10/2019).

[29] P. Christensson. *Daemon Definition*. 2006. URL: https://techterms.com/definition/daemon (visited on 06/10/2019).

[30] Eric S. Raymond. *The Art of Unix Programming*. 2003.

[31] Brian W. Kernighan and Rob Pike. *The Practice of Programming*. Addison-Wesley, 1999.

[32] Jenifer Tidwell. *Designing Interfaces*. O'Reilly, 2005.

[33] Don Norman. *The Design of Everyday Things*. Basic Books, 2013.

[34] Len Silverston and Paul Agnew. *The Data Model Resource Book. Volume 3*. Wiley, 2009.

[35] Peter Coad, Jeff de Luca, and Eric Lefebvre. *Java Modeling in Color With UML*. Prentice Hall, 1999.

[36] Matt Wright. *Flask-Security*. 2012. URL: https://pythonhosted.org/Flask-Security/ (visited on 06/10/2019).

[37] E. Gamma et al. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[38] Luciano Ramalho. *Fluent Python: Clear, Concise, and Effective Programming*. O'Reilly, 2015.

[39] Mark Summerfield. *Python in Practice*. Addison-Wesley, 2014.

[40] M. Wenger et al. "The SIMBAD astronomical database. The CDS reference database for astronomical objects". In: *Astronomy and Astrophysics, Supplement* 143 (Apr. 2000), pp. 9–22. DOI: 10.1051/aas:2000332. eprint: astro-ph/0002110.

[41] Naval Meteorology and Oceanography Command. *Catalog Cone Search*. 2008. URL: https://www.usno.navy.mil/USNO/astrometry/optical-IR-prod/icas/vo_nofs (visited on 06/10/2019).

[42] Semmy Purewal. *Learning Web App Development*. O'Reilly, 2014.

[43] Anders Toxboe. *UI-Patterns*. 2007. URL: http://ui-patterns.com/ (visited on 06/10/2019).

[44] Steve Krug. *Don't make me think, Revisited, 3rd Edition*. New Riders, 2014.

[45] Martin Fowler. *Analysis Patterns: Reusable Object Models*. Addison-Wesley, 1996.

[46] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition*. Addison-Wesley, 2004.

[47] Martin Fowler. *UML Distilled. 3rd Edition*. Addison-Wesley, 2004.

[48] Alistair Cockburn. *Agile Software Development*. Pearson Education, Inc., 2002.

[49] Leslie Lamport. *Latex. A Document Preparation System*. Addison-Wesley, 1994.

[50] William Strunk Jr. and E. B. White. *The Elements of Style*. Pearson Education, Inc., 2000.

[51] Douglas Crockford. *JavaScript: The Good Parts*. O'Reilly, 2008.

[52] Heather Silyn-Roberts. *Writing for Science and Engineering. 2nd Edition*. Elsevier, 2013.

[53] chromatic. *Extreme Programming Pocket Guide*. O'Reilly, 2003.

[54] Miguel Grinberg. *Flask Web Development*. O'Reilly, 2014.

[55] Scott Chacon. *Pro Git*. Apress, 2009.

[56] Jeff Sutherland. *The Art of Doing Twice the Work in Half the Time*. Crown Business, 2014.

[57] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.

[58] Astropy Collaboration et al. "Astropy: A community Python package for astronomy". In: *Astronomy and Astrophysics* 558, A33 (Oct. 2013), A33. DOI: 10.1051/0004-6361/201322068. arXiv: 1307.6212 [astro-ph.IM].

[59] A. M. Price-Whelan et al. "The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package". In: *Astronomical Journal* 156, 123 (Sept. 2018), p. 123. DOI: 10.3847/1538-3881/aabc4f.

[60] F. Merges et al. "MESA: Mercator scheduler and archive system". In: *Software and Cyberinfrastructure for Astronomy II*. Vol. 8451. Proceedings of the SPIE. Sept. 2012, p. 84512C. DOI: 10.1117/12.926623.

Appendix A

Source Code

A.1 Data Import Tool

The data import tool is called from the command line. Calling it with the option -h displays the help for the command.

Listing A.1: Source Code: Data import tool source code

```
@app.cli.command()
@click.argument('dirs', nargs=-1, type=click.Path(dir_okay=True))
@click.option('--telescope', default='mercator',
              help='Source of the images. Default mercator')
@click.option('--reduced', is_flag=True, default=False,
              help='Raw or reduced files. Default raw')
def import_files(dirs, telescope, reduced):
"""Import files into the archive"""
imp_type = 'reduced' if reduced else 'raw'
click.echo(click.style(
   "Importing %s %s data" % (telescope, imp_type), fg='green'))
imp = importer.getcls(telescope, imp_type)()

for d in dirs:
    click.echo("Processing directory %s:" % d)
    files = glob.glob(os.path.join(d, "*.fits"))
    with click.progressbar(files) as bar:
        for f in bar:
            try:
                imp.run(f)
                db.session.commit()
            except:
                click.echo(click.style("\nProblem processing file %s" %
                           click.format_filename(f), fg='red'))
                raise
```

A.2 Controller Permission Checks

Permission requirements for a controller function can be ensured by several ways: one by decorating[37] the function, another, by checking the permissions from inside the function itself. The later is used for an unknown compile time argument, e.g., a program related action. Listing A.2 shows how both can be combined in code, in this case, checking the user is logged in and at least satisfying the *Program User* role for the given program.

Listing A.2: Source Code: Controller function permissions check

```
@bp.route('/<int:id>/')
@login_required          # check user is logged in
def program(id):
    program = Program.query.filter_by(id=id).first_or_404()

    # check permissions
    can_view_program_or_abort(program.id)

    is_admin = is_program_admin(program.id)

    return render_template('program/view.html', program=program,
                           is_admin=is_admin)
```

Appendix B

Deployment Instructions

B.1 Production Environment

```
INTRODUCTION
============

The following instructions cover the deployment of the open astro archive
system to a server.

PREREQUISITES
=============

Make sure you have installed the following software on your server:

* python > 3
* postgresql >= 9.1
* redis > 2.2
* pip >= 1.0

DEBIAN/UBUNTU PREREQUISITES INSTALLATION
========================================

Install required packages:

$ sudo su
# apt-get install build-essential python3
# apt-get install ngix git ssh
# apt-get install redis-server postgresql
# apt-get install libssl-dev python3-dev postgresql-server-dev-all

INSTALLATION
============

Install virtualenv:

# pip install --upgrade pip
# pip install virtualenv

Make sure postgresql accepts local connection using password.
The pg_hba.conf file should have a line like:

host    all             all             127.0.0.1/32            md5

Create database users:

$ su - postgres
$ createuser -P oaa
$ createdb -O oaa open_astro_archive
```

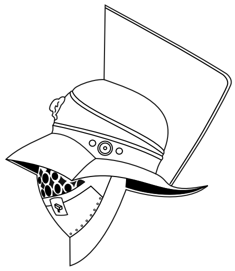
```
Install q3c extension:

$ git clone https://github.com/segasai/q3c
$ cd q3c
$ make
$ make install
$ psql -c "create extension q3c" open_astro_archive oaa
$ exit

Create system user:

$ sudo adduser --gid 100 oaa
$ sudo mkdir -p /srv/www
$ sudo chown oaa.users /srv/www

Create a bare repository to which you are going to sync to:

$ su - oaa
$ cd /srv/www
$ git clone git@github.com:fmerges/open-astro-archive.git
$ cd open-astro-archive
$ virtualenv venv
$ source venv/bin/activate
$ pip install -r requirements.txt

Create instance folder including uploads and logs subfolder:

$ cd /srv/www/open_astro_archive
$ mkdir -p instance/log
$ mkdir -p instance/uploads
$ cd instance

Optionally, create instance settings to override default ones:

$ cat > settings.cfg

Create supervisord configuration file.
Hit CTRL-D for finishing input and to save the file:

$ cat > supervisord.conf

[supervisorctl]
serverurl=unix:///tmp/supervisor.sock ; use a unix:// URL  for a unix socket

[program:uwsgi]
directory=/srv/www/open_astro_archive
environment=TZ="UTC"
command=/srv/www/open_astro_archive/venv/bin/uwsgi %(here)s/uwsgi-open_astro_archive.ini
autostart=true
autorestart=true
stdout_logfile=%(here)s/log/%(program_name)s.log
redirect_stderr=true
exitcodes=0

And uwsgi configuration file:

$ cat > uwsgi-open_astro_archive.ini

[uwsgi]
master = 1
processes = 2
threads = 2
;socket = /tmp/%n.sock
socket = 127.0.0.1:3031
wsgi-file = /srv/www/open_astro_archive/wsgi.py
callable = application
logdate = true
;virtualenv = /srv/www/open_astro_archive/venv
pidfile = /srv/www/open_astro_archive/instance/web.pid
stats = 127.0.0.1:9191
```

```
Create database tables:

$ cd /srv/www/open_astro_archive
$ flask createdb

Start the application:

$ cd /srv/www/open_astro_archive
$ supervisord

Setup nginx webserver:

# cat > /etc/nginx/sites-available/open_astro_archive

server {
    listen        80;
    listen        443 ssl;
    server_name   161.72.58.13;
    keepalive_timeout    70;

    ssl_certificate /etc/ssl/certs/nginx-selfsigned.crt;
    ssl_certificate_key /etc/ssl/private/nginx-selfsigned.key;
    ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers         HIGH:!aNULL:!MD5;

    charset    utf-8;
    client_max_body_size 75M;

    location / { try_files $uri @open_astro_archive; }

    location @open_astro_archive {
        include uwsgi_params;
        uwsgi_pass 127.0.0.1:3031;
    }

    location /static {
        root /srv/www/open_astro_archive/open_astro_archive/;
    }
}

# ln -s /etc/nginx/sites-available/open_astro_archive \
/etc/nginx/sites-enabled/open_astro_archive

Restart nginx webserver:

# systemctl restart nginx.service
```

B.2 Development Environment

```
INTRODUCTION
============

The following instructions cover the deployment of the archive
system to a development environment.

The idea is to have a local copy of the source code that can be pushed
to the server. Therefore, clone the repository to a local folder on your
system:

$ git clone git@github.com:fmerges/open-astro-archive.git

PREREQUISITES
=============

Make sure you have installed the following software on your server:

* python > 3
* postgresql >= 9.1
* redis > 2.2
* pip >= 1.0

DEBIAN/UBUNTU PREREQUISITES INSTALLATION
========================================

Install required packages:

$ sudo su
# apt-get install build-essential python3
# apt-get install ngix git ssh
# apt-get install redis-server postgresql
# apt-get install libssl-dev python3-dev postgresql-server-dev-all

INSTALLATION
============

Install virtualenv:

# pip install --upgrade pip
# pip install virtualenv

Create database users:

$ su - postgres
$ createuser -P oaa
$ createdb -O oaa open_astro_archive

Install q3c extension:

$ git clone https://github.com/segasai/q3c
$ cd q3c
$ make
$ make install
$ psql -c "create extension q3c" open_astro_archive oaa
$ exit

Create system user:

$ sudo adduser --gid 100 oaa
$ sudo mkdir -p /srv/www/open_astro_archive
$ sudo chown oaa.users /srv/www/open_astro_archive
```

```
Create a bare repository to which you are going to sync to:

$ su - oaa
$ mkdir open_astro_archive.git && cd open_astro_archive.git
$ git init --bare
$ cd hooks

Create post receive hook for checking out the code from the repo
Hit CTRL-D for finishing input and to save the file.

$ cat > post-receive

#!/bin/sh
git --work-tree=/srv/www/open_astro_archive \
    --git-dir=/home/oaa/open_astro_archive.git checkout -f

$ chmod +x post-receive

On your local system, add your live system as a remote in order
to automatically deploy to it, so from your local machine inside
the repository do:

$ git remote add live oaa@hostname:~/open_astro_archive.git

In order to deploy the code to the server invoke:

$ git push live origin/master:master

On your server:

$ cd /srv/www/open_astro_archive
$ virtualenv venv
$ source venv/bin/activate
$ pip install -r requirements.txt

Create instance folder, attachment upload directory and logs:

$ cd /srv/www/open_astro_archive
$ mkdir -p instance/log
$ mkdir -p instance/uploads
$ cd instance

Optionally, create instance settings to override the default ones:

$ cat > settings.cfg

Create database tables:

$ cd ..
$ flask createdb

Run the application:

$ flask run
```



PER ASPERA AD ASTRA