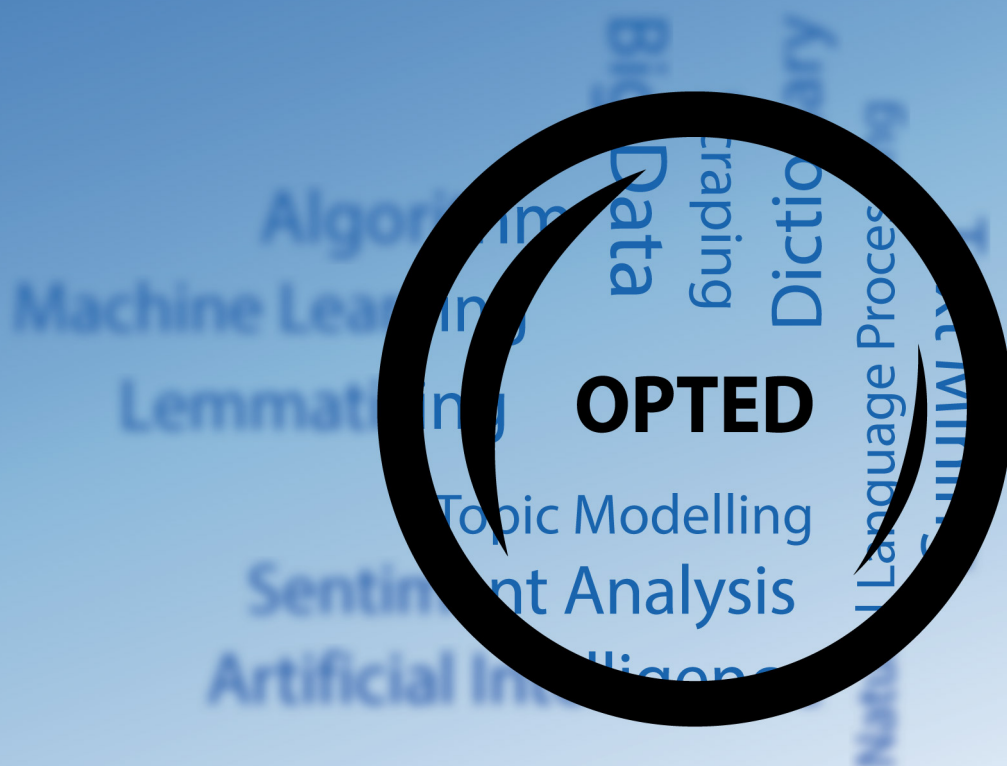


OPTED

Infrastructure for Text Storage

Wouter van Attevelde, Farzam Fanitabasi, & Kasper Welbers



Disclaimer

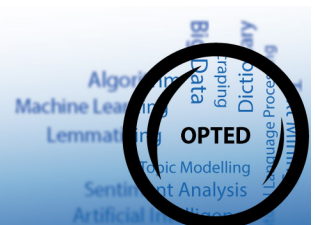
This project has received funding from the European Union's Horizon 2020 research & innovation programme under grant agreement No 951832. The document reflects only the authors' views. The European Union is not liable for any use that may be made of the information contained herein.

Dissemination level

Public

Type

DEC



OPTED

Observatory for Political Texts in European Democracies:
A European research infrastructure

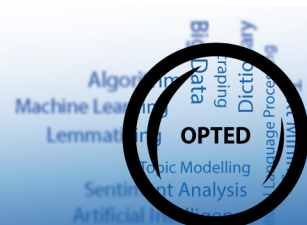
Infrastructure for Text Storage

Deliverable 7.1

Authors: Wouter van Atteveldt¹, Farzam Fanitabasi¹, & Kasper Welbers¹,

¹ Department of Communication Science, Faculty of Social Science, Vrije Universiteit Amsterdam

Due date: June 2021



Executive Summary

The overall objective of WP7 is to establish routines and protocols for varieties of standardizations of pre-processing, pending on source and purpose of usage of text. This work package focuses on assessing and providing prototypes of open science and open data structures in terms of data storage.

As the first step to achieve this objective, in D7.1, a text storage infrastructure is designed and implemented which makes it easier to store, share, and collaborate on political texts. This infrastructure enables and facilitates the access and processing of data that cannot be shared openly, by keeping data owners in direct control of the data, and by employing trusted connections and role-based access control methods to simplify collaboration among multiple users (e.g., political scientists). To this end, D7.1 introduces AmCAT 4.0, a re-designed and re-engineered version of AmCAT 3.0, providing simple, flexible, and modular architecture and codebase focuses on core functionalities, while allowing plug-in extra functionalities to be added.

This deliverable consists of the AmCAT 4.0 codebase (core engine + user interface), published via the OPTED website and publicly accessible on the GitHub repositories. Additionally, a functional and technical design document accompanies this deliverable, helping users understand and utilize the infrastructure as smoothly as possible.

1 AmCAT 4.0: A Streamlined Text Processing Infrastructure

In the last 20 years, the availability of user-friendly tools for text analysis has greatly increased. Many proprietary platforms like *LexisNexis* or *Coosto* allow basic text analysis directly on the dashboard. Moreover, R and Python based toolkits, such as *quanteda* and *scikit-learn*, have put very powerful analysis possibilities within the reach of a technically inclined social scientist. However, even with those advances we believe that a tool such as AmCAT can still play an important role for text analysis in the social sciences. AmCAT facilitates the storage of textual documents, to be later used for text processing and analysis. The benefits of using AmCAT compared to other existing tools include:

- An intuitive user interface gives a visual overview of the stored texts and **allows non-technical users to interact with the system**, without restricting its use to a single proprietary platform. This also makes it a **valuable resource in teaching** quantitative content analysis.
- A common storage format, and API makes it easier to **share and reuse** tools and analysis scripts. It also aids **reproducible science** by allowing analyses and data to be shared and validated based on immutable data sets.
- Ability to aggregate, and also to drill-down on the data, using graphs and other visualization tools. This enables researchers to gain insight into the textual data both at an aggregate (trend) level, and individual (text) level.
- Finally, as communication science data is often proprietary, fine-grained access control allows results to be shared and validated without giving access to the underlying data using **non-consumptive research**.

The previous versions of AmCAT have grown organically from a text storage application with a website user interface, to include the website, API, annotator, and the associated JavaScript files. As a result, the codebase is hard to maintain and secure, and can be challenging to install on a local computer. Thus, we now introduce a more modular codebase focused on the core functionality, while always allowing for other functionality to be added. In particular, we strictly separate the API and website, and split the website into separate modules for the query and annotation facilities. The goal of the redesign is to create an infrastructure that is simple and flexible enough to be adapted for various projects at OPTED partners (e.g., VU, UvA, Vienna, HUJI and others) and beyond.

1.1 Design Principles

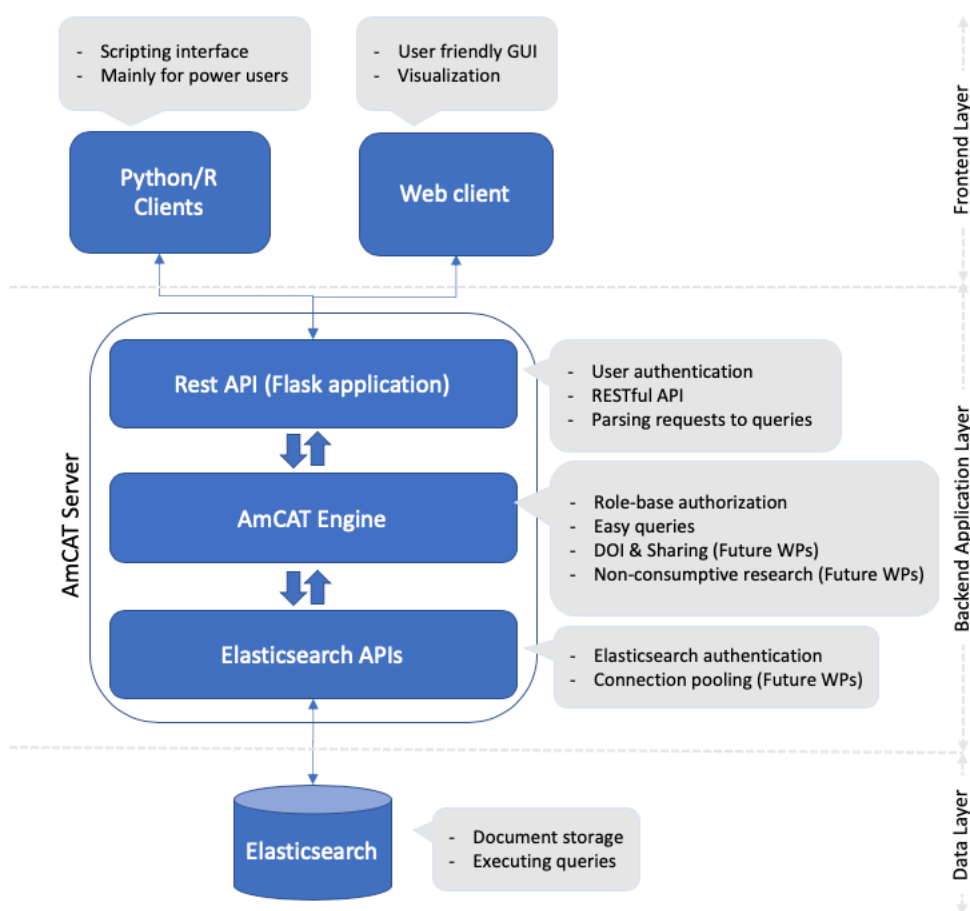
The design principles for AmCAT 4.0 are the following:

- **Minimalism and modularity:** The core components are as small as possible, do their specific task well, and allow easy configuration of multiple components.
- **Plug and play installability:** It is very easy install a new AmCAT server, either with a new or existing Elasticsearch index.
- **API:** All data are made available through an API that allows connection from other programs, both apps like additional annotation interfaces and analysis scripts in e.g. R or Python.
- **Minimal assumptions about the data:** We assume articles have an immutable text field and a set of (Dublin core) metadata (title, date, URL), that together uniquely identify a document. Users are free to use any other fields for additional metadata or annotations.
- **Open for integration:** With e.g. annotators, scrapers, NLP processing, etc.
- **Access control:** Users can determine who can access which parts of a project.
- **Non-consumptive research:** By giving query access and/or data capsule access without access to the underlying texts
- **Reproducible science:** by giving identifiers (DOI or URI) to articles and article sets that can be guaranteed to be unchanged.

1.2 AmCAT 4.0 Architecture

The architecture of AmCAT 4.0 includes the following modules. Note that few of these modules (e.g., Annotator, and Web Scraper) will be added in later OPTED work packages.

Figure 1.1 AMCAT 4.0 ARCHITECTURE



Note that in the AmCAT architecture (Figure 1.1), the communication between the frontend layer, and the core engine is done via a RESTful API. REST is an architectural style that uses simple HTTP calls for inter-machine communication. Benefits of using REST include scalability, as it scales very well since the client and server are very loosely coupled. Additionally, with REST, the server is free to change the exposed resources at will. There is no fixed API above and beyond what REST itself defines. The client needs only know the initial URI, and subsequently chooses from server-supplied choices to navigate or perform actions.

Regarding the two-level authentication (User and Elasticsearch), this design choice enables the separation of database-level and application-level authentication. On the application-level (user authentication), the AmCAT users can create authentication and access control schemes based on the requirements of their project, and not be limited by access-based protocols of the database. On the other hand, having the Elasticsearch authentication limits the number of users who can directly connect to Elastic, and can be used for connection pooling, database-level administrative tasks, and troubleshooting if users forget their credentials.

2 Download and Installation

AmCAT 4.0 and all its associated modules are freely and openly available through the OPTED website and GitHub repositories. The modules have their installation instructions described in detail in their README.me file.

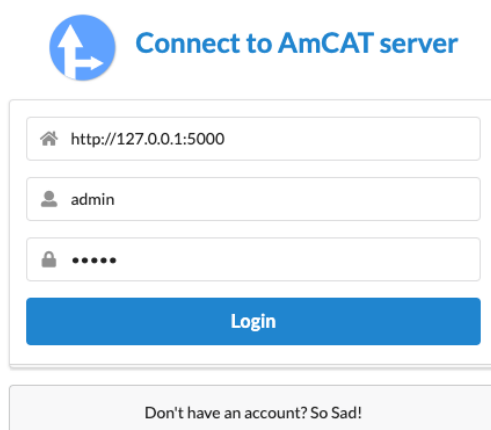
- AmCAT 4.0 Core: <https://github.com/ccs-amsterdam/amcat4>
- AmCAT 4.0 Client (web interface): <https://github.com/ccs-amsterdam/amcat4client>

3 Appendices

3.1 Appendix A: Screenshots of AmCAT 4.0 Webpages

Login page:

Figure 3.1 AMCAT 4.0 LOGIN PAGE



Connect to AmCAT server

http://127.0.0.1:5000

admin

.....


Login

Don't have an account? So Sad!

Home page:

Figure 3.2 AMCAT 4.0 HOME PAGE

Home Manage Indices Run Queries Browse Document Manage Users and Access Logout



Welcome to AmCAT4 (AKA AmKitten)

Hello admin

Here are your current settings:

Current Index:	state_of_the_union
Role over Index:	Current Role over Index: ADMIN
Last Query:	No Previous Queries
Remaining Tasks:	ToDo

Core Functionalities

Select and index:	Manage Indices!
Upload documents:	Manage Documents!
Run queries:	Run Queries on Index!
Manage users and their access:	Manage User Access!

AmCAT Plugins

AmCAT Annotator:	ToDo
Run NLP Methods:	ToDo
Non-consumptive Research:	ToDo

Index and document management page:

Figure 3.3 AMCAT 4.0 LOGIN PAGE

Home Manage Indices Run Queries Browse Document Manage Users and Access Logout

+ Create new index

Click to search results! 1 item(s) found

Index	Role
state_of_the_union	ADMIN

Query Documents

- Delete Index

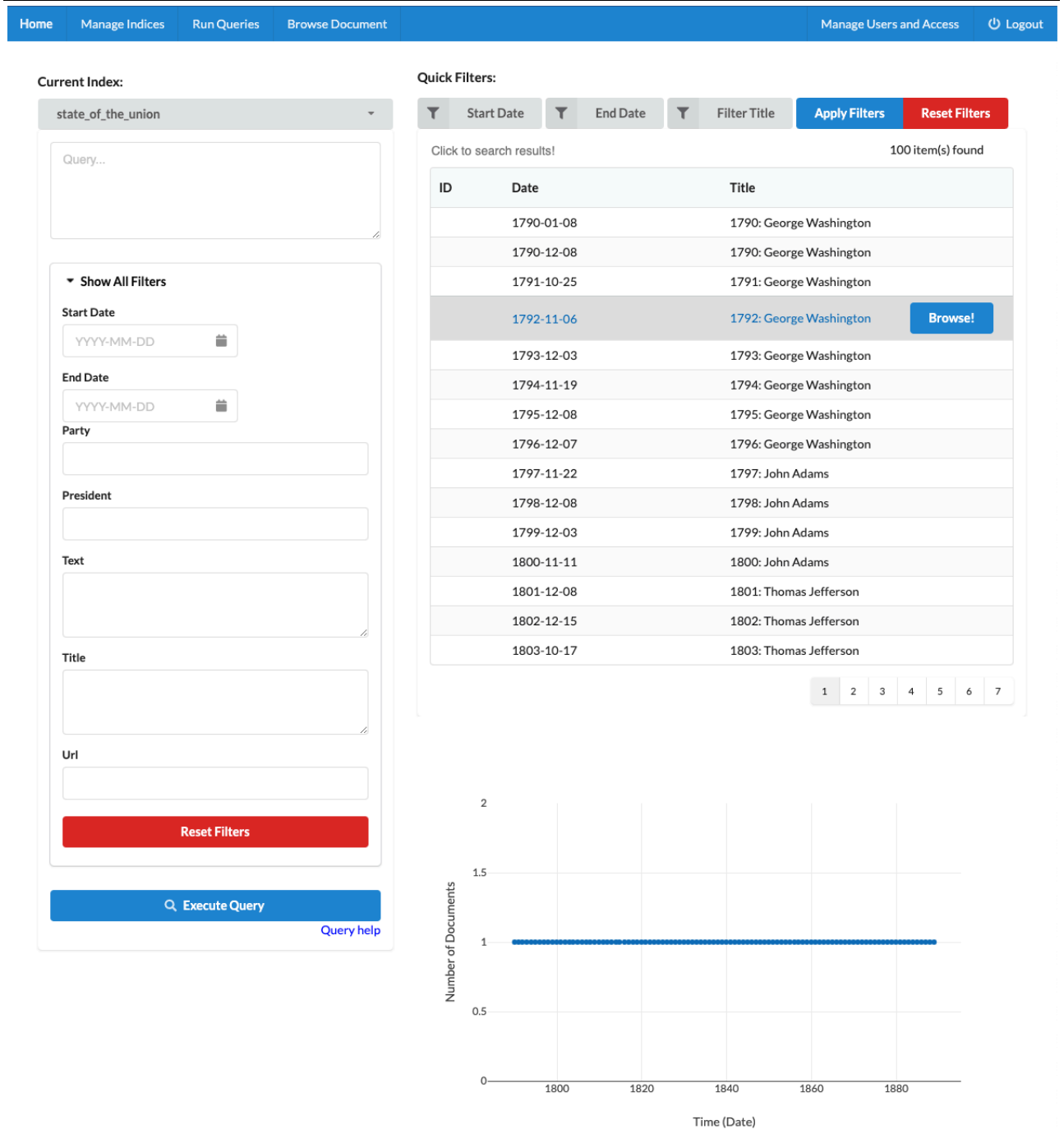
Index Details Upload Documents Create New Document

state_of_the_union

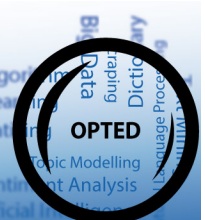
I am an index
And here should be some stuff that you can do with an index

Query page:

Figure 3.4 AMCAT 4.0 QUERY PAGE



Note that the filter fields are adaptive, and are shown based on the current queried index.



3.2 Appendix B: Functional and Technical Design Document

A continuously update version of this document can be found [here](#).

AmCAT4 Functional and Technical Design (version 23/06/2021)

Table of Contents

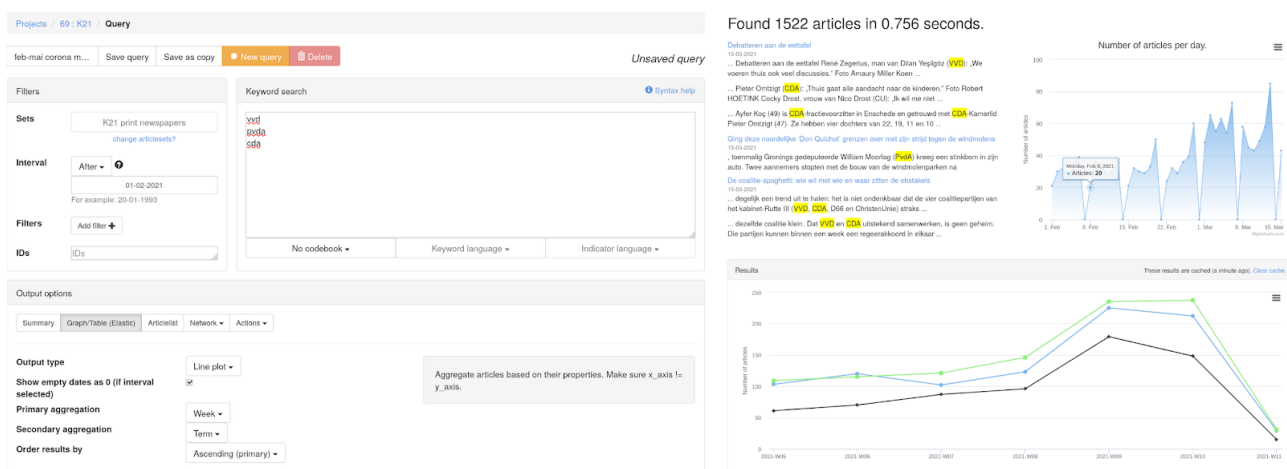
- 1 Functional design: what is AmCAT4 for?
 - 1.1 Introduction & overall goal
 - 1.2 Why a new version?
 - 1.3 Requirements
 - 1.4 Integration between projects and programs
- 2 Technical design
 - 2.1 Module Description
 - 2.1.1 Document Storage/Database
 - 2.1.2 AmCAT Server
 - 2.1.2.1 AmCAT Engine
 - 2.1.2.2 Elasticsearch API
 - 2.1.2.3 RESTful API
 - 2.1.3 Frontend
 - 2.1.3.1 Web Client
 - 2.1.3.2 Python/R Clients
 - 2.2 Security, Privacy, and Access Control
 - 2.2.1 Security
 - 2.2.2 Privacy
 - 2.2.3 Access Control

1. Functional design: what is AmCAT4 for?

1.1 Introduction & overall goal

AmCAT has been in development in various guises since about 2001 as a text research / content analysis platform at the VU. From the start, the core has been a **database** of news articles and an intuitive **query interface** for performing quantitative analysis. In addition, an **annotation interface** allows for manual content analysis, while an **API** allows other programs and power users to connect to it directly from e.g., Python or R. The screenshots below (Figure A1) give some indication of the database and query interface of AmCAT 3.5.

Figure A1 AMCAT 3.5 INTERFACE



In the last 20 years, the availability of user friendly tools for text analysis has greatly increased. Many proprietary platforms like *LexisNexis* or *Coosto* allow basic text analysis directly on the dashboard, while R

and Python based toolkits like *quanteda* and *scikit-learn* have put very powerful analysis possibilities within the reach of a technically inclined social scientist. However, even with those advances we believe that a tool such as AmCAT can still play an important role for text analysis in the social sciences:

- An intuitive user interface gives a visual overview of the stored texts and **allows non-technical users to interact with the system**, without restricting its use to a single proprietary platform. This also makes it a **valuable resource in teaching** quantitative content analysis.
- A shared storage format, and API makes it easier to **share and reuse** tools and analysis scripts. It also aids **reproducible science** by allowing analyses and data to be shared and validated based on immutable data sets.
- Finally, as communication science data is often proprietary, fine-grained access control allows results to be shared and validated without giving access to the underlying data using **non-consumptive research**.

1.2 Why a new version?

The goal of the redesign is to create an infrastructure that is simple and flexible enough to be adapted for various projects at OPTED partners (e.g., VU, UvA, Vienna, HUJI and others) and beyond.

The current version of AmCAT has grown organically from an application focused on the website and includes the website, API, annotator, and the associated JavaScript files. As a result, the codebase is hard to maintain and secure, and can be challenging to install on a local computer. Thus, we propose a more modular codebase focused on the core functionality, but allowing for other functionality to be added. In particular, we will strictly separate the API and website, and split the website into separate modules for the query and annotation facilities.

1.3 Requirements

- **Minimalism and modularity.** The core components should do as little as possible, do that well, and allow easy configuration of multiple components.
- **Plug and play installability.** It should be as easy as possible to install a new AmCAT server, either with a new or existing elasticsearch index.
- **API.** All data will be made available through an API that allows connection from other programs, both apps like additional annotation interfaces and analysis scripts in e.g. R or Python.
- **Minimal assumptions** about the data. We assume articles have a text field and a set of (Dublin core) metadata (title, date), that together uniquely identify a document. Users are free to use any other fields for additional metadata or annotations.
- **Open for integration** with e.g., annotators, scrapers, NLP processing, etc.
- **Access control** so users can determine who can access which parts of a project.
- **Non-consumptive research** by giving query access and/or data capsule access without access to the underlying texts
- **Reproducible science** by giving identifiers (DOI or URI) to articles and article sets that can be guaranteed to be unchanged.

1.4 Integration between projects and programs

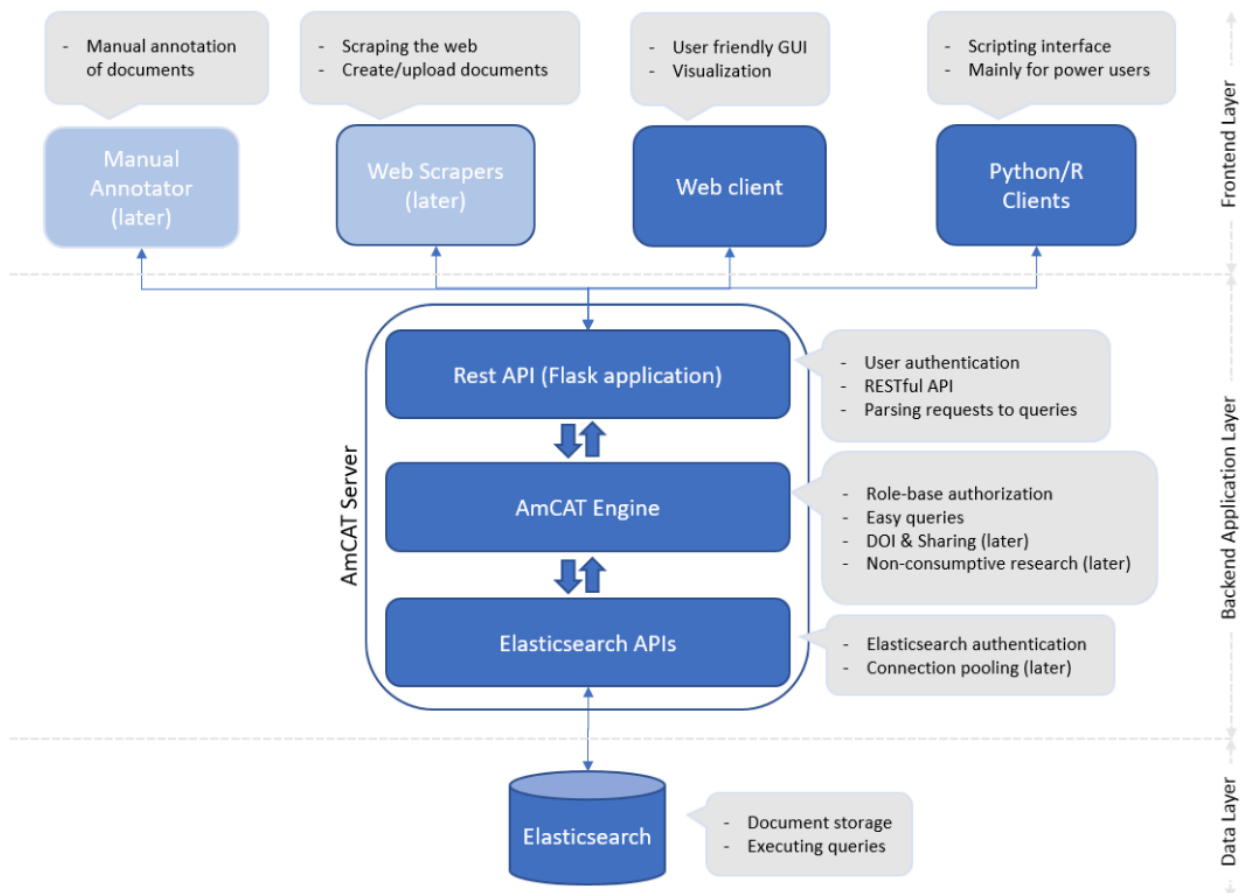
The express goal of AmCAT is not to be a single server that serves all and does all. Rather, it is intended to be an integral part of more complicated workflows. This requires AmCAT to connect both to other services and to other AmCAT servers.

- Connecting with other services, e.g., annotation, scraping, NLP analysis
- Connections between AmCAT servers

2. 2 Technical design

This section describes various technical design aspects and illustrates different modules of AmCAT 4.0 - codename “*AmKitten*” (hereon in AmCAT). The overall modular design of AmCAT and their relationship are illustrated in Figure A2.

Figure A2 MODULAR DESIGN OF AMCAT



While designing and implementing AmCAT, the following technical design principles were considered:

- **Lightweight Core Server:** AmCAT is designed and implemented to be lightweight while providing all the necessary functionality to the standard users. Additionally, AmCAT provides programming clients for power users who require extended and more complicated functionalities.
- **Limited but Extendable Feature Set:** By default, AmCAT provides a limited entity/feature set, including:
 - Documents containing title, date, set, and arbitrary other fields
 - Article sets with name, provenance, and field restrictions
 - Users having rights on the whole instance: admin, read/write, read, meta-read
- **Modular and Flexible Design:** AmCAT is designed in a modular manner to provide flexibility and facilitate the incorporation of new functionalities. This modular design makes it easier for users to tailor AmCAT to their needs and existing (legacy) software if they want to.
- **Open Science:** The code base, algorithms, and software artifact are all made available online for the community (more on Dissemination on Section 4)

- **Data Security and Access Control:** All the data (document) are securely stored on the database module and only accessible via the AmCAT server. AmCAT server also performs user authentication and authorization to make sure the users can only access the documents they have permission for.
- **Persistence and Reproducibility:** The performed analysis and experiments should be reproducible. Thus, the data and documents are stored in a persistent manner and each have a unique identifier.
- **Non-consumptive Research:** Due to possible confidentiality of the data, direct access and analysis might not be possible. To address this, AmCAT provides tools for non-consumptive research, whereby it executes users' queries (given appropriate access is granted) and returns the aggregated results to the users.

2.1 Module Description

This section provides an overview of the modules within AmCAT.

2.1.1 Document Storage/Database

The database module is responsible for storing the documents, and running certain queries on them. In AmCAT, the database is an Elasticsearch cluster, managing multiple instances. Only the users with the role "server admin" can create instances, upload plugins, and reset instance passwords. By default, each index is stored in one instance and replicated on another one (thus, the minimum number of instances is 2). This design enables replication among instances which in turn reduces the probability of data loss in case of disk failures, and improves the availability of the indexes.

Each document has 3 mandatory fields: Date, Title, and Text. These 4 fields are hashed together to create a unique identifier for each document. In addition to the mandatory fields, each document can have many optional fields, such as topic, writer, and annotations. These optional fields are mutable by the users. Accessing the Elasticsearch is solely by the AmCAT Server and no other direct connections are allowed.

2.1.2 AmCAT Server

AmCAT server acts as a middleware between the frontend (where users connect and communicate with the system) and the database (where the documents are stored). The server takes care of authorizing users' actions on the database, parsing users' requests, forwarding the authorized queries to the database, and performing non-consumptive queries. The backend of AmCAT consists of three sub-modules, as described below.

2.1.2.1 AmCAT Engine

The AmCAT engine is the main module in the AmCAT server. It is responsible for authorization of users' actions, as well as parsing users' requests, sending them to the database, and reporting the results back to the users. These user requests can for instance be creating an index, uploading documents, querying documents based on filters or topics, querying aggregated results based on document sets, or even deleting documents or indices.

2.1.2.2 Elasticsearch API

The Elasticsearch API module is responsible for communicating with the Elasticsearch database. It exposes specific APIs to correctly send and receive data from Elasticsearch. Due to the modular design of the AmCAT server, power users who opt to change/update their document storage module only need to modify this module, while the rest of the server remains unaffected.

2.1.2.3 RESTful API

The RESTful API module is in charge of communicating with the frontend of AmCAT (Section 2.1.3). It exposes APIs and endpoints for receiving queries, management commands, and security policies. This module is based on the Flask application library, providing lightweight and robust RESTful access to the AmCAT server. Similar to the Elasticsearch API, this modular design allows power users to create and set up their own frontend modules to communicate with AmCAT.

2.1.3 Frontend

The frontend module of AmCAT is where the users communicate with and access AmCAT functionalities. This module by default consists of three sub-modules, detailed as follows.

2.1.3.1 Web Client

The web client for AmCAT provides easy-to-use and intuitive interfaces for users to connect to AmCAT and run text analysis tasks. The provided functionalities include creating/deleting indices, user management, uploading/deleting documents, querying documents based on different filters, and visualizing the results of queries via line/scatter plots. This module is based on the React library, providing a lightweight and responsive graphical interface for non-power users. The web client communicates with AmCAT server via the RESTful API module.

2.1.3.2 Python/R Clients

By default, AmCAT provides two RESTful clients for Python and R to connect to the AmCAT server. Both these clients connect through the RESTful API to communicate with the AmCAT server. The provided functionalities include document upload/download, users and access management, as well as a querying/scripting interface for more complex tasks. These two clients can be used as plugins in other software, and are mostly intended for power users who want to use AmCAT either from within their software, or to run more advanced queries on the documents. A more detailed description of these two modules is presented on Section 3.1.

2.2 Security, Privacy, and Access Control

2.2.1 Security

The only point of access to the stored documents is through the AmCAT server, which performs both user authentication and authorization (Section 2.2.3). Additionally, during AmCAT setup the users can choose to limit and specify the permission of each account to provide additional security.

2.2.2 Privacy

In scenarios where the stored documents on AmCAT are subject to confidentiality or non-disclosure agreements, AmCAT provides different methods for guaranteeing privacy while enabling non-consumptive analysis and research on them. In summary, to do so AmCAT performs the request and queries (given appropriate permissions and authorization) locally on its server and only reports the aggregated results. This functionality will be added in work package 7.4

2.2.3 Access Control

The access control in AmCAT server is done at two points: (i) Authentication of users is performed by the RESTful API, either by entering username/password or providing an authentication token. (ii) Authorization of users' actions (such as document upload, download, or queries) are performed by the AmCAT engine using a role-based access control method. Both authentication credentials and authorization policies are set by the user of AmCAT during the installation, and can be modified during the runtime of the system.