Masaryk University
Faculty of Informatics

Web application for searching for astronomical objects in internet catalogues

Bachelor's Thesis

Ľuboslav Halama

Brno, Spring 2021

Masaryk University
Faculty of Informatics

Web application for searching for astronomical objects in internet catalogues

Bachelor's Thesis

Ľuboslav Halama

Brno, Spring 2021

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Ľuboslav Halama

Advisor: RNDr. Martin Kuba, Ph.D.

Acknowledgements

I would like to thank my advisor RNDr. Martin Kuba, Ph.D. for his advice and supervision, and my consultant Ernst Paunzen, Dr.rer.nat, for countless consultations and his valuable advice, help, willingness, and patience.

Last but not least, I would like to thank my family and friends for unwavering support and encouragement during my studies.

Abstract

Astronomical databases represent a significant and extensive source of various information about a wide range of astronomical objects. This thesis describes three of the largest astronomical databases and catalogues and presents a web application for searching for astronomical objects in them - AstroSearcher. The main part of the application is written in Java with a use of HTML, CSS, and Javascript for a view part. The work describes analysis, design and implementation of the web application.

Keywords

Astronomy, MAST, Simbad, Vizier, Java, Spring, HTML, Thymeleaf, CSS

Contents

List of Tables

List of Figures

Introduction

Astronomical databases and catalogues represent a significant and extensive source of various information about a wide range of objects in our solar system, as well as objects far beyond the system of the Sun. As there are millions of celestial objects and measurements connected to them with a constant flow of new data, the natural need for categorizing and systematic naming of those objects, also known as the nomenclature of celestial objects, has risen. The nomenclature, together with celestial coordinates, provided in several interchangeable variants as galactic coordinates or ICRS coordinates[1], creates a robust system capable of handling both simple and complex queries for obtaining astronomical data.

My goal is to implement a web application for astronomers that provides mostly tabular results from 3 different databases – MAST, Simbad, and Vizier – in a simple and transparent way. Among a larger amount of query parameters, only identifiers, a pair of coordinates (ICRS coordinates in the form of decimal degrees), and a file with pairs of coordinates were used as primary query parameters.

The thesis is divided into six chapters. In chapter 1, I introduce the three mentioned databases. In chapter 2, I list and analyse functional and non-functional requirements that my application must meet. Chapter 3 describes how responses from each of the databases can be obtained and which API[2] functions for the corresponding database are used for queries in my web application. In chapter 4, I describe the design of the application, defines tools and methods used for implementation. In chapter 5, I describe the implementation of the web application, encountered problems, and their solutions. In chapter 6, I present possible future improvements in the application.

The result of my thesis is a working astronomical web application for querying astronomical databases – AstroSearcher.

1. International Celestial Reference System
2. Application Programming Interface

1 Available searching tools

ICRS coordinates are represented as a pair of coordinates, right ascension and declination, whose meaning as the celestial coordinates can be compared to longitude and latitude on the surface of Earth.

Right ascension is the east-west coordinate, the angular distance of a body's hour circle east of the vernal equinox, measured along the celestial equator. It is often expressed in units of time rather than degrees of arc [1].

Declination is the angular distance of a body north or south of the celestial equator, where +90° declination marks the north celestial pole, and -90° the south celestial pole [2].

However, to query astronomical databases, especially with the use of coordinates, one more piece of information is needed – radius, measured in degrees, arcminutes, or arcseconds. Radius describes the size of the area around a point defined by a given pair of coordinates, thus radius directly affects the size of a result set.

My primary aim for web application was to provide uniform, simple and quick access to data from all the mentioned databases at once, without the need for typing queries on multiple websites. In terms of a suitable format for obtaining the results, web portals could not be used to query already mentioned databases and services, since they provide information in the form of an HTML response, which is not the optimal format for parsing, because it contains a high amount of web design blocks besides the requested data. From the remaining two standard variants, I have used the HTTP requests method in the web application I have implemented, since requests are not language-dependent, unlike astroquery package.

1.1 Mikulski Archive for Space Telescopes

Mikulski Archive for Space Telescopes, also known as MAST, is one of the three databases used for queries in this thesis. MAST is NASA's astronomy data center which provides access to most of its collections with data from large past or active space missions as well as data for planned missions. As of March 2014, the total volume of MAST's data holdings was approximately 296 TB, with an average of 18 TB

of data downloaded per month [3, p. 1]. Mast provides three main approaches on how to obtain the required data – MAST portal, Python astroquery package, and direct HTTP requests.

1.2 Centre de Données astronomiques de Strasbourg

Centre de Données astronomiques de Strasbourg, abbreviated CDS, provides access to two remaining databases used in this thesis. Both of the services mentioned in the following subsections can be queried using portal on CDS website, as well as Python astroquery package and URL queries (HTTP requests).

1.2.1 Simbad

The first database is SIMBAD, Set of Identifications, Measurements and Bibliography for Astronomical Data. Simbad provides more general information and basic data about astronomical objects, including a main identifier and its aliases, coordinates in several formats (frames), bibliography overview and fluxes (magnitues). According to information from the CDS website, in June 2020, Simbad contained approximately 5,800,000 stars and 5,500,000 nonstellar objects, such as galaxies, planetary nebulae, clusters, novae and supernovae, among others [4].

Simbad can be used to obtain only a limited amount of information about queried objects coming from various kinds of instruments (thus should not be called catalogue) in a short and structured format, thus Vizier is advisable for getting more complex and rich data from its large set of astronomical catalogues. C. Jashek defined astronomical catalogues as:

> *An collection of a large number of observations of a certain kind made for a certain purpose and carried out at a certain place and time.* [5].

1.2.2 Vizier

Vizier is a service that provides a library of published astronomical catalogues, including tables and verified data associated with them.

Vizier provides a significantly higher and more complex amount of data and information in a vast list of catalogues and their tables. According to website information actual to April 2021, 20,829 catalogues are available with 22,760 tables associated with these catalogues [6].

2 Analysis of Requirements

This chapter introduces and analyses both functional and non-functional requirements based on the first consultation with the thesis consultant and the thesis assignment.

2.1 Functional requirements

Functional requirements define intended behaviour and usable functionality - what should and what should not be included in the developed product. The Following subsections summarise functional requirements for the developed web application for this thesis.

2.1.1 Search input

Search input of my web application can be divided into two categories: primary query parameters, which are compulsory, and secondary query parameters, which are optional and affect the size of the output.

Primary query parameters

There were three types of queries which were required to be implemented, based on the parameter distinguishing it from other types of queries. These distinguishing parameters were:

- Identifier
- Coordinates (decimal ICRS format)
- File with coordinates (Crossmatch)

The user had to be able to choose which type of query would be used and a natural need for a use of the regular expressions and values boundaries has risen, whose aim is to filter invalid inputs in order to prevent failed queries to occur before they are executed.

The format of identifiers was not closely specified, but identifiers containing only numbers were considered to be invalid.

Coordinates of astronomical objects might come in two main forms, which are:

- Decimal (in degrees)
- Sexagesimal (degrees/hours, arcminutes, arcseconds)

For the purposes of the web application, decimal coordinates were agreed to be used as input format.

Crossmatch query is a type of query, where two catalogues are joined, thus measurements and data from each catalogue are put into pairs that correspond to the same astronomical object, using its coordinates (Right Ascension and Declination) and a radius, which defines a distance between the first pair of coordinates (1st catalogue) and the second pair of coordinates (2nd catalogue), while these coordinates are considered matching. In other words, crossmatch is a service used for identifying rows from different tables that refer to the same object.

As for the crossmatch input file, the file must contain the row which defines the names of the columns for right ascension and declination and rows with two values - coordinates.

Secondary query parameters

Secondary query parameters directly affect two parts of the query:

- Output size
- Catalogues queried

Output size is an important aspect of searching, since the user does not need several thousands of results most of the time, thus being able to limit the size of the output is a convenient option to have.

As stated in the assignment, the user had to be able to choose which catalogues would be queried, thus functionality that provides this had to be implemented as well.

2.1.2 View

As for the representation of results of queries, they should have been listed. Combined with the fact that most of the responses are obtained

in VOTable format, tabular representation has been a natural and convenient offered solution. Additionally, JSON format contains tabular data, thus a tabular representation for this variant has been a satisfactory solution as well. However, it is worth of mentioning, that not all of the query results necessarily need to be in tabular form. A representative example of this case is query by identifier within Simbad service. Even though a VOTable response is for tabular representation, after closer examination the observer can see that only one table with only one row has been contained in the response, thus transforming this into a table is not the clearest way of representation and opens a possibility for custom representation of the response data.

The thesis consultant has not had any preferences for the graphical design of the web application since the very beginning, thus it has been completely up to me as the creator of the web application.

2.2 Non-functional requirements

This type of requirement is concentrating on how the developed product should be done. There were four main requirements:

- **sustainability** - ability that ensures the developed software will be available and working in the future, not only for a short amount of time
- **maintainability and manageability** - ability to implement new features quickly and easily, as well as keeping the application functional and working
- **portability** - ability to transfer software into different environments without losing any functionality
- **usability** - application that is easy to work with, user-friendly and intuitive

3 Initial state

As mentioned earlier, three internet databases and catalogues are used in this thesis - MAST, Simbad, and Vizier. Each of them provides an option to query using the object's identifier, coordinates (position), along with database-specific queries like queries by object's reference (bibliography), astronomical mission, and others. However, each of the common searching criteria requires a different approach in each of the mentioned databases, as the query is processed by different servers. The following sections describe how the proper URL request should be created for each server as well as providing basic information about responses provided by servers.

3.1 MAST queries

MAST provides concise and precise API documentation which describes a creation of valid MAST query requests. All services are usable for queries with brief information including its parameters, as well as examples of usage written in Python programming language.

MAST request URL is simple and straightforward, based on the fact that only one parameter is required - the MAST Request object given in JSON format.

3.1.1 MAST request object

MashupRequest object, as mentioned in MAST documentation [7], contains several fields, from which the most are optional or can be omitted (taking MAST default values into account), but there are two specifically important and required among them: `service` and `params`.

Service, as the name suggests, defines a service that will be queried for data. Among all of the services that can be omitted, only three were needed in my application, since other services are mostly derivatives from these three main services and provide a certain subset or filtered data.

Three main services are:

- Mast.Caom.Cone - query by coordinates (Figure 3.1)
- Mast.Name.Lookup - resolves an object name into position (which is used in cone search)
- Mast.Caom.Crossmatch - crossmatch with all MAST data

Parameters for request, contained in 'params', vary for each service, but all the parameters needed for a given service are described in MAST documentation.

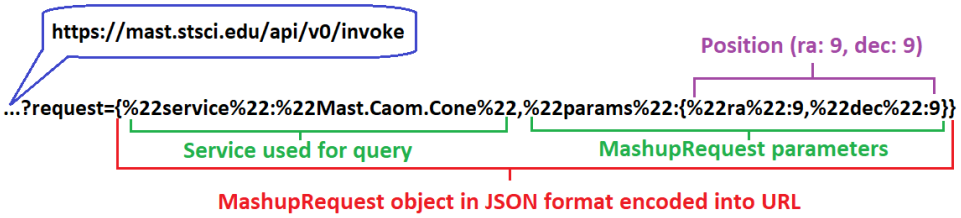

Figure 3.1: Example of Mast query URL

3.2 CDS queries

CDS covers usage of both Simbad and Vizier, however, crossmatch with Simbad or Vizier is done via a separate service called 'x-match', thus it is a standalone subsection in this section.

Each of the following services has its own documentation or manual available on the given service's website with a different set of parameters, so each of the services has parameters described separately.

Since CDS does not provide a response in JSON format but allows a response in HTML, ASCII, and VOTable formats, VOTable format has been chosen as it is the most convenient option in terms of parsing. This choice is supported by the fact, that CDS does provide their libraries for working with VOTable format[1].

1. https://cdsweb.u-strasbg.fr/cdsdevcorner/savot.html

3.2.1 VOTable format

The VOTable format is an XML standard for the interchange of data represented as a set of tables [8, ch. 1].

3.2.2 Simbad

Simbad service provides a short manual[2] on its website with basic examples and instructions on how to create a valid URL request for each type of query, whether it is an identifier, coordinates or other types, that were not part of this thesis.

Unlike MAST URL queries with a single request object, Simbad URL contains several separate parameters, depending on the type of specialized query that is used, like shown in Figure 3.2.

Response contains only general overview and fundamental data about queried astronomical object (IDs, coordinates, fluxes, basic measurements), while Vizier contains vast amount of astronomical catalogues. Simbad and Vizier are supposed to be used as complementary tools.

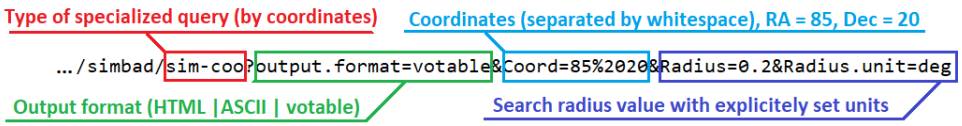

Figure 3.2: Example of Simbad query parameters

3.2.3 Vizier

Vizier service provides access to more than 20,000 catalogues, thus the source must be defined with previously mentioned parameters like position or id, which are both defined by '-c' parameter.

Vizier portal executes a full-scale search for given id or coordinates by default, which is a useful option if a user does not know which specific catalogue should be used for a query. According to Vizier

2. http://simbad.u-strasbg.fr/guide/sim-url.htx

manual [9], this type of search is selected by setting '-source=' parameter value to blank, instead of specifying one or more abbreviations of available catalogues.

3.2.4 X-match

CDS X-match service allows a user to do a crossmatch search of uploaded file with either Simbad or one of the Vizier tables. This service allows a crossmatch between Simbad and Vizier, or two Vizier tables as well, but these two cases were not a part of this thesis.

Compared to the previous two CDS services, the x-match service has a higher amount of compulsory parameters listed in its documentation[3], including a specification of which catalogues should be used, along with names of columns with RA and Dec values in case of an uploaded file.

3. http://cdsxmatch.u-strasbg.fr/xmatch/doc/index.html

4 Design

This chapter is devoted to design of my web application - design methods, resources, components used.

4.1 Agile Development

Agile software development is a development method, in which software is delivered to the customer quickly, starting in the early stages, often, after short periods of time, and constantly evolving, reevaluating and changing its priorities and requirements definitions, to provide a dynamic and adaptable way of ensuring the customer's satisfaction.

On the contrary, plan-driven development is often used for large and complex systems development, where extensive analysis is needed before design and implementation itself, and the process is divided into separate subsequent stages, rather than cycle in agile methods.

Since this thesis was a small project with a significantly smaller amount of time available for the development with a need for constant feedback, the agile development method has been chosen. Following the Agile Manifesto's [10] third principle:

> *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*

Regular meetings were set up for every week with the thesis consultant to discuss the implementation of a new functionality as well as improving already provided software. Regular communication prevented a large amount of software to be changed as time advanced with imperfections and problems being detected at the very beginning. This has resulted in a faster, more transparent development in the end, as expected.

4.2 Thymeleaf

There were several possible choices in the matter of web-pages generating, including JSP[1], Free Marker, and Thymeleaf. Any of these

1. Java Server Pages

three was a viable option, but Thymeleaf has been chosen to manage dynamic content of pages.

Thymeleaf is a modern server-side Java template engine for a maintainable and transparent way of creating HTML templates, among others. Its exhaustive manual [11] provided all the required information to build valid HTML templates that can be used in my web application.

4.3 Spring Boot

For a fast delivery of the very first version of a working web application, a suitable framework had to be chosen. Apache Tomcat was one of the possible options, but Spring Boot has been picked, since it provides a quick environment initialization and creation of a project with already working, clean web application without any functionality, but with no more configuration needed to start.

4.4 Queries

Several basic java classes were designed to provide a starting point for the implementation of queries. These classes could be divided into three categories:

- **engine classes** - these classes would handle choosing the type of search and executing it, as well as providing obtained responses in the form of return value, their methods would be used by controllers

- **request classes** - these specific classes would provide default methods for creating and sending a request for each service (database/catalogue), implemented methods would be given by abstract super-class for these request classes

- **response classes** - these would serve for storing the obtained data and provide methods for easier work

4.5 User Interface

There were no explicit requirements in terms of graphical design, but one of the key non-functional requirements was usability. This implied a design of both input form and results representation to be user-friendly, intuitive and simple, with the smallest amount of text-fields and other elements possible.

Based on these assumptions, a basic form composed of three elements was designed, to determine key search parameters - search type, text input (both id and coordinates, with radius optional), and file input.

For the results, as mentioned before, tabular forms of representation were chosen, thus only simple functionality for changing the source of displayed results (current table) was needed, which, in terms of web application, implied the use of JavaScript code.

5 Implementation

This chapter describes the whole implementation process, from its beginning, with all the problems I have encountered and had to solve, as well as steps taken towards optimization and overall improvement of developed web application - AstroSearcher.

The server-side part of AstroSearcher is written in Java 11, which was convenient option after consideration of the fact, that my web application must be runnable and working on more than one platform. Java-based applications can be run anywhere, where JVM[1] is included and running, which includes OS like Windows, MAC OS, and Linux.

JVM executes compiled java bytecode in a form of JARs[2], which include compiled java code packed together with all the resources needed for running, like images and HTML templates. JVM is obtained as a component by downloading Java Runtime Environment (JRE) along with core java classes and libraries.

As for the user interface, the web application uses dynamic HTML templates enriched by Thymeleaf, with a portion of CSS[3] and JavaScript.

5.1 Project Structure

The project is divided into two main groups - java files and resources, which include HTML templates, CSS and JavaScript files, and images.

Java files are further separated into several packages, based on their function, as displayed in Figure 5.1.

The important thing to note is, the `View` part of the web application is completely handled by a combination of dynamic HTML templates, CSS and JavaScript, whereas the `Controller` and `Model` classes are handled by java files. The rest of the java packages are directly connected to search functionality, further divided by which database and service they are used.

---

1. Java Virtual Machine
2. Java archives
3. Cascading Style Sheets

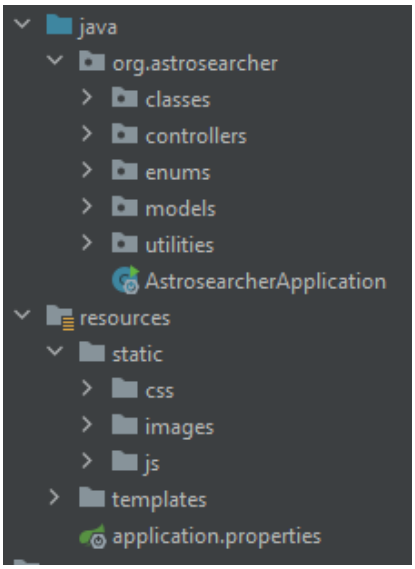

Figure 5.1: Main project structure

5.1.1 Maven

As for dependencies and repositories management, two options were considered - Maven and Gradle. My personal experience include using Maven exclusively, thus I have decided to use it in AstroSearcher as well to eliminate problems caused by using a new tool. However, gradle would be another viable option.

5.2 Application Architecture

As could be seen in the previous section, the architecture chosen for my web application is MVC[4] architecture, thus an application control is divided into three separate blocks.

5.2.1 Model

The model part of an application represents data, that the user works with. In the context of my web application, the model part represents

4. Model-View-Controller

data transferred between view and controller parts, that are used even further in astronomical queries. Model classes in my web application are:

- **SearchFormInput** - all the data obtained from HTML form that are used for queries

5.2.2 View

The view part of an application handles User Interface (UI) logic, thus it may include elements like textboxes, checkboxes, drop-down menus, and more (GUI). Since my application is a web application, the view part consists mostly from HTML with Thymeleaf, CSS and Javascript files and is fully described in following section 5.3.

5.2.3 Controller

Controllers handle all the requests and corresponding responses and interact with the business-logic part of an application. In the context of my web application, I have implemented several controllers (Table 5.1), each for handling requests from one page.

Table 5.1: AstroSearcher Application Controllers

Group	Controllers	Description
Input	*IndexController*	home page, processes search input
Info	*AboutController*	"about" page (basic information about application)
Results	*ResultsController, MeasurementsController, IdentifiersController*	results from several types of queries (ResultsController - search form input queries)

5.3 Graphical User Interface

Most of the applications require implementation of a GUI (Graphical User Interface), since the vast majority of users do not use command line interface and are used to a graphical approach, which provides much more intuitive, well-readable, easy-to-learn and transparent user-friendly environment. This is amplified even more with a development of web application, which is mostly built on graphical representations in a form of HTML pages (templates) combined with a reasonable amount of CSS and JavaScript files.

From the beginning, there was no external graphical design library used for a design of my application, thus I have designed all the CSS files on my own. The reason was based on the fact, that the output of my application would be large tabular data, which is not suitable for browsing on a small screen, thus no frameworks like Bulma or Bootstrap were used. The first search form for getting basic search parameters to query MAST contained only three input elements and submit button, however it was quickly proven, that major adjustments had to be made.

5.3.1 Default search form enhancements

Starting with MAST, but progressing throughout all the queried databases, a need for higher query customization was getting more and more noticeable.

Size

This option has already been implemented for querying MAST, which had its output size set to 1000 by default, which was a sufficiently large amount for most cases, but sometimes needed to be increased or even decreased, to get only a basic set of results. Implementation of this limitation included adding the two new input elements into the search form:

- **Page size** - as the name suggests, this option sets the limit of total results obtained and displayed

- **Page** - this option needs to be set together with the page size and it does exactly what the name suggests - if the pagesize sets the limit of characters on a book page, page parameters lists the given page. This option works only for MAST, which supports this.

Vizier catalogues

Vizier executes full-scale search by default when using its portal for queries. This has proven to be a highly restrictive solution in terms of performance, since the user does not always know which catalogue should be used for queries.

As a result, two new inputs have been added into the search form:

- **Vizier input type** - drop-down menu, which offers two options:
  - code - input acquired from the following textfield contains code(s) of Vizier catalogues
  - keywords - following textfield contains keywords which should be used for queries (different type of query parameter is used in a resulting query)
- **Vizier catalogues input** - text field, used for obtaining code of one or more Vizier catalogues, or a keyword

Checkboxes

The last enhancement to the search form has been adding three checkboxes to determine, which database should be queried or not. These inputs were compulsory and requested by the thesis assignment, but implemented as the last part of the search form, since they meant only insignificant changes to the implementation and all three databases were needed to be queried most of the time.

5.3.2 Output

Graphical design of the output underwent several extensive changes, mostly on tabular basis (rearranging/renaming of columns, data presentation format exchanging, and more), however the most noticeable redesign came in terms of Simbad queries by the identifier.

Even though a response for this type of query is in VOTable format (like for the rest of the Simbad queries), this particular response can easily be transformed into a significantly more readable format by extracting important information from the response by selecting only specific data based on the column IDs or names, since only one row of a table is obtained.

5.3.3 Bootstrap

As the time advanced, using Bootstrap with a small portion of custom adjustments for graphical design was being reconsidered.

There was one primary reason to do it - Simbad queries by object's identifier, which were unique in terms of representation possibilities, which was described in the previous subsection 5.3.2. This was a motivation to completely throw away the current CSS design and rework the graphical side of the application to use Bootstrap.

The rest of the responses came in purely tabular formats often with ten or more columns, which are not displayed and readable ideally on screens with small resolution like mobile phones, but Bootstrap reduced this problem to an acceptable level.

5.4 Java libraries

External libraries have played an important role in the implementation of my web application and I have described them in this section.

5.4.1 Lombok

Project Lombok is a Java library that improves and accelerates the writing of Java code. A main benefit of this library lies in introducing several annotations, that allow a developer to create classes with their getters (@Getter), setters(@Setter), constructors(@*Constructor), and more, without having a need to write them on their own.

Adding specific annotation causes that corresponding method(s) is created and usable exactly in the same way as if I have implemented those methods. Lombok has brought significant transparency to the Java code by making it shorter by hundreds of code that I would have to write otherwise.

5.4.2 Gson

Json format belongs to the most common formats used for data packaging and exchanging and usage of this format was a key part of the implementation of my web application as well.

There are several implementations for work with Json format, including the two most common:

- **Gson** - implementation provided by Google, provides basic methods, which are easy to work with
- **Jackson** - more complex implementation, providing higher amount of annotations

For the purposes of my web application, I have chosen the Gson implementation, since only a small portion of functionality was required to be used, but both of the mentioned variants were viable.

5.4.3 Spring

As mentioned in the Design - chapter 4, Spring is widely used in the AstroSearcher application. AstroSearcher runs on Spring Boot which is part of the Spring Framework, thus AstroSearcher is launchable from the command line, after being installed into a single JAR file with Maven.

5.4.4 SAVOT

This external library is provided by CDS[5] and stands for *Simple Access to VOTable*. As the name suggests, the library provides necessary functionality for working with data in VOTable format. Considering the fact, that this format is widely used in astronomers community (CDS including), this library contains four important packages (as shown in table 5.2) and is a core part of implementation for processing of obtained data from servers.

5. https://cdsweb.u-strasbg.fr/cdsdevcorner/savot.html

Table 5.2: SAVOT packages

Package	Description
cds.savot.common	general functionality (for all modes)
cds.savot.model	VOTable model (memory management)
cds.savot.pull	VOTable parsing
kxml2-min	XML parsing

5.4.5 AstroCC

This library provides a wide range of functionality for different astronomical calculations, however, only trasformation from ICRS coordinates to galactic coordinates is used in my web application, since only ICRS coordinates are provided in VOTable format of response.

5.5 Queries

Queries represent the absolute essence of my web application, thus substantial attention is paid to them in this section. Functionality that handles queries and response parsing is divided into three main parts, each described in a separate subsection.

5.5.1 Search Engine classes

These classes serve as an interface for executing queries and are directly used by application controllers.

5.5.2 Requests

Classes with suffix `RequestObject` represent all the parameters, data, and additional information needed for sending a valid request. Each of the request subclasses extends an abstract class `RequestObject` that declares three public abstract methods, shown in the Table 5.3.

Table 5.3: public abstract methods from abstract class RequestObject

Method	Description
String send()	sending a request, returns response data as String
URL getConnectionURL()	returns URL to establish a valid connection
byte[] getParamsAsBytes()	returns all the parameters as bytes array

5.5.3 Response

Classes with suffix `Response` represent all the required data parsed from the response from the corresponding service. Response classes handle parsing and processing of an obtained String response into an object that is handed into an instance of general `ResponseData` class by `SearchEngine` class (as shown in Diagram 5.2, which shows query by ID in a simplified way). This general response contains data from all three primary services queried.

Instances of `ResponseData` class, filled with corresponding responses from each service, is returned to the controller that called the search method from the `SearchEngine` class.

The controller subsequently processes `responseData` and sets proper attributes of an instance of *Model* class (required by controller as a method argument). Instances of *Model* class is then directly used in Thymeleaf blocks inside HTML templates to fill the template with data, thus create dynamic content.

5.6 Encountered problems

There were several problems encountered while implementing the AstroSearcher application. Each of these problems is described in the following subsections together with the solution used.

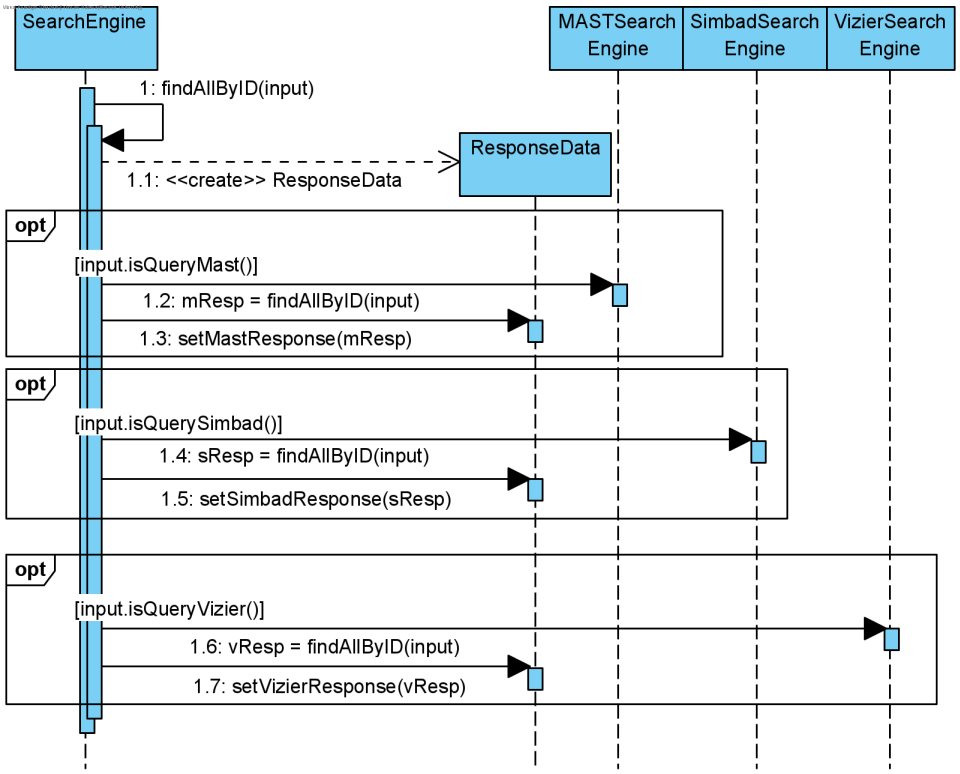

Figure 5.2: Query by ID (Sequence Diagram) after `process(input)` was called by controller on the `SearchEngine` class

5.6.1 Invalid data URLs

The first major problem that occurred were invalid data URLs in MAST results. These URLs were used in my application as icons working as links for downloading the data. After closer examination, I discovered, that the problem was caused by MAST, not by my application corrupting obtained data.

I contacted MAST about the problems found and after proper discussion, these were solved. However, during continuous testing of my application, several astronomers and I had found more invalid URLs, thus communication between me and MAST happened on a more regular basis.

My application helped and is still helping to detect these problems with MAST data, thus proving its value and significance.

5.6.2 External JARs

Soon after finishing the first implementation of MAST queries (excluding cross-match), work on Simbad begun. As mentioned earlier in subsection 5.4.4, the external library provided by CDS is used for parsing VOTable response.

This library is not in a central repository, thus I had to obtain it directly from the CDS website. This started to cause problems immediately, since external JARs must be added to project into the `pom.xml` file to work properly. This was done by setting the relative path inside the project folder to each JAR file and solved the problem while running my application on my computer.

However, after trying to launch my web application on a virtual machine, I discovered that the previous solution was not meeting portability requirements, since the application could not be built.

To solve this issue, I have created a local repository inside my project folder, installed all the external JARs with Maven `deploy:file` command and added a new repository (newly created) into the `pom.xml` file and set a relative path to it.

Deploying JARs into the project local repository caused Maven to install these JARs into the computer repository in the same way as it would install it for JARs in the central Maven repository, thus solving the problem with portability.

5.6.3 J-entries

An additional complication was detected when multiple astronomers tested my application and tried to use J entries, that were not working. This type of entries are composed of two parts - a preceding letter representing the equinox used (J for standard equinox) and coordinates themselves, for example: *J195251.15+403621.4*.

The solution for this problem was to create a special regular expression for detecting this type of input and simply change the search type.

5.6.4 Maximum queries per second

Based on the documentations mentioned in chapter 3, one specific issue had to be solved. All the services have a specific maximum limit for queries per second defined and when this limit is exceeded, the IP address is black-listed for a short amount of time and the user is not able to obtain any additional information.

Before describing a solution, I would like to state, that this situation has never been encountered during the time, when my application has been running.

In terms of a solution, a one-second long time period (1000 milliseconds) was divided by the corresponding number of maximum queries per second (Qps) decreased by 1 for each service (Equation 5.1) - this value is called *timeQuantum* in the following text. The corresponding `SearchEngine` class for every main service has been extended by:

```
private static boolean timeQuantumUsed = false;
```

which serves as an indicator whether the time quantum (window) is free or taken.

$$timeQuantum_{\text{service}} = 1000/(Qps_{\text{service}} - 1) \tag{5.1}$$

The reason why *Qps* had to be decremented is visualised in Figure 5.3. Assume $Qps_{Simbad} = 5$, thus $timeQuantum_{Simbad} = 250\ ms$. Additionally, assume that the first query is executed on time 0.24sec, the next three queries 0.25sec one after another and the fifth query 0.02sec after the fourth one.

When the times of execution of the first and the fifth queries are subtracted, we can see that 5 queries were executed in less than a second, thus the maximum allowed amount of queries was reached.

In order to achieve that all required services would be queried even though timeQuantum was taken, method calls were put inside a `while` block. Each iteration verifies whether a service should be queried and whether *timeQuantum* is free, if so, `timeQuantumUsed` is set to true and the service is queried. At the end of an iteration, if-block verifies whether there is anything more to query and if needed, `wait()` is used until `notify()` is called by scheduled methods.

Scheduled methods are annotated with `@Schedule` annotation and for the needs of my application, `fixed delay` was set using calcu-

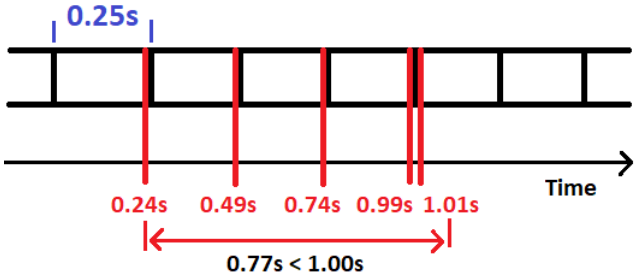

Figure 5.3: Time quantum usage - visualisation

lated *timeQuantum* for the corresponsing service. These methods set `timeQuantumUsed` to false, thus opening a time window and then `notify()` is used to awaken waiting threads.

5.7 Further Enhancements

After the main functionality was finished, several possibilities for improvement have been established, and I have decided to implement some of them, as listed below in following subsections.

5.7.1 Measurements and Identifiers

Previous Simbad queries in VOTable format had one major problem to solve, even though it was not required before. VOTable response did not contain measurements and object's other identifiers (aliases), which was confirmed by CDS in our communication as well. This problem was solved by two types of queries:

- **Sesame XML queries** - Sesame is the source of basic information mainly about object's aliases and fluxes, provided by CDS
- **Simbad ASCII queries** - ASCII format was not optimal in terms of parsing, however, I managed to implement simple ASCII parser to obtain needed measurements

5.7.2 Asynchronous queries

Previous solutions handling the amount of executed queries per second had one major problem - queries from one request were being executed sequentially and not all available *timeQuantum* windows were used, even though they could be free.

This problem could be seen (Figure 5.4) when the first request needed to execute long queries (longer than timeQuantum) and after a certain amount of time, the second request was being handled, however, the query might not be able to be executed, if the first request took the timeQuantum window, even though the first request could have executed all its queries at once with asynchronous methods.

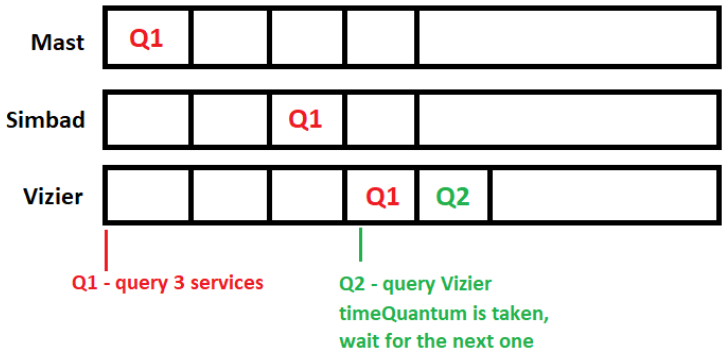

Figure 5.4: Problem with sequential execution of queries

This problem has been solved with usage of several annotations and keywords, such as `@Async`, `@Autowired` or `synchronized`.

The return type of asynchronous methods has changed from `T` to `CompletableFuture<T>` and a `while` block ensuring that all required queries are executed has been extended by a block, which ensures that all the queries are finished before proceeding with obtained results - with methods `allOf()` and `join()`.

5.8 Running the application

To run my web application, the server needs to have *Java Runtime Environment* installed and have a working internet connection for obtaining and presenting the data to the user.

AstroSearcher application is provided as an executable JAR file, which is created with Maven. Maven ensures that all required external libraries are included in JAR file, as well as resources, such as images, HTML templates, etc.

The JAR file is created (in local repository) by executing the following command inside the project folder:

```
$ mvn install
```

The following command is used to start the AstroSearcher application (in the background):

```
$ java -jar <target>/astrosearcher-1.7.4k.jar&
```

6 Further development

The current version of the AstroSearcher application provides only a limited amount of functionality available on the queried services' websites. This chapter introduces several possibilities for future development.

6.1 Queries

Simbad offers queries by object's reference, that results in a bibliography response, where given bibcode is used. A list of all object's references could be queried and displayed in my web application as well, to reduce the need for using external websites besides my application. This functionality would require certain changes in search form as well.

Finally, adding completely a new service for querying my web application is a large and demanding vision, nevertheless it is certainly possible.

6.2 GUI

Considering the communication with astronomers who tested my application, the feedback on the latest version of the GUI confirmed that it is satisfying and transparent, however thorough reworking would be convenient for mobile devices.

Web application is simple and usable on desktops as well as on mobile phones, however some aspects (including search form) might be completely re-designed for mobile devices to display different elements, in order to provide a better user experience together with preserving the core design of the web application.

6.3 Input

An additional extension could involve implementing a parsing of a wider range of formats of the input. Coordinates might be obtained in several additional frames (galactic), as well as provided in sexagesimal

format. Files could contain different column headers and coordinates could be in sexagesimal format, for instance.

Conclusion

The main goal of this thesis was to implement a web application for astronomers, that would allow them to query multiple astronomical catalogues and databases in one place.

Within the thesis, I introduced and analysed three required services and defined functional and non-functional requirements for the application. The design chapter described what tools would be used for implementation, as well as discussing the structure and basic functionality required in my application. In the implementation chapter, I provided details about implementation, encountered problems, enhancements and running the AstroSearcher application.

AstroSearcher is a multi-platform web application containing all the required functionality, which aims to provide a simple and transparent way to use what is already available to astronomers. In addition, the application helped to reveal problems with invalid data in MAST. The application can be improved further by adding the queries of new services and other future development possibilities described in the previous chapter.

Implementation of the application is available in the AstroSearcher repositories at:

- `https://github.com/Kuliak/AstroSearcher`
- `https://gitlab.fi.muni.cz/xhalama/bp-astrosearcher`

Bibliography

1. BRITANNICA, The Editors of Encyclopaedia. *Right Ascension* [online]. Encyclopedia Britannica, 2019-03-08 [visited on 2021-04-15]. Available from: https://www.britannica.com/science/right-ascension.
2. BRITANNICA, The Editors of Encyclopaedia. *Declination* [online]. Encyclopedia Britannica, 2019-03-08 [visited on 2021-04-15]. Available from: https://www.britannica.com/science/declination.
3. KOEKEMOER, Anton. *Barbara A. Mikulski Archive for Space Telescopes* [online]. Mikulski Archive for Space Telescopes, 2014 [visited on 2021-04-15]. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.431.6132&rep=rep1&type=pdf.
4. *SIMBAD: Introduction* [online]. [N.d.] [visited on 2021-04-14]. Available from: http://simbad.u-strasbg.fr/guide/simbad.htx#s4.
5. JASCHEK, C. Astronomical Catalogues - Definition Elements and Afterlife. *Quarterly Journal of the Royal Astronomical Society*. 1984, vol. 25, p. 260. ISSN 0035-8738.
6. *VizieR* [online]. [N.d.] [visited on 2021-04-14]. Available from: https://vizier.u-strasbg.fr/index.gml.
7. *MAST API Documentation* [online]. [N.d.] [visited on 2021-02-25]. Available from: https://mast.stsci.edu/api/v0/.
8. *VOTable Format Definition Version 1.4* [online]. 2019-10-21 [visited on 2021-04-21]. Available from: https://www.ivoa.net/documents/VOTable/20191021/REC-VOTable-1.4-20191021.pdf.
9. *The "vizquery" program* [online]. [N.d.] [visited on 2021-03-22]. Available from: http://vizier.u-strasbg.fr/doc/vizquery.htx.

10. KENT, Beck; MIKE, Beedle, et al. *Manifesto for Agile Software Development* [online] [visited on 2021-03-08]. Available from: https://moodle2019-20.ua.es/moodle/pluginfile.php/2213/mod_resource/content/2/agile-manifesto.pdf.

11. *Thymeleaf* [online]. 2018 [visited on 2021-02-26]. Available from: https://www.thymeleaf.org/doc/tutorials/3.0/extendingthymeleaf.html.

A Electronic attachments

This thesis is stored in the archive of the Information System of Masaryk University with the following attachments:

- Executable JAR file containing complete web application packed with all the resources needed. In order to run the application, Java 11 needs to be installed.